Advantages of Radial Basis Function Networks for Dynamic System Design

Hao Yu, *Student Member, IEEE*, Tiantian Xie, Stanisław Paszczyñski, *Senior Member, IEEE*, and Bogdan M. Wilamowski, *Fellow, IEEE*

***Abstract*—Radial basis function (RBF) networks have advantages of easy design, good generalization, strong tolerance to input noise, and online learning ability. The properties of RBF networks make it very suitable to design flexible control systems. This paper presents a review on different approaches of designing and training RBF networks. The recently developed algorithm is introduced for designing compact RBF networks and performing efficient training process. At last, several problems are applied to test the main properties of RBF networks, including their generalization ability, tolerance to input noise, and online learning ability. RBF networks are also compared with traditional neural networks and fuzzy inference systems.**

***Index Terms*—Adaptive control, fuzzy inference systems, neural networks, online learning, radial basis function (RBF) networks.**

I. Background

THE Proportional-Integral-Differential (PID) algorithm dominates the design of controllers in industrial applications [1]–[5]. It is maturely developed and can be easily implemented in both software and hardware. The drawback of a PID controller is that it only works well for linear systems which are seldom appear in the real world.

For nonlinear controller design, one method is to approximate the system linearly around equilibrium points. With this linear approximation, in limited input range, PID algorithm can still be applied for nonlinear controller design.

An alternative method is to compensate the system nonlinearity by introducing an opposite adaptive signal, so as to linearize the input–output relationship.

Fuzzy inference systems are often adopted for the nonlinear compensation. Li and Lee [6] presented the dynamic fuzzy controller combined with two synergic PID controllers to simultaneously control both fluid- and radiation-based cooling mechanisms, to dissipate exhaust heat of onboard electronic components inside spacecraft to the outer space environment. Suetake *et al.* [7] implemented the fuzzy controller on a digital signal processor, used to adjust the scalar speed of three-phase induction motor. Abiyev and Kaynak [8] presented a type 2 TSK fuzzy neural system for identification and control of time-varying plants. The advantage of fuzzy inference systems is that the fuzzy models can be very easily designed using the given data set without parameter adjustment. However, the tradeoff of the very simple design process is the accuracy of approximation. The control surfaces (input–output relationship) obtained by fuzzy inference systems are often very raw, which may lead to raw control and instabilities [9]. Therefore, for nonlinear compensation, fuzzy inference systems are often not directly applied in the control loop, instead, they are used to adjust the control parameters, such as proportional, integral, and differential factors in PID control. Another main disadvantage of fuzzy inference systems is that both the computation cost and response time are increased exponentially proportional to the size of inputs.

Another way of nonlinear compensation is to use neural networks as approximator. Bhattacharya and Chakraborty [10] designed an adaptive controller based on the "*adaline*" network to improve the dynamic performance of a shunt-type active power filter. Cotton *et al.* [11] implemented neuron-by-neuron algorithm which is used to train arbitrarily connected neural networks on an inexpensive microcontroller. The system was applied to compensate the nonlinearity in forward kinematics. Comparing with fuzzy inference systems, neural networks can achieve more accurate approximation and response much faster. However, because of the improper selection network architectures and training algorithms [12], engineers often feel frustrated on the generalization ability of neural networks.

Like neural networks and fuzzy inference systems, radial basis function (RBF) networks [13] were also proven to be universal approximator [14]. Because of the simple and fixed three-layer architecture (Fig. 1), RBF networks are much easier to be designed and trained than neural networks. From the point of generalization, RBF networks can respond well for patterns which are not used for training. RBF networks have strong tolerance to input noise, which enhances the stability of the designed systems. Therefore, it is reasonable to consider RBF network as a competitive method of nonlinear controller design. Lin and Lian [15] merged RBF networks with self-organizing fuzzy controller, to optimize the parameter selection. The hybrid controller was applied to manipulate an active suspension system. Tsai *et al.* [16] presented an adaptive controller using RBF networks to perform self-balancing and yaw control for a two-wheeled self-balancing scooter.

Manuscript received April 6, 2011; revised June 8, 2011 and August 1, 2011; accepted August 1, 2011. Date of publication August 15, 2011; date of current version September 20, 2011.

H. Yu, T. Xie, and B. M. Wilamowski are with the Department of Electrical and Computer Engineering, Auburn University, Auburn, AL 36849 USA (e-mail: hzy0004@auburn.edu; tzx0004@auburn.edu; wilambm@auburn.edu).

S. Paszczyñski is with the Department of Distributed Systems, University of Information Technology and Management, 35-959 Rzeszów, Poland (e-mail: spaszczynski@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/TIE.2011.2164773

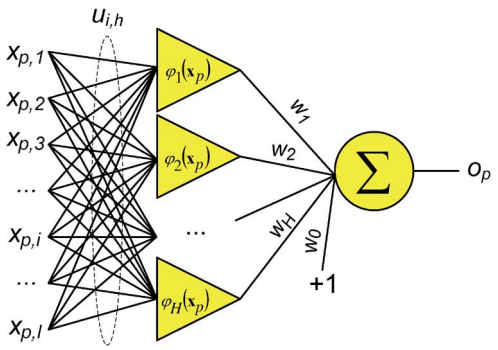


Fig. 1. RBF network with H RBF units and a single output unit.

