Fast and Efficient Second-Order Method for Training Radial Basis Function Networks

Tiantian Xie, Hao Yu, *Student Member, IEEE*, Joel Hewlett, Paweł Różycki, and Bogdan Wilamowski, *Fellow, IEEE*

***Abstract*— This paper proposes an improved second order (ISO) algorithm for training radial basis function (RBF) networks. Besides the traditional parameters, including centers, widths and output weights, the input weights on the connections between input layer and hidden layer are also adjusted during the training process. More accurate results can be obtained by increasing variable dimensions. Initial centers are chosen from training patterns and other parameters are generated randomly in limited range. Taking the advantages of fast convergence and powerful search ability of second order algorithms, the proposed ISO algorithm can normally reach smaller training/testing error with much less number of RBF units. During the computation process, quasi Hessian matrix and gradient vector are accumulated as the sum of related sub matrices and vectors, respectively. Only one Jacobian row is stored and used for multiplication, instead of the entire Jacobian matrix storage and multiplication. Memory reduction benefits the computation speed and allows the training of problems with basically unlimited number of patterns. Several practical discrete and continuous classification problems are applied to test the properties of the proposed ISO training algorithm.**

***Index Terms*— Levenberg–Marquardt algorithm, radial basis function networks, second order algorithm.**

I. Introduction

BEING inherited from the concept of biological receptive fields, radial basis function (RBF) network was proposed in literatures [1], [2], used for multivariable classification and interpolation. Followed, Park and Sandberg proved that, RBF networks were capable to build any nonlinear mappings between stimulus and response [3]. With this property, RBF networks are broadly applied to solve practical problems, such as fault diagnosis [4]–[6], image processing [7], [8], and adaptive control [9]–[11].

The original RBF networks do not need training process, but as many RBF units as training patterns are required. This is not proper for solving practical problems where there are usually hundreds of training patterns. In order to achieve the similar approximation/classification accuracy with less RBF units, the training process was introduced for parameters adjusting. One of the oldest training algorithms for RBF networks design is linear least squares method (also known as "linear regression").

This method is simple, but it works only for output weights adjusting and performs poorly for nonlinear cases. Non-linear least squares [12] and singular value decomposition [13] enhance the nonlinear performance of output layer. Based on gradient decent concept, lots of methods [14], [15] have developed to perform "deeper" training on RBF networks because, besides output weights, more parameters, such as centers, and widths, are adjusted during the learning process. But first order gradient methods have very limited search ability and take a long time for convergence. Kalman filter training algorithm provides similar performance to first order gradient descent method, but it significantly improves the training speed [16]. Genetic algorithm [17] is very robust for training RBF networks. Since it performs global search, genetic algorithm does not suffer from local minima problem, but it is very time and computation expensive, especially when the search space is huge.

In order to design compact RBF networks, many strategies and algorithms are proposed to grow/prune the number of RBF units according with instantaneous training information. Orr *et al.* [18] introduced a regularized forward selection method to select proper centers of RBF units. Chen *et al.* [19] proposed the orthogonal least squares learning algorithm which optimally increased the number of RBF units one by one until reaching required conditions. Huang *et al.* [20] presented a generalized growing/pruning strategy for designing compact RBF networks. Wu *et al.* [21] developed an extended self-organizing map to optimize the number of RBF units.

In this paper, the second order gradient method is proposed to train RBF networks. This method is derived from neuron-by-neuron (NBN) algorithm [22] and improved Levenberg–Marquardt algorithm [23] used for traditional neural network training. Enhancing the training algorithm itself leads to design compact and well-behaved RBF networks. In the proposed approach, all the parameters (as shown in Fig. 1), such as input weights, output weights, centers, and widths, are adjusted by second order update rule. Furthermore, the proposed algorithm does not suffer from huge Jacobian matrix storage and its side effects, when training data is huge. With the proposed training algorithm, RBF networks can be designed very compactly, at the same time, the network performances, such as training speed and approximation accuracy, are improved.

This paper is organized as follows. In Section II, the basic concepts of RBF networks are introduced, as well as

Manuscript received January 9, 2012; revised January 9, 2012; accepted January 13, 2012. Date of publication February 10, 2012; date of current version March 6, 2012.

T. Xie, H. Yu, J. Hewlett, and B. M. Wilamowski are with the Department of Electrical and Computer Engineering, Auburn University, Auburn, AL 36849-5201 USA (e-mail: tzx0004@tigermail.auburn.edu; hzy0004@tigermail.auburn.edu; jhewlett@uidaho.edu; wilam@ieee.org).

P. Różycki is with the University of Information Technology and Management, Rzeszów 35-959, Poland (e-mail: prozycki@wsiz.rzeszow.pl).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2012.2185059

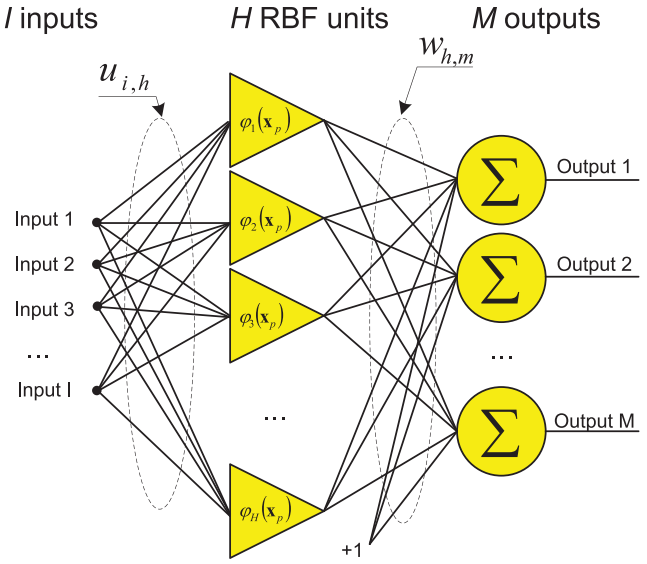


Fig. 1. RBF network with I inputs, H hidden units, and M outputs.

Levenberg–Marquardt algorithm. Section III presents the proposed algorithm for training RBF networks. Section IV describes a software implementation of the proposed algorithm. Section V gives several practical examples, as comparison between the proposed algorithm and other algorithms for training RBF networks.

II. Computational Fundamentals

Before presenting the computation fundamentals of RBF networks and Levenberg–Marquardt algorithm, let us introduce the commonly used indices in this paper.

1) i is the index of inputs, from 1 to I, where I is the number of input dimensions.
2) p is the index of patterns, from 1 to P, where P is the number of input patterns.
3) h is the index of hidden units, from 1 to H, where H is the number of units in the hidden layer.
4) m is the index of outputs, from 1 to M, where M is the number of outputs.
5) k is the index of iterations.

Other indices will be interpreted in related places.

A. RBF Networks

For a given problem with I-dimension inputs $\boldsymbol{x}= [x_1, x_2, x_3 \ldots x_i \ldots x_I]$, P patterns and M outputs, H RBF units in the hidden layer are assumed. Fig. 1 shows the standard three-layer architecture of RBF networks: input layer, hidden layer, and output layer.

When applying pattern p, each input $x_{p,i}$ is re-scaled by the input weights $u_{i,h}$, which represents the weight connection between the ith input and RBF unit h

$$y_{p,h,i} = x_{p,i} u_{i,h}. \tag{1}$$

In the traditional approaches, the elements of weight matrix $\boldsymbol{u}$ on the input layer are all set as "1." But in the proposed training algorithm, they will be treated as variables.

The scaled vector $\boldsymbol{y}_{p,h}= [\ y_{p,h,1}, y_{p,h,2} \ldots y_{p,h,i} \ldots y_{p,h,I}]$ is mapped into H-dimension by function

$$\varphi_h\left(\mathbf{x}_p\right) = \exp\left(-\frac{\left\|\mathbf{y}_{p,h} - \mathbf{c}_h\right\|^2}{\sigma_h}\right) \tag{2}$$

where $\|\cdot\|$ represents the computation of Euclidean norm, $\varphi_h(\cdot)$ is the activation function of RBF unit h, $\boldsymbol{c}_h$ and σ_h are the center and width of RBF unit h, respectively. Notice that the activation function in (2) is not the standard Gaussian function.

The outputs of RBF networks can be calculated as

$$o_{p,m} = \sum_{h=1}^{H} w_{h,m} \varphi_h\left(\mathbf{x}_p\right) + w_{0,m} \tag{3}$$

where $w_{h,m}$ is the output weight on the connection between hidden unit h and output m, $w_{0,m}$ is the bias weight of output m.

B. Levenberg–Marquardt Algorithm

The update rule of Levenberg–Marquardt algorithm is [24]

$$\Delta_{k+1} = \Delta_k + \left(\mathbf{J}_k^T \mathbf{J}_k + \mu_k \mathbf{I}\right)^{-1} \mathbf{J}_k^T \mathbf{e}_k \tag{4}$$

where μ is combination coefficient, Δ is the variable vector, $\boldsymbol{e}$ is error vector, and $\boldsymbol{J}$ is Jacobian matrix ($PM \times N$) defined as

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial \Delta_1} & \frac{\partial e_{1,1}}{\partial \Delta_2} & \cdots & \frac{\partial e_{1,1}}{\partial \Delta_N} \\ \frac{\partial e_{1,2}}{\partial \Delta_1} & \frac{\partial e_{1,2}}{\partial \Delta_2} & \cdots & \frac{\partial e_{1,2}}{\partial \Delta_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{1,M}}{\partial \Delta_1} & \frac{\partial e_{1,M}}{\partial \Delta_2} & \cdots & \frac{\partial e_{1,M}}{\partial \Delta_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{P,1}}{\partial \Delta_1} & \frac{\partial e_{P,1}}{\partial \Delta_2} & \cdots & \frac{\partial e_{P,1}}{\partial \Delta_N} \\ \frac{\partial e_{P,2}}{\partial \Delta_1} & \frac{\partial e_{P,2}}{\partial \Delta_2} & \cdots & \frac{\partial e_{P,2}}{\partial \Delta_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{P,M}}{\partial \Delta_1} & \frac{\partial e_{P,M}}{\partial \Delta_2} & \cdots & \frac{\partial e_{P,M}}{\partial \Delta_N} \end{bmatrix} \quad \boldsymbol{e} = \begin{bmatrix} e_{1,1} \\ e_{1,2} \\ \cdots \\ e_{1,M} \\ \cdots \\ e_{P,1} \\ e_{P,2} \\ \cdots \\ e_{P,M} \end{bmatrix} \tag{5}$$

where N is the number of variables, $e_{p,m}$ is the error calculated by

$$e_{p,m} = y_{p,m} - o_{p,m} \tag{6}$$

where $y_{p,m}$ is the desired output and $o_{p,m}$ is the actual output, at network output m when training pattern p

In implementation of Levenberg–Marquardt algorithm as described in [25], the whole Jacobian matrix in (5) is calculated and stored for further matrix multiplication using (4). As a consequence, the Levenberg–Marquardt algorithm is not capable of training problems with large number of patterns [26]. This limitation of Levenberg–Marquardt algorithm can be practically eliminated by using a different matrix multiplication routine, which allows replacing storage and computation of Jacobian matrix with Jacobian vector.

III. Improved Second Order (ISO) Algorithm

Following the computation procedure in NBN algorithm [23], the update rule (4) can be replaced with

$$\Delta_{k+1} = \Delta_k + \left(\mathbf{Q}_k + \mu_k \mathbf{I}\right)^{-1} \mathbf{g}_k \tag{7}$$

In		out
-1	-1	+1
-1	+1	-1
+1	-1	-1
+1	+1	+1

Fig. 2. Data set of XOR problem.

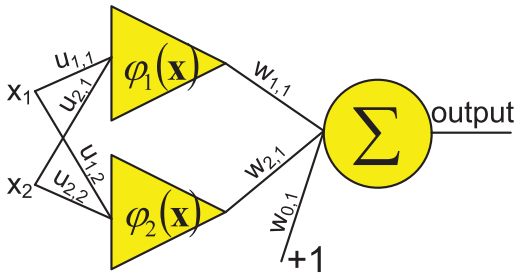


Fig. 3. RBF network for solving XOR problem.

where quasi Hessian matrix $\boldsymbol{Q}$ is directly calculated as the sum of $P\times M$ sub matrices $\boldsymbol{q}_{p,m}$

$$\mathbf{Q}=\sum_{p=1}^{P}\sum_{m=1}^{M}\mathbf{q}_{p,m}\quad \mathbf{q}_{p,m}=\mathbf{j}_{p,m}^{T}\mathbf{j}_{p,m} \tag{8}$$

and gradient vector $\boldsymbol{g}$ is calculated as the sum of $P\times M$ sub vectors $\eta_{p,m}$

$$\mathbf{g}=\sum_{p=1}^{P}\sum_{m=1}^{M}\boldsymbol{\eta}_{p,m}\quad \boldsymbol{\eta}_{p,m}=\mathbf{j}_{p,m}^{T}e_{p,m} \tag{9}$$

where vector $\boldsymbol{j}_{p,m}$ is one row of Jacobian matrix for pattern p associated with output m calculated by

$$\mathbf{j}_{p,m}=\left[\frac{\partial e_{p,m}}{\partial\Delta_1},\frac{\partial e_{p,m}}{\partial\Delta_2}\cdots\frac{\partial e_{p,m}}{\partial\Delta_n}\cdots\frac{\partial e_{p,m}}{\partial\Delta_N}\right] \tag{10}$$

and the error $e_{p,m}$ is given by (6).

In the proposed algorithm, there are four types of variables: output weight matrix $\boldsymbol{w}$, width vector $\boldsymbol{\sigma}$, input weight matrix $\boldsymbol{u}$, and center matrix $\boldsymbol{c}$. Therefore, the Jacobian row in (10) consists of four parts

$$\mathbf{j}_{p,m}=\left[\frac{\partial e_{p,m}}{\partial w_{h,m}}\cdots\frac{\partial e_{p,m}}{\partial\sigma_h}\cdots\frac{\partial e_{p,m}}{\partial u_{i,h}}\cdots\frac{\partial e_{p,m}}{\partial c_{h,i}}\cdots\right]. \tag{11}$$

A. Computation of $\partial e_{p,m}/\partial w_{h,m}$

The output weight matrix $\boldsymbol{w}$ presents the weight values on the connections between hidden layer and output layer, also including the bias weights on output units. Therefore, the output weight matrix $\boldsymbol{w}$ has $(H+1)\times M$ elements.

Using (6), the Jacobian element $\partial e_{p,m}/\partial w_{h,m}$ is calculated as

$$\frac{\partial e_{p,m}}{\partial w_{h,m}}=-\frac{\partial o_{p,m}}{\partial w_{h,m}}. \tag{12}$$

By combining with (3) and (12) is rewritten as

$$\frac{\partial e_{p,m}}{\partial w_{h,m}}=-\varphi_h\left(\mathbf{x}_p\right). \tag{13}$$

For bias weight $w_{0,m}$, related Jacobian element is calculated by

$$\frac{\partial e_{p,m}}{\partial w_{0,m}}=-1. \tag{14}$$

B. Computation of $\partial e_{p,m}/\partial\sigma_h$

The width vector $\boldsymbol{\sigma}$ consists of the width of each RBF unit, so the total number of elements is H.

For the RBF unit h, using (6) and the differential chain rule, the Jacobian element $\partial e_{p,m}/\partial\sigma_h$ is calculated as

$$\frac{\partial e_{p,m}}{\partial\sigma_h}=-\frac{\partial o_{p,m}}{\partial\sigma_h}=-\frac{\partial o_{p,m}}{\partial\varphi_h\left(\mathbf{x}_p\right)}\frac{\partial\varphi_h\left(\mathbf{x}_p\right)}{\partial\sigma_h}. \tag{15}$$

By combining with (2) and (3), (15) is rewritten as

$$\frac{\partial e_{p,m}}{\partial\sigma_h}=-\frac{w_{h,m}\varphi_h\left(\mathbf{x}_p\right)\left\|\mathbf{y}_{p,h}-\mathbf{c}_h\right\|^2}{\sigma_h^2} \tag{16}$$

where the vector $\boldsymbol{y}_{p,h}$ is defined in (1).