Control systems become more complicated if the nonlinear behaviors change with the time, since unpredictable observation data may be added or removed from the previous data set. In this case, the traditional offline design of neural networks is not capable to satisfy the dynamical design. Instead, approximation models with online learning ability, which can handle dynamically changing data set, become attractive. Pucci and Cirrincione [17] developed a wind generator based on induction machines, and the growing neural gas network was applied in the control loop as a virtual anemometer to replace the speed sensors. Le and Jeon [18] presented a neural network based low-speed-damping controller implemented on FPGA, to remove nonlinear disturbance of the stepper motor at low speeds. Online backpropagation learning algorithm was applied to avoid identification process for network parameters. Xia *et al.* [19] introduced a fuzzy controller combined with an online learning neural network identifier, to perform dynamic decoupling control of permanent-magnet spherical motor. Cai *et al.* [20] proposed a hybrid controller using fuzzy logic and RBF networks, for intelligent cruise control of semiautonomous vehicles. The network parameters are adjusted online via gradient-based algorithm. Orlowska-Kowalska *et al.* [21] developed an adaptive speed controller based on a fuzzy neural network model with online parameter tuning ability. The neurofuzzy controller is applied for speed estimation, so as to remove mechanical speed sensors in the two-mass induction motor drive.

The recently developed error correction (ErrCor) algorithm with online training ability is introduced to design compact RBF networks. The very efficient improved second-order gradient algorithm is described for parameter adjustment in RBF networks. By comparing with neural networks and fuzzy inference systems, the paper is purposed to present the advantages of RBF networks for dynamic system design.

The paper is organized as follow. In Section II of the paper, the fundamentals of RBF networks are introduced briefly. Section III presents the relationships between RBF networks, neural networks, and fuzzy inference systems. Section IV introduces the ErrCor algorithm, as a hierarchical method of constructing the hidden layer of RBF networks. Section V derives the second-order gradient method for training RBF networks. Section VI gives experiments to test the properties of RBF networks, comparing with neural networks and fuzzy inference systems; experiments also prove the online behaviors of RBF networks.

II. Fundamentals of RBF Networks

Fig. 1 shows the three-layer architecture of the RBF network consisting of I inputs, H RBF units, and a single output unit. Notice that, for problems with multiple outputs, they can be analyzed as the combination of several subproblems, each of which has a single output unit.

Applying the data set $\boldsymbol{x}_p = \{x_{p,1}, x_{p,2}, x_{p,3}, \ldots, x_{p,i}, \ldots, x_{p,I}\}$, the basic computations of the RBF network in Fig. 1 consist of three steps:

1) Input Layer Computation: At the input layer, each input $x_{p,i}$ is scaled by the input weights $u_{i,h}$ which presents the weight connection between the ith input and RBF unit h

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

where vector $\boldsymbol{y}_{p,h} = \{y_{p,h,1}, y_{p,h,2} \ldots y_{p,h,i} \ldots y_{p,h,I}\}$ is the scaled inputs. h is the index of RBF units, from 1 to H; i is the index of inputs, from 1 to I; p is the index of training patterns, from 1 to P. In simplest approach, all the input weights $\boldsymbol{u}$ are set as "1."

2) Hidden Layer Computation: The output of RBF unit h is calculated by

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

where $\varphi_h(\bullet)$ is the activation function of RBF unit h. $\boldsymbol{c}_h$ and σ_h are the center and width, respectively, which are the key properties to describe the RBF unit h. $\|\bullet\|$ represents the computation of Euclidean norm of two vectors.

3) Output Layer Computation: The network output for pattern $\boldsymbol{x}_p$ is calculated as the sum of weighted outputs from RBF units

$$o_p = \sum_{h=1}^{H} \varphi_h(\mathbf{x}_p) w_h + w_0 \tag{3}$$

Where: w_h represents the weight value on the connection between RBF unit h and network output. w_0 is the bias weight.

III. Relationships Between RBF Networks, Neural Networks, and Fuzzy Inference Systems

A. RBF Networks and Neural Networks

Because of the similar layer-by-layer topology, it is often considered that RBF networks belong to multilayer perceptron (MLP) networks. It was proved that RBF networks can be implemented by MLP networks with increased input dimensions [22].