C. Computation of $\partial e_{p,m}/\partial u_{i,h}$

The input weight matrix $\boldsymbol{u}$ describes the weights on the connections between input layer and the hidden layer, so the total number of elements is $I\times H$.

Using (6) and the differential chain rule, the Jacobian element $\partial e_{p,m}/\partial u_{i,h}$ is calculated as

$$\frac{\partial e_{p,m}}{\partial u_{i,h}}=-\frac{\partial o_{p,m}}{\partial u_{i,h}}=-\frac{\partial o_{p,m}}{\partial\varphi_h\left(\mathbf{x}_p\right)}\frac{\partial\varphi_h\left(\mathbf{x}_p\right)}{\partial u_{i,h}}. \tag{17}$$

By combing with (1)–(3), (17) is rewritten as

$$\frac{\partial e_{p,m}}{\partial u_{i,h}}=\frac{2w_{h,m}\varphi_h\left(\mathbf{x}_p\right)x_{p,i}\left(x_{p,i}u_{i,h}-c_{h,i}\right)}{\sigma_h}. \tag{18}$$

D. Computation of $\partial e_{p,m}/\partial c_{h,i}$

The center matrix $\boldsymbol{c}$ consists of the centers of RBF units, and the number of elements is $H\times I$.

For RBF unit h, using (6) and the differential chain rule, the Jacobian element $\partial e_{p,m}/\partial c_{h,i}$ is calculated by

$$\frac{\partial e_{p,m}}{\partial c_{h,i}}=-\frac{\partial o_{p,m}}{\partial c_{h,i}}=-\frac{\partial o_{p,m}}{\partial\varphi_h\left(\mathbf{x}_p\right)}\frac{\partial\varphi_h\left(\mathbf{x}_p\right)}{\partial c_{h,i}}. \tag{19}$$

By combining with (1)–(3), (19) is rewritten as

$$\frac{\partial e_{p,m}}{\partial c_{h,i}}=-\frac{2w_{h,m}\varphi_h\left(\mathbf{x}_p\right)\left(x_{p,i}u_{i,h}-c_{h,i}\right)}{\sigma_h}. \tag{20}$$

With (13), (14), (16), (18), and (20), all the Jacobian row elements for output m when applying pattern p can be obtained. Then related sub quasi Hessian matrix $\boldsymbol{q}_{p,m}$ and sub gradient vector $\boldsymbol{\eta}_{p,m}$ can be computed by (8) and (9),

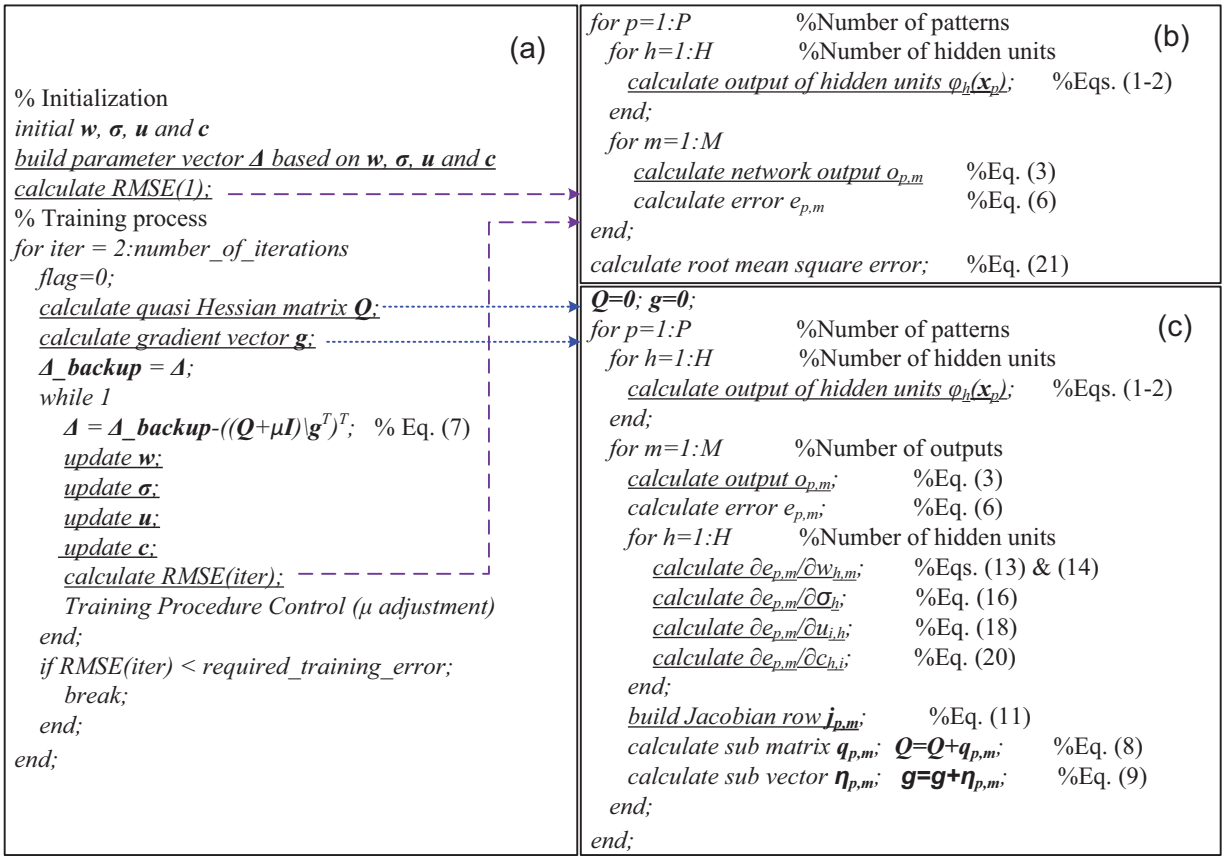


Fig. 4. Pseudo code (following MATLAB syntax) of the proposed algorithm to train RBF networks. Block (a), is the procedure for weight updating, block (b), evaluates the root mean square error (RMSE), and block (c) is used for quasi Hessian matrix and gradient vector computation. The underlined steps are additions to the computation in [23].

respectively, so do the quasi Hessian matrix and gradient vector. Notice that, all patterns are independent, so the related memory for $\boldsymbol{j}_{p,m}$, $\boldsymbol{q}_{p,m}$, and $\boldsymbol{\eta}_{p,m}$ can be reused.

IV. Implementation of the ISO Algorithm

In order to explain the training process of RBF networks using the proposed algorithm, let us use the parity-2 (XOR) classification problem as an illustration vehicle.

Fig. 2 shows the data set of XOR problem. The goal is to classify the four points $(-1,-1)$, $(-1,1)$, $(1,-1)$, and $(1,1)$ into two groups, which are marked by $+1$ and -1. Fig. 3 shows the minimum RBF network for solving the parity-2 problem.

Implementing the proposed algorithm on the example, the training procedure can be organized in the following steps.

A. Initialization

As shown in Fig. 3, the initial conditions are set as: output weights $\boldsymbol{w} = [\ w_{0,1},\ w_{1,1},\ w_{2,1}\ ]$, widths of two RBF units $\sigma = [\ \sigma_1,\ \sigma_2]$, input weights $\boldsymbol{u} = [\ u_{1,1},\ u_{2,1};\ u_{1,2},\ u_{2,2}\ ]$, and centers of two RBF units $\boldsymbol{c} = [\ c_{1,1}, c_{1,2}; c_{2,1}, c_{2,2}\ ]$.

In order to apply the update rule in (7), the variable vector Δ_1 is built by reforming the current parameters $\boldsymbol{w}$, $\boldsymbol{\sigma}$, $\boldsymbol{u}$, and $\boldsymbol{c}$ as: $\Delta_1 = [w_{0,1},\ w_{1,1},\ w_{2,1},\ \sigma_1,\ \sigma_2, u_{1,1},\ u_{2,1},\ u_{1,2},\ u_{2,2}, c_{1,1}, c_{1,2}, c_{2,1}, c_{2,2}\ ]$.

B. Error Evaluation

The root mean square error E is defined to evaluate the training procedure

$$E = \sqrt{\frac{\sum_{p=1}^{P}\sum_{m=1}^{M} e_{p,m}^2}{P \times M}} \tag{21}$$

where P is the number of patterns and M is the number of outputs.