Except the similarity of network topologies, RBF networks and MLP networks have different properties. First, RBF networks are simpler than MLP networks which usually have more complex architectures. Second, RBF networks are often easier to be trained than MLP networks because of the simple

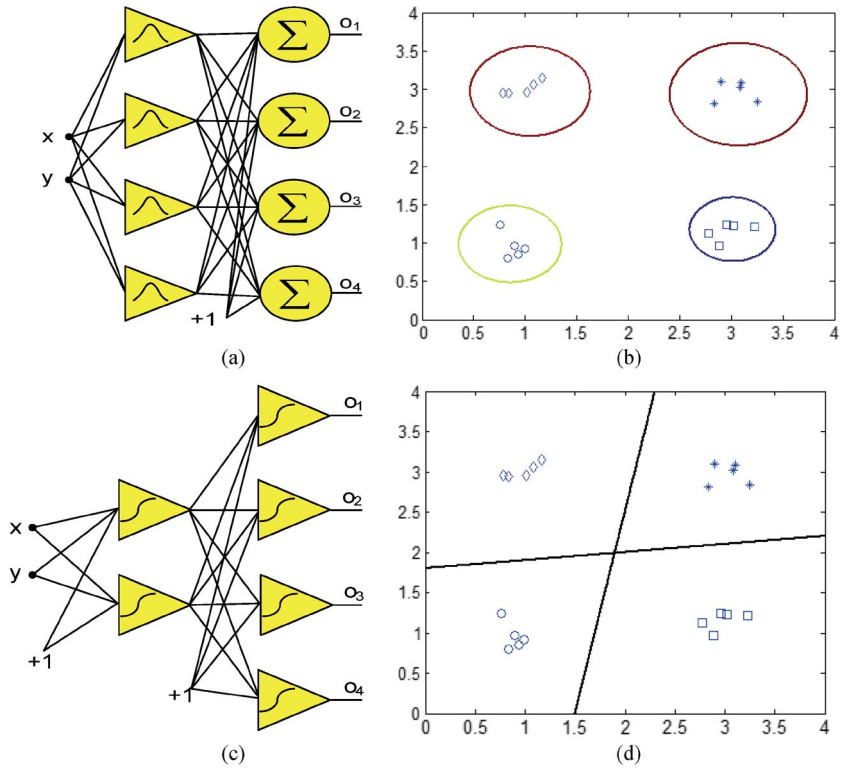


Fig. 2. Different classification mechanisms for pattern classification in two-dimension space. (a) RBF network. (b) Separation result of RBF network. (b) MLP network. (c) Separation result of MLP network.

and fixed three-layer architecture. Third, RBF networks act as local approximation networks and the network outputs are determined by specified hidden units in certain local receptive fields, while MLP networks work globally and the network outputs are decides by all the neurons. Fourth, it is essential to set correct initial states for RBF networks, while MLP networks use randomly generated parameters initially. Last and most importantly, the mechanisms of classification for RBF networks and MLP networks are different: RBF clusters are separated by hyper spheres, while in neural networks, arbitrarily shaped hyper surfaces are used for separation. In the simple two-dimension case as shown in Fig. 2, the RBF network in Fig. 2(a) separates the four clusters by circles or ellipses [Fig. 2(b)], while the neural network in Fig. 2(c) does the separation by lines [Fig. 2(d)].

B. RBF Networks and Fuzzy Inference Systems

The original design of RBF networks was somehow similar to TSK fuzzy inference systems [23], as shown in Fig. 3.

- Both models have weighted sum or weighted average as network outputs
- The number of hidden units of RBF networks can be the same as the number of IF-THEN fuzzy rules in fuzzy inference systems
- The receptive filed functions of RBF networks perform the similar mapping like the membership functions do in fuzzy inference systems

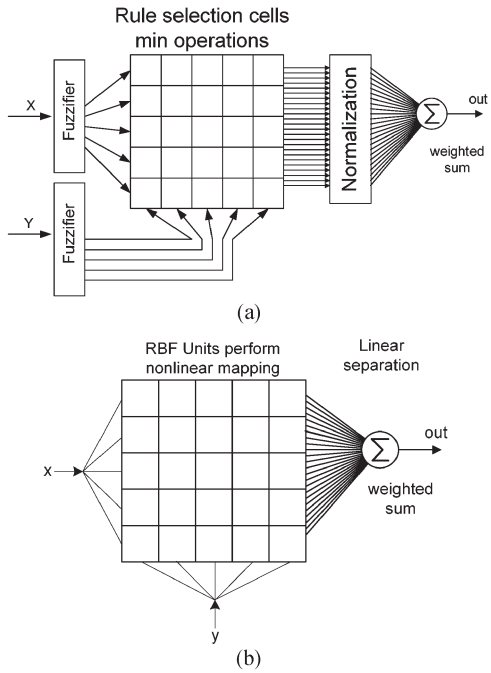


Fig. 3. Similar architectures of fuzzy inference systems and RBF networks. (a) TSK fuzzy system. (b) RBF networks.

Like fuzzy inference systems, the original RBF networks can be directly designed based on a given data set.

- The number of RBF units is equal to the number of patterns or clusters
- Each pattern is applied as the center of related RBF unit
- No training process is required

With this approach, RBF networks can be considered as direct replacement of TSK fuzzy inference systems with Gaussian membership functions. In both TSK fuzzy inference systems and RBF networks, better approximation accuracy can be obtained if systems are tuned with learning process. Similarities between RBF networks and fuzzy inference systems were presented in literatures: Roger and Sun [24] proved the equivalence between RBF networks and fuzzy inference systems; Jin and Sendhoff [25] proposed a method to extract interpretable fuzzy rules from RBF networks; Li and Hori [26] developed an algorithm using RBF networks to interpret the fuzzy rules.

C. Improved RBF Networks

To design more compact and efficient RBF networks, the approach described above was further improved by several methods of RBF network constructions. Moody and Darken [27] applied self-organized selection to determine the centers and widths of receptive fields. Wu and Chow [28] proposed an extended self-organizing map to optimize the number of RBF units. Chen *et al.* [29] presented an orthogonal least square (LS) algorithm to evaluate the optimal number of hidden units. Hwang and Bang [30] constructed the hidden layer of RBF networks by an improved adaptive pattern classifier. Orr [31] introduced a regularized forward selection method, as the combination of forward subset selection and zero-order regularization, to select the centers of RBF networks.

Further improvements were possible by introducing learning algorithms to adjust parameters of RBF networks. The simplest learning algorithm is the linear LS method, which works only for output weights adjusting and performs poorly for nonlinear cases. Iterative regression [32] and singular value decomposition [33] enhance the nonlinear performance of output layer. Based on gradient decent concept, lots of methods [34], [35] have developed to perform "deeper" training on RBF networks because, besides output weights, more parameters, such as centers and widths of RBF units, are adjusted during the learning process. First-order gradient methods have very limited search ability and take a long time for convergence. Kalman filter training algorithm provides similar performance with first-order gradient descent method, but it significantly improves the training speed [36]. Genetic algorithm [37] 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, particularly when the search space is huge.

In conclusion, the design of RBF networks consists of two important parts: (1) network construction; (2) parameter adjustment. In the following two sections, we will introduce our recently developed ErrCor algorithm for network construction and improved second-order (ISO) algorithm for parameter adjustment.

IV. RBF Network Construction

Like other nonlinear networks, RBF networks face the same controversy to choose the number of RBF units: too few RBF units cannot get acceptable approximations, while too many RBF units lead to expensive computation and may cause overfitting problem [9], [12], [38]. In this section, we will introduce a recently developed ErrCor method, purposed to find proper size of RBF networks and initial centers of RBF units. Then, two examples are presented to test the efficiency of ErrCor algorithm by comparing with other algorithms.

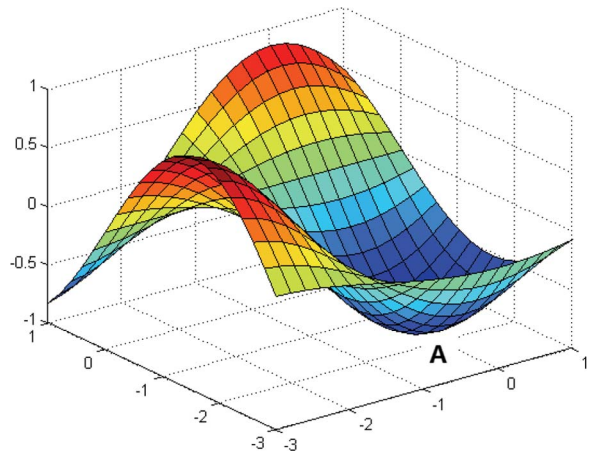


Fig. 4. Desired surface.

A. Error Correction Algorithm

To illustrate the basic idea of the ErrCor algorithm, let us have an example to approximate the simple surface shown in Fig. 4, obtained by

$$z(x, y) = \sin x + \cos y. \tag{4}$$

As shown in Fig. 4, there are three main peaks and valleys in the surface. Considering the peak shape of the output of RBF unit with kernel function (2), at least three RBF units are required for approximation. In the following steps, let us build the RBF network from scratch using the ErrCor algorithm.

1) Consider the initial outputs of the RBF network as 0. In this case, Fig. 4 not only presents the desired surface, but also describes the error surface between desired outputs and actual outputs. By going through the data set of current error surface in Fig. 4, the lowest valley marked as point $\boldsymbol{A}$ can be found.
2) Add the first RBF unit and set its initial center as the coordinate of point $\boldsymbol{A}$. Initial width and weight are "1," as shown in Fig. 8(a).
3) Train the RBF network until convergence [Fig. 8(b)]. Fig. 5 shows the approximation result of the network with one RBF unit. By comparing with the error surface in Fig. 5(b), one may notice that the lowest valley in Fig. 4 (point $\boldsymbol{A}$) is eliminated.
4) Go through the data set of error surface in Fig. 5(b) and find the location of the lowest valley marked as point $\boldsymbol{B}$.
5) Add the second RBF unit and set its initial center equal to the coordinate of point $\boldsymbol{B}$; also using "1" as its initial width and weight. Keep the rest of the RBF network the same as it was constructed in step 3), as shown in Fig. 8(c).