Applying the first pattern $[-1,-1,1]$ to RBF unit 1, the vector multiplication of $\boldsymbol{x}_1 = [-1,-1]$ and $\boldsymbol{u}_1 = [u_{1,1}, u_{2,1}]$ is obtained using (1)

$$\mathbf{y}_{1,1} = [-u_{1,1}, -u_{2,1}]. \tag{22}$$

With (22), the Euclidean Norm of vector $\boldsymbol{y}_{1,1}$ and vector $\boldsymbol{c}_1$ is calculated as

$$\left\|\mathbf{y}_{1,1} - \mathbf{c}_1\right\|^2 = \left(-u_{1,1} - c_{1,1}\right)^2 + \left(-u_{2,1} - c_{1,2}\right)^2. \tag{23}$$

Using (2) and (23), the output of the RBF unit 1 is calculated by

$$\varphi_1(\mathbf{x}_1) = \exp\left(-\frac{\left\|\mathbf{y}_{1,1} - \mathbf{c}_1\right\|^2}{\sigma_1}\right). \tag{24}$$

Then by applying pattern $[-1,-1,1]$ to RBF unit 2, using similar computation with RBF unit 1, $\boldsymbol{y}_{1,2}$, and $\varphi_2(\boldsymbol{x}_1)$ are calculated.

TABLE I
PERFORMANCE COMPARISON FOR BOSTON HOUSING PROBLEM

Algorithms	CPU Time (s)		Training RMSE		Testing RMSE		Number of RBF units (mean)
	Mean	Dev	Mean	Dev	Mean	Dev	
GGAP	1.2399	0.2812	0.1507	0.0128	0.1418	0.0466	3.5
MRAN	12.731	2.2585	0.1440	0.0108	0.1356	0.0411	13.58
RANEKF	22.572	6.4159	0.1328	0.0086	0.1437	0.0464	19.98
RAN	4.2664	0.4846	0.3449	0.0620	0.3432	0.0770	18.8
ISO	0.6192	0.0591	0.1327	0.0471	0.1403	0.0553	1
ISO	1.1103	0.1596	0.0996	0.0383	0.1018	0.0430	2
ISO	1.5349	0.1688	0.0904	0.0318	0.0926	0.0369	3

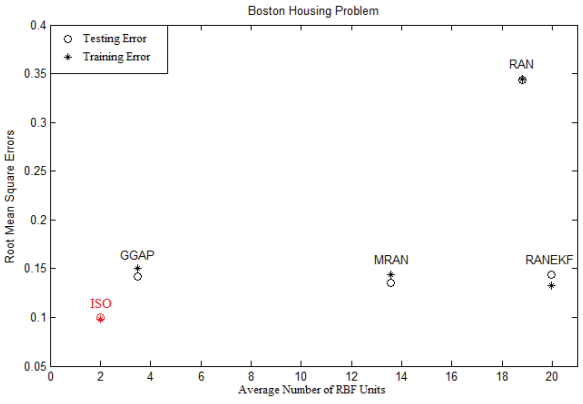


Fig. 5. RMS errors versus average number of RBF units for Boston housing problem.

Following the computation in (3), the network output is calculated as:

$$o_{1,1} = w_{0,1} + w_{1,1}\varphi_1(\mathbf{x}_1) + w_{2,1}\varphi_2(\mathbf{x}_1). \tag{25}$$

Using (6) and (25), the error $e_{1,1}$ for the first pattern is computed by

$$e_{1,1} = 1 - o_{1,1}. \tag{26}$$

Repeating the computation from (22) to (26) for other three patterns, the errors $e_{2,1}$, $e_{3,1}$, and $e_{4,1}$ are all obtained, then the root mean square error defined in (21) is calculated as

$$E_1 = \frac{1}{2}\sqrt{e_{1,1}^2 + e_{2,1}^2 + e_{3,1}^2 + e_{4,1}^2}. \tag{27}$$

C. Computation of Quasi Hessian Matrix and Gradient Vector

First of all, the quasi Hessian matrix $\boldsymbol{Q}_1$ and gradient vector $\boldsymbol{g}_1$ are initialized as zero

$$\mathbf{Q}_1 = \mathbf{0}, \quad \mathbf{g}_1 = \mathbf{0}. \tag{28}$$

Applying the first pattern $[-1, -1, 1]$ and going through the computation from (22) to (26), parameters $\varphi_1(\boldsymbol{x}_1)$, $\varphi_2(\boldsymbol{x}_1)$, $o_{1,1}$, *and* $e_{1,1}$ are all obtained.

Using (13), (14), (16), (18), and (20), all the elements of Jacobian row for the first pattern can be calculated and built in the format of (11): $\boldsymbol{j}_{1,1} = [\ \partial e_{1,1}/\partial w_{0,1},\ \partial e_{1,1}/\partial w_{1,1},\ \partial e_{1,1}/\partial w_{2,1},\ \partial e_{1,1}/\partial \sigma_1,\ \partial e_{1,1}/\partial \sigma_2,\ \partial e_{1,1}/\partial u_{1,1},\ \partial e_{1,1}/\partial u_{2,1},\ \partial e_{1,1}/\partial u_{1,2},\ \partial e_{1,1}/\partial u_{2,2},\ \partial e_{1,1}/\partial c_{1,1},\ \partial e_{1,1}/\partial c_{1,2},\ \partial e_{1,1}/\partial c_{2,1},\ \partial e_{1,1}/\partial c_{2,2}]$.

Using (8), the sub quasi Hessian matrix $\boldsymbol{q}_{1,1}$ is calculated to update the quasi Hessian matrix $\boldsymbol{Q}_1$

$$\mathbf{q}_{1,1} = \mathbf{j}_{1,1}^T \mathbf{j}_{1,1} \quad \mathbf{Q}_1 = \mathbf{Q}_1 + \mathbf{q}_{1,1}. \tag{29}$$

Using (9), the sub gradient vector $\boldsymbol{\eta}_{1,1}$ calculated to update the gradient vector $\boldsymbol{g}_1$

$$\boldsymbol{\eta}_{1,1} = \mathbf{j}_{1,1}^T e_{1,1} \quad \mathbf{g}_1 = \mathbf{g}_1 + \boldsymbol{\eta}_{1,1}. \tag{30}$$

By repeating the computation (29) and (30) for the other three patterns, the accumulated results of matrix $\boldsymbol{Q}_1$ and vector $\boldsymbol{g}_1$ are the required quasi Hessian matrix and gradient vector.

D. Parameters Update

After the computation of quasi Hessian matrix $\boldsymbol{Q}_1$ and gradient vector $\boldsymbol{g}_1$, using (7), the updated parameter vector Δ_2 can be calculated as

$$\boldsymbol{\Delta}_2^T = \boldsymbol{\Delta}_1^T + (\mathbf{Q}_1 + \mu_1 \mathbf{I})^{-1} \mathbf{g}_1. \tag{31}$$

From the new parameter vector $\boldsymbol{\Delta}_2$, the parameters $\boldsymbol{w}$, $\boldsymbol{\sigma}$, $\boldsymbol{u}$, and $\boldsymbol{c}$ can be extracted according with the order of constructing the parameter vector $\boldsymbol{\Delta}_1$ as was done previously.

E. Training Procedure Control

With the updated parameters $\boldsymbol{w}$, $\boldsymbol{\sigma}$, $\boldsymbol{u}$, and $\boldsymbol{c}$, new root mean square error E_2 can be evaluated by following the procedure described in Section IV-B. Then the training process is controlled by the rules.

1) If E_2 is less than the setting value, training converges.
2) If $E_1 \geq E_2$, reduce parameter u_1 and keep the current parameter values (used for E_2 calculation). Then go through Sections IV-B–IV-E for next iteration.
3) If $E_1 < E_2$, increase parameter u_1 and recover previous parameter values (used for E_1 calculation). Then go through Sections IV-C–IV-E. A counter (variable *flag* in Fig. 4) should be added here to help avoid dead loop.