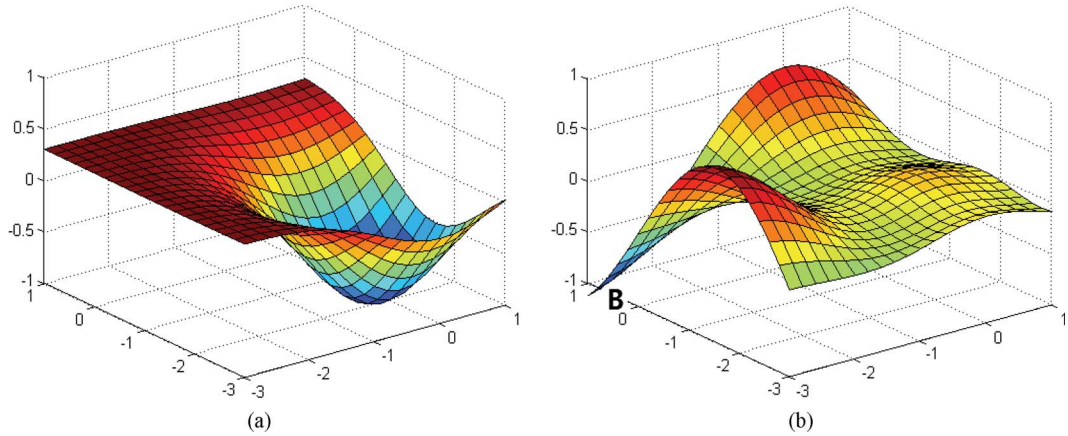


Fig. 5. Approximation results with 1 RBF unit. (a) Approximated surface. (b) Error surface.

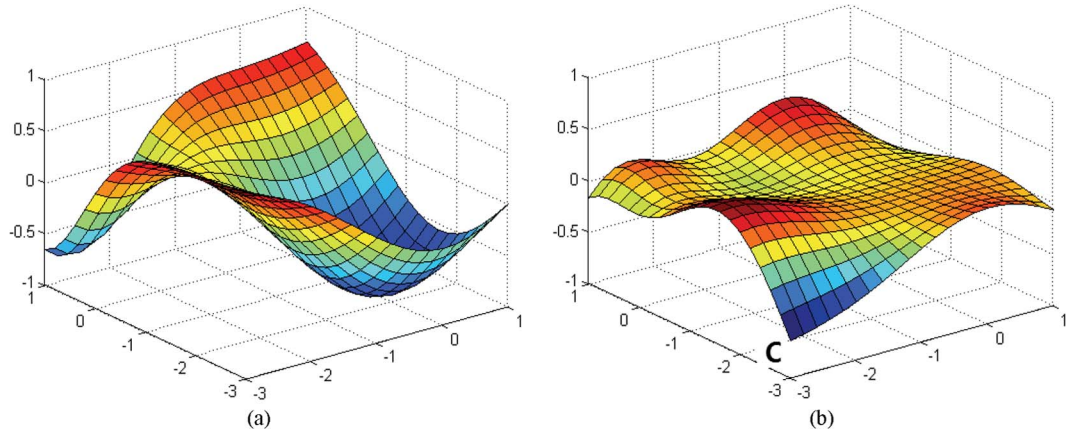


Fig. 6. Approximation results with 2 RBF units. (a) Approximated surface. (b) Error surface.

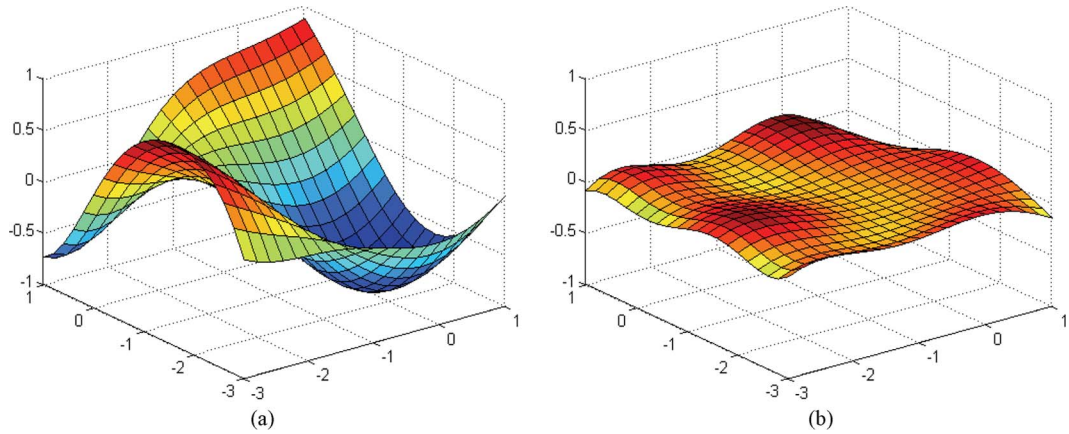


Fig. 7. Approximation results with 3 RBF units. (a) Approximated surface. (b) Error surface.