Fig. 4 shows the pseudo code of the proposed algorithm with links to the equations given in previous sections. The block (1) in Fig. 4 is the main procedure for weight updating, the block (2) evaluates the training process according to root mean

TABLE II
PERFORMANCE COMPARISON FOR ABALONE AGE PREDICTION

Algorithms	CPU time (s)		Training RMSE		Testing RMSE		Number of RBF units (mean)
	Mean	Dev	Mean	Dev	Mean	Dev	
GGAP-GMM	/	/	0.08	/	0.0850	0.0027	5.13
GGAP	83.784	73.401	0.0963	0.0061	0.0966	0.0068	23.62
MRAN	1500.4	134.08	0.0836	0.0039	0.0837	0.0042	87.571
RANEKF	90806	18193	0.0738	0.0042	0.0794	0.0053	409
RAN	105.17	6.1714	0.0931	0.0091	0.0978	0.0092	345.58
ISO	5.9672	0.7495	0.0792	0.0109	0.0792	0.0101	4
ISO	8.4672	0.9885	0.0778	0.0066	0.0762	0.0085	5
ISO	11.625	1.7499	0.0748	0.0047	0.0738	0.0035	6

/ data not available in the literature [30].

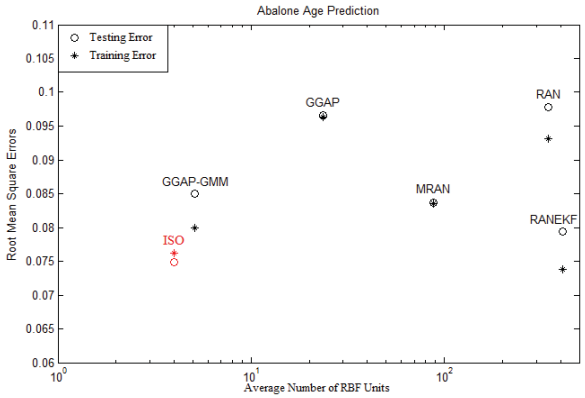


Fig. 6. RMS errors versus average number of RBF units for abalone age prediction problem.

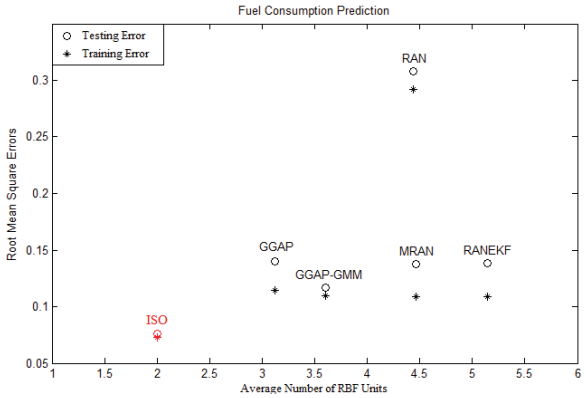


Fig. 7. RMS errors versus average number of RBF units for fuel consumption prediction problem.

square error and the block (3) performs quasi Hessian matrix and gradient vector computation. Normally, the μ parameter in the proposed algorithm is initialed as 0.01.

One may notice that the proposed ISO algorithm for RBF network training takes the advantages of the NBN algorithm introduced in literatures [22], [23] for neural network training. However, because of the different activation functions, network architectures and parameters of the two network models, the Jacobian row computation and parameter update are very different, as the underlined steps shown in Fig. 4.

V. EXPERIMENTAL RESULTS

Several practical issues are presented to test the performance of the ISO algorithm, from the point of training speed, required hidden units, training error, and generalization error. In Section V-A, the ISO algorithm is compared with several algorithms, including GGAP [20], MRAN [27], RANEKF [28], RAN [29], and GGAP-GMM [30]. In Section V-B, four cases with different training parameters are compared based on two practical problems. In Section V-C, the proposed ISO algorithm is compared with first order gradient algorithm and Gauss–Newton method, based on MATLAB PEAK problem.

The testing environment of the proposed algorithm consists of: Windows 7 Professional 32-bit operating system, AMD Athlon (tm) $\times$ 2 Dual-Core QL-65 2.10 GHz processor, 3.00GB (2.75GB usable) RAM, MATLAB 2007b platform. All the attributes in the data set are normalized (divided by the maximum value of the attribute) in range [0, 1], and 100 trials are repeated under similar conditions for each study case, when applying the proposed algorithm for training/testing.

A. Comparing With Other Algorithms

In the performed experiments, three practical problems, including Boston housing problem, abalone age prediction, and fuel consumption prediction from [31], are applied to test the performance of the proposed ISO algorithm, by comparing with other five algorithms. Each testing case for ISO algorithm is repeated for 100 trials and testing results of other algorithms are from [20] and [30].

The Boston Housing problem has total of 506 observations, each of which consists of 13 input attributes (12 continuous attributes and one binary-valued attribute) and one continuous output attribute (the median value of owner-occupied homes). For each trial, 481 randomly selected observations are going to be applied for training, and the rest 25 observations will be used to test the trained RBF network. The experiment results are shown in Table I. Fig. 5 shows the training/testing RMS error trajectories when increasing the number of RBF units.

TABLE III
PERFORMANCE COMPARISON FOR FUEL CONSUMPTION PREDICTION OF AUTOS

Algorithms	CPU Time (s)		Training RMSE		Testing RMSE		Number of RBF units (mean)
	Mean	Dev	Mean	Dev	Mean	Dev	
GGAP-GMM	/	/	0.11	/	0.1167	0.0134	3.6
GGAP	0.4520	0.0786	0.1144	0.0132	0.1404	0.0270	3.12
MRAN	1.4644	0.2453	0.1086	0.0100	0.1376	0.0226	4.46
RANEKF	1.0103	0.1694	0.1088	0.0117	0.1387	0.0289	5.14
RAN	0.8042	0.1417	0.2923	0.0808	0.3080	0.0915	4.44
ISO	0.2105	0.0107	0.0975	0.0463	0.0995	0.0433	1
ISO	0.3043	0.0248	0.0784	0.0294	0.0817	0.0289	2
ISO	0.5922	0.0683	0.0622	0.0077	0.0645	0.0094	3

/ data not available in the literature [30].

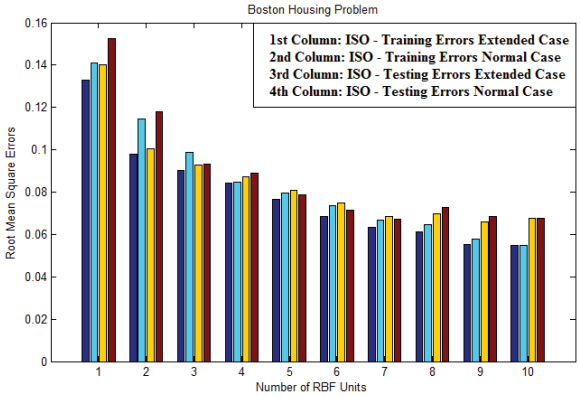


Fig. 8. Boston housing problem: training/testing errors versus number of RBF units plotting of ISO algorithm for different parameters: 1) normal case and 2) extended case.

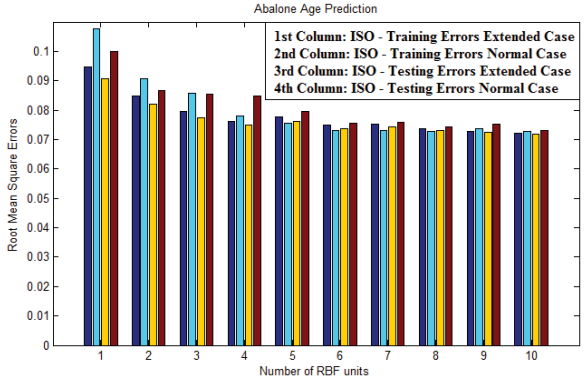


Fig. 9. Abalone age prediction: training/testing errors versus number of RBF units plotting of ISO algorithm for different parameters: 1) normal case and 2) extended case.

As shown in Fig. 5, for the Boston housing problem, the ISO algorithm can obtain smaller training/testing errors than other four algorithms with only 2 RBF units.

The Abalone problem consists of 4177 observations, each of which consists of seven continuous input attributes and one continuous output attribute (age in years). For each trial, 3000 randomly selected observations are applied as training data and the remaining 1177 observations are applied for testing. The experimental results are presented in Table II. Fig. 6 shows the relationship between training/testing RMS errors and the average number of RBF units.