6) Train the increased RBF network until convergence [Fig. 8(d)]. Fig. 6 presents the approximation result. Again, the lowest valley in the previous error surface [point $\boldsymbol{B}$ in Fig. 5(b)] has disappeared.
7) Repeat the process from steps 4) to 6), the lowest valley in current error surface [point $\boldsymbol{C}$ in Fig. 6(b)] is corrected, by adding the third RBF unit. The result is shown in Fig. 7.

Fig. 8 shows the building process of RBF networks, based on the procedure described in step 1) to step 7).

One may notice that the ErrCor algorithm described from steps 1) to 7) can find the locations of the three peaks and valleys in the desired surface in Fig. 4 with 3 RBF units. More accurate results can be obtained by furthering the ErrCor computation above with more RBF units.

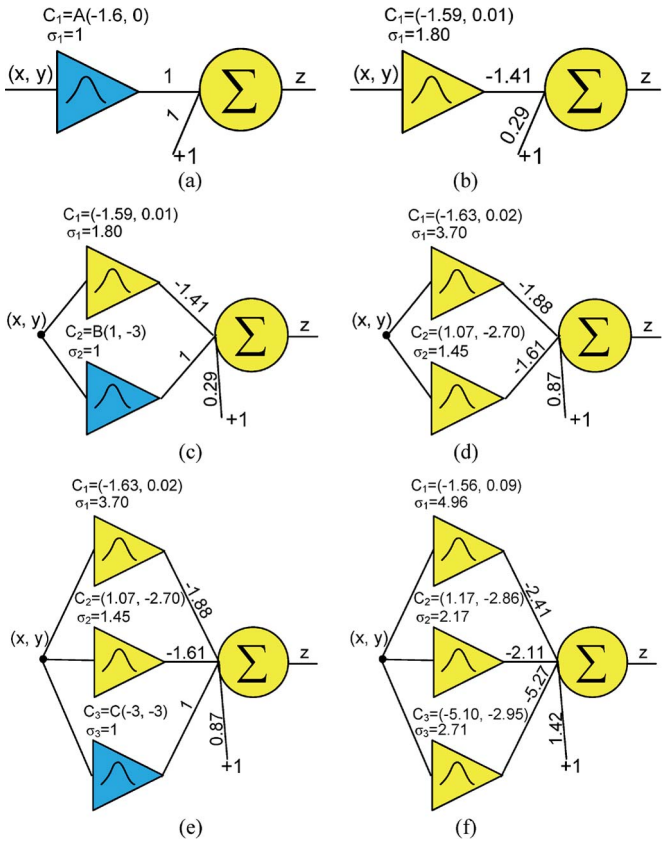


Fig. 8. Network constructions according to the procedure described from step 1) to step 7): (a) step 2); (b) step 3); (c) step 5); (d) step 6); (e) and (f) step 7). Blue RBF unit is newly added and initialed by ErrCor algorithm. All the input weights are set as "1" and not adjusted during learning process.

B. Comparison With Other Algorithms

To illustrate the efficiency of the ErrCor algorithm for RBF network construction, let us have two examples to make comparison between different algorithms for designing RBF networks. The training process used in the two examples will be discussed in the next section.

The first example is aimed to solve the Boston Housing problem [39]. The problem consists of 506 observations. In the experiment, for each trial, 481 observations are randomly selected (without duplication) as training data and the remaining 25 observations are used to test the trained RBF networks. The training/testing results are averaged by ten trials. In this example, the proposed ErrCor algorithm is compared with another hierarchical growing/pruning strategy (GGAP algorithm) for network construction presented in a well-cited paper [40]. In addition, other three algorithms, MRAN algorithm [41], RANEKF algorithm [42], and RAN algorithm [43], are also taken into comparison.

Fig. 9 presents the average training/testing root mean square errors, as the increasing of the number of RBF units.

From Fig. 9, one may notice that ErrCor algorithm can reach the better training/testing accuracy with much less number RBF units than other four algorithms.

The second example is the two-spiral classification problem, which is purposed to separate the two groups of twisted points (blue star points and red circle points) as shown in Fig. 10(a).

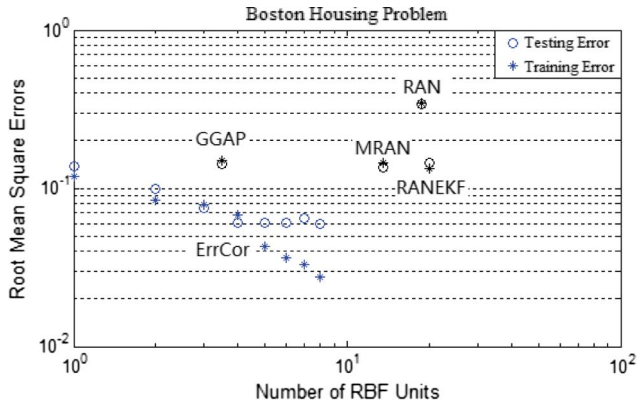


Fig. 9. Relationship between training/testing root mean square errors and the number of RBF units.

With ErrCor algorithm, to reach the training average sum square error, 0.0001, at least 30 RBF units are required for network construction, and the classification result is presented in Fig. 10(b).

In addition to the ErrCor algorithm, the experimental results of other three algorithms are extracted from literature [44]–[46] for comparison, as presented in Table I.

With the comparison results of the two examples, it is reasonable to recommend the proposed ErrCor algorithm as a very efficient algorithm for design compact RBF networks.

V. Learning Algorithms

In this section, we will introduce a newly developed ISO algorithm, which is capable of adjusting not only output weights $\boldsymbol{w}$, widths $\boldsymbol{\sigma}$, and centers $\boldsymbol{c}$, but also the input weights $\boldsymbol{u}$, as shown in Fig. 1.

By incorporating the second-order computation procedure presented in [47], the update rule of the ISO algorithm is

$$\boldsymbol{\Delta}_{k+1} = \boldsymbol{\Delta}_k - (\mathbf{Q}_k + \mu_k \mathbf{I})^{-1} \mathbf{g}_k \tag{5}$$

where k is the index of iteration, $\boldsymbol{\Delta}$ is the parameter vector, μ is the combination coefficient, $\boldsymbol{I}$ is the identity matrix, $\boldsymbol{Q}$ is the quasi Hessian matrix, and $\boldsymbol{g}$ is the gradient vector.

Quasi Hessian matrix Q is calculated as the sum of submatrices $\boldsymbol{q}_p$

$$\mathbf{Q} = \sum_{p=1}^{P} \mathbf{q}_p \tag{6}$$

where submatrix $\boldsymbol{q}_p$ is calculated by

$$\mathbf{q}_p = \mathbf{j}_p^T \mathbf{j}_p \tag{7}$$

where $\boldsymbol{j}_p$ is the Jacobian row calculated as

$$\mathbf{j}_p = \left[\frac{\partial e_p}{\partial \Delta_1} \frac{\partial e_p}{\partial \Delta_2} \cdots \frac{\partial e_p}{\partial \Delta_n} \cdots \frac{\partial e_p}{\partial \Delta_N} \right] \tag{8}$$

where n is the index of parameters, from 1 to N, where N is the number of parameters. e_p is the error calculated by

$$e_p = d_p - o_p \tag{9}$$

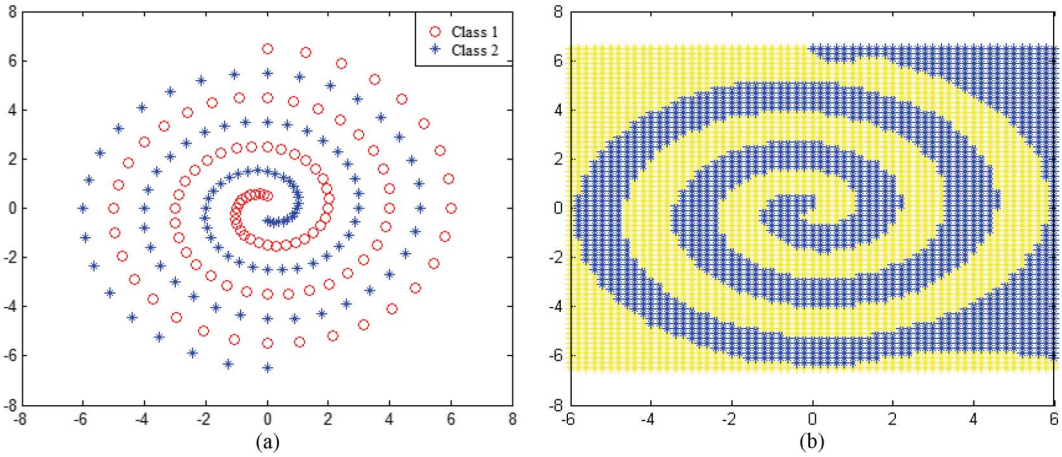


Fig. 10. Two-spiral problem (a) and the generalization result (b) obtained by the ErrCor algorithm with 30 RBF units.

TABLE I
COMPARISON OF NETWORK SIZES REQUIRED FOR SOLVING TWO-SPIRAL PROBLEM USING DIFFERENT ALGORITHMS

Network Construction Methods	Number of RBF units
RBF-MLP networks [44]	74
Orthonormalization procedure [45]	64
Weighted norm method [46]	70
ErrCor algorithm	30

where $\boldsymbol{d}$ is the desired outputs obtained from data set and $\boldsymbol{o}$ is the actual output calculated by (3).

Gradient vector $\boldsymbol{g}$ is calculated as the sum of subvectors $\boldsymbol{\eta}_p$

$$\mathbf{g} = \sum_{p=1}^{P} \boldsymbol{\eta}_p \tag{10}$$

where subgradient vector $\boldsymbol{\eta}_p$ is calculated by

$$\boldsymbol{\eta}_p = \mathbf{j}_p^T e_p. \tag{11}$$

Considering the four types of parameters, including input weights $\boldsymbol{u}$, output weights $\boldsymbol{w}$, centers $\boldsymbol{c}$, and widths $\boldsymbol{\sigma}$, the elements of Jacobian row $\boldsymbol{j}_p$ in (8) can be rewritten as

$$\mathbf{j}_p = \left[\frac{\partial e_p}{\partial u_{i,h}} \cdots \frac{\partial e_p}{\partial w_0} \cdots \frac{\partial e_p}{\partial w_h} \cdots \frac{\partial e_p}{\partial c_{h,i}} \cdots \frac{\partial e_p}{\partial \sigma_h} \right]. \tag{12}$$

By combining (1)–(3) and (9), and using the differential chain rule, elements of Jacobian row $\boldsymbol{j}_p$ are calculated by

$$\frac{\partial e_p}{\partial u_{i,h}} = \frac{2w_h \varphi_h(\mathbf{x}_p) x_{p,i}(x_{p,i} u_{i,h} - c_{h,i})}{\sigma_h} \tag{13}$$

$$\frac{\partial e_p}{\partial w_0} = -1 \tag{14}$$

$$\frac{\partial e_p}{\partial w_h} = -\varphi_h(\mathbf{x}_p) \tag{15}$$

$$\frac{\partial e_p}{\partial c_{h,i}} = -\frac{2w_h \varphi_h(\mathbf{x}_p)(x_{p,i} u_{i,h} - c_{h,i})}{\sigma_h} \tag{16}$$

$$\frac{\partial e_p}{\partial \sigma_h} = -\frac{w_h \varphi_h(\mathbf{x}_p) \|\mathbf{x}_p \times \mathbf{u}_h - \mathbf{c}_h\|^2}{\sigma_h^2}. \tag{17}$$

With (13)–(17), all the Jacobian row elements in (12) for pattern p can be obtained. Then, the related sub quasi Hessian matrix $\boldsymbol{q}_p$ and subgradient vector $\boldsymbol{\eta}_p$ can be computed by (7) and (11), respectively.

Being different from traditional Levenberg Marquardt algorithm [48], the ISO algorithm does not require Jacobian matrix storage and multiplication. All elements of quasi Hessian matrix $\boldsymbol{Q}$ and gradient vector $\boldsymbol{g}$ are computed directly using (6) and (10). This computation routine can be applied to handle problems with basically unlimited number of training patterns.

VI. EXPERIMENTAL RESULTS

To design the dynamic systems with good performance, it is important to choose the network models with:

- *Good generalization ability*: the generalization ability evaluates the quality of responses to the new patterns which are not used for system design. For a given network model, as the increasing of network size, the generalization ability often becomes better firstly; when the network size reaches certain point, the generalization ability gets saturated or unpredictably worse [12].
- *Strong tolerance to input noise*: in really system design, input signals are often not completely clean; instead, they consist of original signals and noises. The tolerance to input noise represents the difference of responses when both original signals and noised signals are applied as inputs. The stronger the tolerance is, the smaller the difference will be.
- *Online learning ability*: the online process is an opposite concept of traditional offline design. For offline design, the whole systems have to be redesigned from scratch when new data set are introduced. Differently, for online process, the systems can be updated based on the previous design parameters: if the previous data set are not important any more (in some time various systems), only the new data set take part in system updating; otherwise, the whole data set should be considered (in the experiment C followed).

Three problems are applied to test the abilities of RBF networks from the point of the three requirements above for dynamic system design. In all the problems, ErrCor algorithm

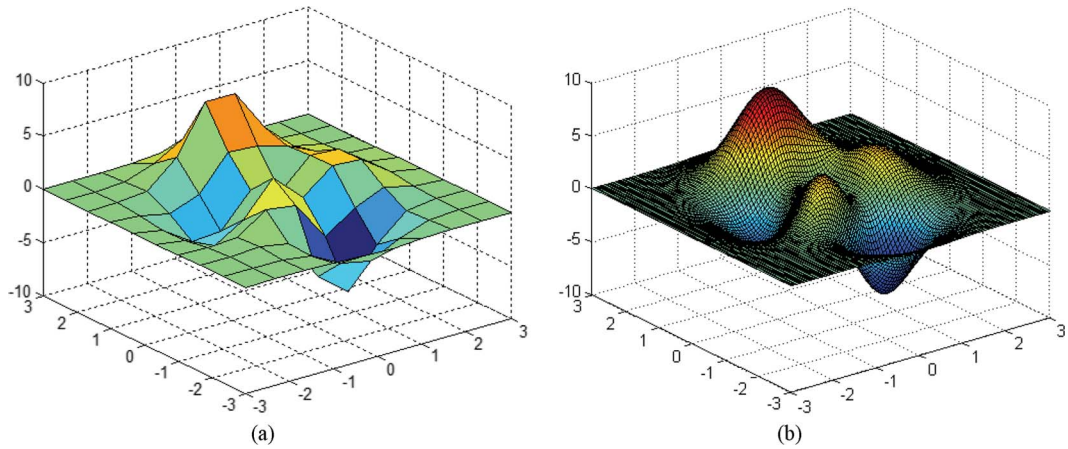


Fig. 11. Peak surface with different number of points. (a) 10 × 10 points. (b) 100 × 100 points.