As shown in Fig. 6, the proposed ISO algorithm can reach similar or smaller training/testing errors with other algorithms using significantly compact RBF network consisting of 4 RBF units.

The Auto MPG problem has 398 patterns. Each pattern consists of seven continuous input attributes and one continuous output attribute (the fuel consumption in mile-per-gallon). For each trial, 320 randomly selected patterns are applied for training and the remaining 78 patterns are applied for testing. The experimental results are shown in Table III. Fig. 7 presents the changing of training/testing RMS errors as the average number of RBF unit increases.

As the comparison results shown in Fig. 7, for the fuel consumption prediction problem, the proposed ISO method can reach smaller training/testing errors than other algorithms, with very compact RBF network consisting of 2 RBF units. One may also notice that the ISO method got better generalization ability on this problem, since the differences between training errors and testing errors are much smaller than other algorithms.

The comparison results presented in Tables I–III show that the proposed algorithm can reach similar or smaller training/testing RMS errors with much less number of RBF units and less training time than other algorithms.

Furthermore, in order to clarify the contribution of the second order update rule and the strategy of using extended parameters to the presented performance, the ISO algorithm is tested in two cases: 1) normal case: training parameters consist of output weights, centers, and widths; and 2) extended case: training parameters consist of input weights, output weights, centers, and widths. Figs. 8–10 present the results of training/testing errors for different number of RBF units, in both normal and extended cases of ISO algorithm. For each case, from left to right, the first column (blue bar) is training errors of extended case, the second column (cyan bar) is training errors of normal case, the third column (yellow bar) is the testing errors of extended case, and the fourth column (red bar) is the testing errors of normal case.

As per the results shown in Figs. 8–10, one may also notice that the good performance of the proposed ISO algorithm

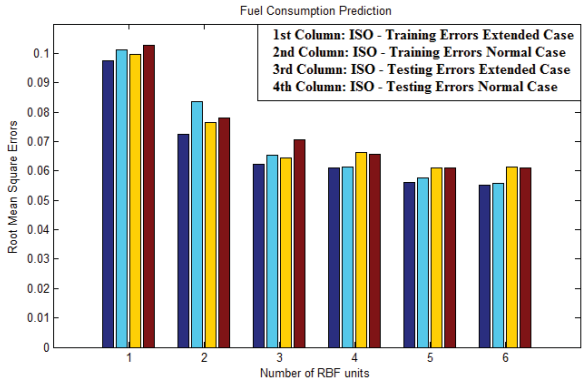


Fig. 10. Fuel consumption prediction: training/testing errors versus number of RBF units plotting of ISO algorithm for different parameters: 1) normal case and 2) extended case.

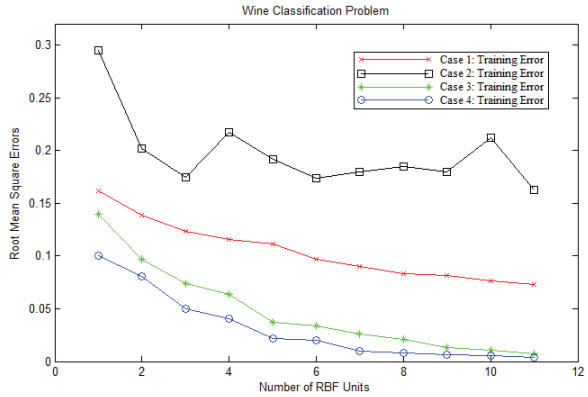


Fig. 11. Root mean square errors versus number of RBF units for wine classification problem.

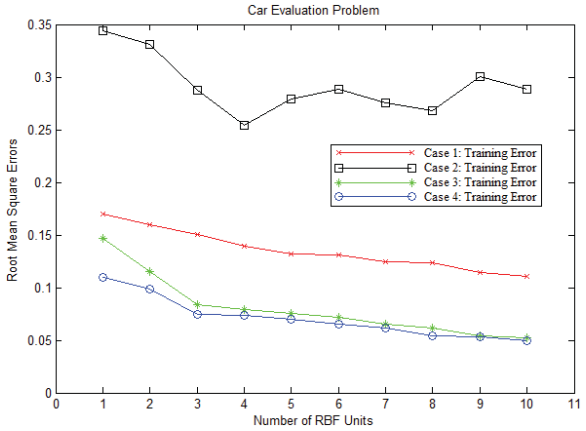


Fig. 12. Root mean square errors versus number of RBF units for car evaluation problem.

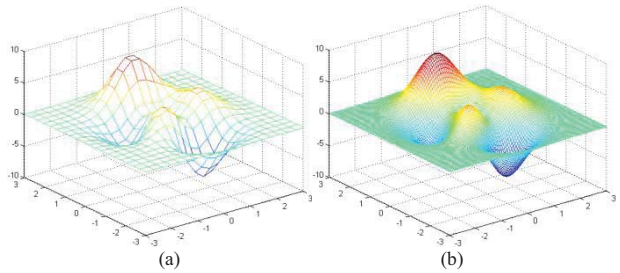


Fig. 13. Peaks function approximation problem: (a) training data, $20 \times 20 = 400$ points and (b) testing data, $100 \times 100 = 10\,000$ points.

can be mainly ascribed to the powerful search ability of second order algorithm. By applying the extra input weights as training parameters, the performance of RBF networks can be improved for small networks, as the increase of network size, the improvement becomes less obvious, which means that, at certain point of network sizes, both the extended models and the normal models have enough power for good approximation.

B. Performance With Different Training Parameters

In the experiments conducted, two of the most popular data sets in [31], wine classification and car evaluation, will be applied to test the proposed ISO algorithm.

In the wine problem, there are three types of wines required to be classified according with other 13 attributes. There are 178 observations in total.

In the car evaluation problem, six input attributes are used to evaluate the degree of satisfaction about the car. String data are replaced by natural numbers 1, 2, 3…. There are 1728 observations in total.

With the two benchmark problems, the following four cases with different variables are tested, using the proposed ISO algorithm.

1) Case 1: Only output weight matrix $\boldsymbol{w}$ is updated.
2) Case 2: Only input weight matrix $\boldsymbol{u}$ is updated.
3) Case 3: Output weight matrix $\boldsymbol{w}$, width vector $\boldsymbol{\sigma}$ and center matrix $\boldsymbol{c}$ are updated.
4) Case 4: Output weight matrix $\boldsymbol{w}$, width vector $\boldsymbol{\sigma}$, input weight matrix $\boldsymbol{u}$, and center matrix $\boldsymbol{c}$ are all updated.

All the input weights, output weights, and widths are randomly generated in range (0, 1). All centers are randomly selected from the training dataset. The maximum iteration is 100 and each testing case is repeated for 50 trials.

As shown in Figs. 11 and 12, several observations can be concluded.

1) As the number of RBF unit increases, except the Case 2, the training errors decrease stably.
2) Case 2 (squares in line) shows the worst performance, which is mainly depends on the initial conditions. So adjusting input weights only is not helping for RBF networks design.
3) For the same number of RBF units, Case 4 (circles in line) mostly gets smaller training errors than other three cases.
4) As the number of RBF unit increases, the difference of training results between Case 3 and Case 4 becomes small.

C. Training Speed Comparison

In the experiment conducted, the proposed ISO method is compared with the first order gradient method (with momentum [32]) and enhanced Gauss–Newton method, by solving the PEAK function approximation problem. Notice that, the

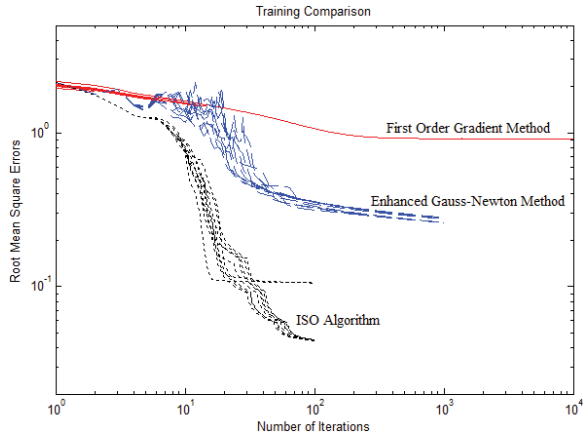


Fig. 14. Training RMS error trajectories: red solid-line is for first order gradient method, blue dash-line is for enhanced Gauss–Newton method, and black dot-line stands for ISO algorithm (ten trials for each algorithm).

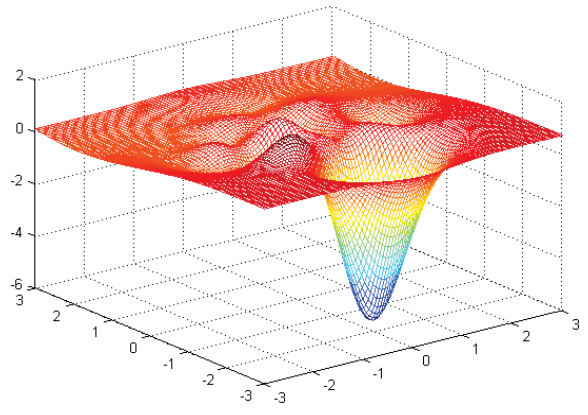


Fig. 15. Error surface of the approximating result using first order gradient method, with $E_{Train} = 0.8911$ and $E_{Test} = 0.9192$.

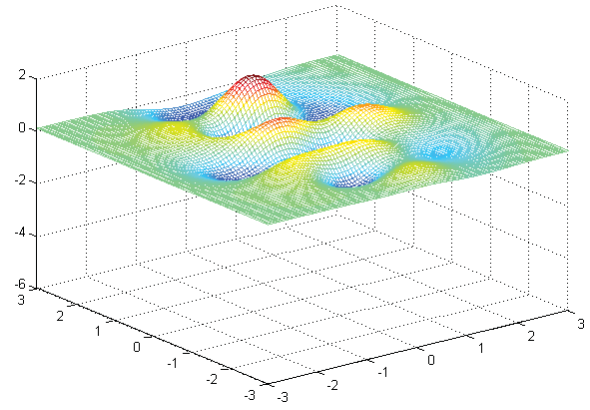


Fig. 16. Error surface of the approximating result using enhanced Gauss–Newton method, with $E_{Train} = 0.2769$ and $E_{Test} = 0.2869$.

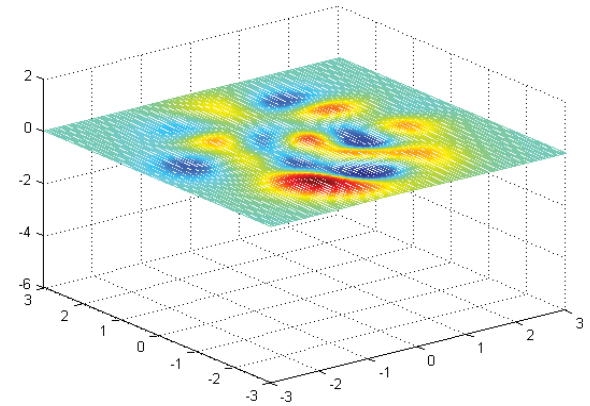


Fig. 17. Error surfaces of the approximating result using the proposed ISO method with $E_{Train} = 0.0424$ and $E_{Test} = 0.0440$.

original Gauss–Newton method seldom converges because Hessian matrix is mostly not invertible for complex error surfaces. In the experiment, the Gauss–Newton method is enhanced by adding a constant value to the diagonal elements of Hessian matrix when it is not invertible.

The purpose is to approximate the surface in Fig. 13(b) using the surface in Fig. 13(a). All the data come from MATLAB PEAKS function. RBF network with five hidden units is applied to do the approximation. The maximum number of iteration is 200 for ISO algorithm, 10 000 for first order gradient method and 1000 for the enhanced Gauss–Newton algorithm. All the parameters, including input weights, output weights, centers, and widths are adjusted by the three algorithms.

With the same initial centers, widths, and input weights, but randomly generated output weights, the training RMS error trajectories of the three algorithms for ten trials each are shown in Fig. 14. One may notice that the proposed ISO algorithm (black dot-line) costs significantly less iterations than first order gradient method (red solid-line) and converges to much smaller RMS errors than both first order gradient method and enhanced Gauss–Newton method (blue dash-line), in the limited iterations (when training errors get saturated).

Figs. 15–17 show the best error surfaces of the approximating results we have tried using first order gradient method (Fig. 15), enhanced Gauss–Newton method (Fig. 16), and the proposed ISO algorithm (Fig. 17), respectively.

One may notice that, with the smallest training error, the proposed ISO algorithm also gets the smaller approximation errors (better generalization results) than both first order gradient method and enhanced Gauss–Newton method. Obviously, because of its limited search ability, first order gradient method is not able to adjust the centers of RBF units properly.

VI. Conclusion

This paper applied the improved Levenberg–Marquardt algorithm for training RBF networks. Inheriting the high quality performance of Levenberg–Marquardt algorithm, the proposed ISO algorithm exhibits its powerful search ability and fast convergence in the presented examples. The power of second order algorithms contributes mostly to design more compact RBF networks than other algorithms. Compact RBF networks benefit the design in two aspects. First of all, compact architecture could be more efficient for hardware implementation. On the other hand, the less number of RBF units used for design, the better generalization ability, the trained networks can obtain [26]. Based on the experimental results presented in Figs. 8–12, it could be empirically con-

cluded that, variable space with higher dimensions (extended networks) is helpful to improve the approximating accuracy of RBF networks when the network size is small, in other words, the extended networks can be trained to smaller errors, so their size can be reduced by one or two RBF units in comparison to normal networks. However, this is not true when network size was increased. As the size of networks increases, the extended networks perform very similarly with the normal networks.

During the implementation of the ISO proposed algorithm, matrix operations were replaced by vector operations, which lead to significant memory reduction and speed benefit. The proposed ISO algorithm can be applied to design RBF networks for problems with basically unlimited number of training patterns.

The proposed ISO training algorithm already shows advantages, such as good search ability, fast convergence, and high network efficiency, even with randomly selected initial parameters. However, we believe that, by combining with proper clustering methods for initialization or pruning/growing strategies to adjust the number of hidden units during training process, the performance of the proposed algorithm can be further improved.

The proposed ISO algorithm was implemented in the training tool and can be downloaded freely from the following website: Available from http://www.eng.auburn.edu/∼wilambm/nnt/index.htm.

REFERENCES

[1] J. Moody and C. J. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Comput.*, vol. 1, no. 2, pp. 281–294, 1989.

[2] M. J. D. Powell, "Radial basis functions for multivariable interpolation: A review," in *Proc. IMA Conf. Algorithms Applicat. Funct. Data*, 1985, pp. 143–167.

[3] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Comput.*, vol. 3, no. 2, pp. 246–257, 1991.

[4] K. Meng, Z. Y. Dong, D. H. Wang, and K. P. Wong, "A self-adaptive RBF neural network classifier for transformer fault analysis," *IEEE Trans. Power Syst.*, vol. 25, no. 3, pp. 1350–1360, Feb. 2010.

[5] K. Meng, Z. Y. Dong, D. H. Wang, and K. P. Wong, "A self-adaptive RBF neural network classifier for transformer fault analysis," *IEEE Trans. Power Syst.*, vol. 25, no. 3, pp. 1350–1360, Aug. 2010.

[6] S. Huang and K. K. Tan, "Fault detection and diagnosis based on modeling and estimation methods," *IEEE Trans. Neural Netw.*, vol. 20, no. 5, pp. 872–881, Apr. 2009.

[7] Y. J. Lee and J. Yoon, "Nonlinear image upsampling method based on radial basis function interpolation," *IEEE Trans. Image Process.*, vol. 19, no. 10, pp. 2682–2692, Oct. 2010.

[8] S. Ferrari, F. Bellocchio, V. Piuri, and N. A. Borghese, "A hierarchical RBF online learning algorithm for real-time 3-D scanner," *IEEE Trans. Neural Netw.*, vol. 21, no. 2, pp. 275–285, Feb. 2010.

[9] L. Cai, A. B. Rad, and W. L. Chan, "An intelligent longitudinal controller for application in semiautonomous vehicles," *IEEE Trans. Indust. Electron.*, vol. 57, no. 4, pp. 1487–1497, Apr. 2010.

[10] C. C. Tsai, H. C. Huang, and S. C. Lin, "Adaptive neural network control of a self-balancing two-wheeled scooter," *IEEE Trans. Indust. Electron.*, vol. 57, no. 4, pp. 1420–1428, Apr. 2010.

[11] H. Yu, T. T. Xie, S. Paszczynski, and B. M. Wilamowski, "Advantages of radial basis function networks for dynamic system design," *IEEE Trans. Indust. Electron.*, vol. 58, no. 12, pp. 5438–5450, Dec. 2011.

[12] B. M. Wilamowski, N. Cotton, J. Hewlett, and O. Kaynak, "Neural network trainer with second order learning algorithms," in *Proc. 11th INES Int. Conf. Intell. Eng. Syst.*, Budapest, Hungary, Jun.–Jul. 2007, pp. 127–132.

[13] Z. Hong, "Algebraic feature extraction of image for recognition," *Pattern Recognit.*, vol. 24, no. 3, pp. 211–219, 1991.

[14] E. S. Chng, S. Chen, and B. Mulgrew, "Gradient radial basis function networks for nonlinear and nonstationary time series prediction," *IEEE Trans. Neural netw.*, vol. 7, no. 1, pp. 190–194, Jan. 1996.

[15] N. B. Karayiannis, "Reformulated radial basis neural networks trained by gradient descent," *IEEE Trans. Neural Netw.*, vol. 10, no. 3, pp. 657–671, Aug. 2002.

[16] D. Simon, "Training radial basis neural networks with the extended kalman filter," *Neurocomputing*, vol. 48, nos. 1–4, pp. 455–475, 2002.

[17] B. A. Whitehead and T. D. Choate, "Cooperative-competitive genetic evolution of radial basis function centers and widths for time series prediction," *IEEE Trans. Neural Netw.*, vol. 7, no. 4, pp. 869–880, Jul. 1996.

[18] M. J. L. Orr, "Regularization in the selection of radial basis function centers," *Neural Computat.*, vol. 7, no. 3, pp. 606–623, May 1995.

[19] S. Chen, C. F. N. Cowan, and P. M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Trans. Neural Netw.*, vol. 2, no. 2, pp. 302–309, Mar. 1991.

[20] G. B. Huang, P. Saratchandran, and N. Sundararajan, "An efficient sequential learning algorithm for growing and pruning RBF (GAP-RBF) networks," *IEEE Trans. Syst. Man Cybern.*, vol. 34, no. 6, pp. 2284–2292, Dec. 2004.

[21] S. Wu and T. W. S. Chow, "Induction machine fault detection using SOM-based RBF neural networks," *IEEE Trans. Indust. Electron.*, vol. 51, no. 1, pp. 183–194, Feb. 2004.

[22] A. Malinowski and H. Yu, "Comparison of embedded system design for industrial applications," *IEEE Trans. Indust. Informat.*, vol. 7, no. 2, pp. 244–254, May 2011.

[23] B. M. Wilamowski and H. Yu, "Improved computation for levenberg marquardt training," *IEEE Trans. Neural Netw.*, vol. 21, no. 6, pp. 930–937, Jun. 2010.

[24] B. M. Wilamowski and H. Yu, "Neural network learning without backpropagation," *IEEE Trans. Neural Netw.*, vol. 21, no. 11, pp. 1793–1803, Nov. 2010.

[25] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the marquardt algorithm," *IEEE Trans. Neural Netw.*, vol. 5, no. 6, pp. 989–993, Nov. 1994.

[26] B. M. Wilamowski, "Challenges in applications of computational intelligence in industrial electronics," in *Proc. Int. Symp. Indust. Electron.*, Bari, Italy, Jul. 2010, pp. 15–22.

[27] N. Sundararajan, P. Saratchandran, and Y. W. Li, *Radial Basis Function Neural Networks With Sequential Learning: MRAN and Its Applications*. Singapore: World Scientific, 1999.

[28] V. Kadirkamanathan and M. Niranjan, "A function estimation approach to sequential learning with neural networks," *Neural Comput.*, vol. 5, no. 6, pp. 954–975, 1993.

[29] J. Platt, "A resource-allocating network for function interpolation," *Neural Comput.*, vol. 3, no. 2, pp. 213–225, 1991.

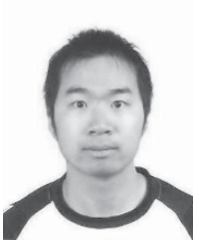
[30] M. Bortman and M. Aladjem, "A growing and pruning method for radial basis function networks," *IEEE Trans. Neural Netw.*, vol. 20, no. 6, pp. 1039–1045, Jun. 2009.

[31] C. Blake and C. Merz, "UCI repository of machine learning databases," Ph.D. dissertation, Dept. Inform. Comput. Sci., Univ. California, Irvine, 1998.

[32] H. Yu and B. M. Wilamowski, "Efficient and reliable training of neural networks," in *Proc. 2nd IEEE Human Syst. Interaction Conf.*, Catania, Italy, May 2009, pp. 109–115.

Tiantian Xie received the Ph.D. degree in microelectronics and solid-state electronics from the Huazhong University of Science and Technology, Hubei, China, in 2009. She is currently pursuing the Ph.D. degree in electrical engineering with Auburn University, Auburn, AL.

She is a Research Assistant with the Department of Electrical and Computer Engineering, Auburn University. Her current research interests include computational intelligence, piezoelectrical, and pyroelectrical materials.



Hao Yu (S'10) received the M.S. degree in electrical engineering from the Huazhong University of Science and Technology, Hubei, China, and the Ph.D. degree in electrical and computer engineering from Auburn University, Auburn, AL, in 2006 and 2011, respectively.

He was a Research Assistant with the Department of Electrical and Computer Engineering, Auburn University. He is currently a Senior Software Developer in lattice semiconductor Corporation, Hillsboro, OR. His current research interests include machine learning, EAD, and FPGA developments.

Dr. Yu is an IES Member and serves as a reviewer for the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, the IEEE TRANSACITONS ON INDUSTRIAL INFORMACTICS, and the IEEE TRANSACTIONS ON NEURAL NETWORKS.

Joel D. Hewlett received the M.S. and Ph.D. degrees in electrical engineering from Auburn University, Auburn, AL, in 2009 and 2011, respectively.

He is currently a Post-Doctoral Fellow with the Department of Computer Science, Center for Advanced Energy Studies, University of Idaho, Idaho Falls. He was a Systems Analyst with Delta Research, Inc., Huntsville, AL. His current research interests include intelligent systems, neural networks, numerical optimization, evolutionary computation, and control systems.

Dr. Hewlett is a reviewer for the IEEE TRANSANCTIONS ON INDUSTRIAL ELECTRONICS, the IEEE TRANSACTIONS ON INDUSTRIAL INFORMACTICS, and the IEEE TRANSACTIONS ON NEURAL NETWORKS.

Paweł Różycki received the M.S. and Ph.D. degrees in telecommunications from the AGH University of Science and Technology, Kraków, Poland, in 2001 and 2011, respectively.

He is an Assistant Professor with the University of Information Technology and Management, Rzeszów, Poland. He is the co-author of more than 20 journal and conference papers in the area of telecommunications network reliability. His current research interests include design and modeling of resilient networks as well as control plane architecture of the next-generation optical transport networks.

Bogdan M. Wilamowski (SM'83–F'00) received the M.S. degree in computer engineering, the Ph.D. degree in neural computing, and the Dr.Habil. degree in integrated circuit design in 1966, 1970, and 1977, respectively.

He was with the Gdansk University of Technology, Gdansk, Poland, University of Information Technology and Management, Rzeszow, Poland, Auburn University, Auburn, AL, University of Arizona, Tucson, University of Wyoming, Laramie, and the University of Idaho, Moscow. He is currently the Director of the Alabama Micro/Nano Science and Technology Center, Auburn University.

Dr. Wilamowski was the Vice President of the IEEE Computational Intelligence Society from 2000 to 2004 and the President of the IEEE Industrial Electronics Society from 2004 to 2005. He served as an Associate Editor in numerous journals. He was the Editor-in-Chief of the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS from 2007 to 2010, and currently serves as the Editor-in-Chief of the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS.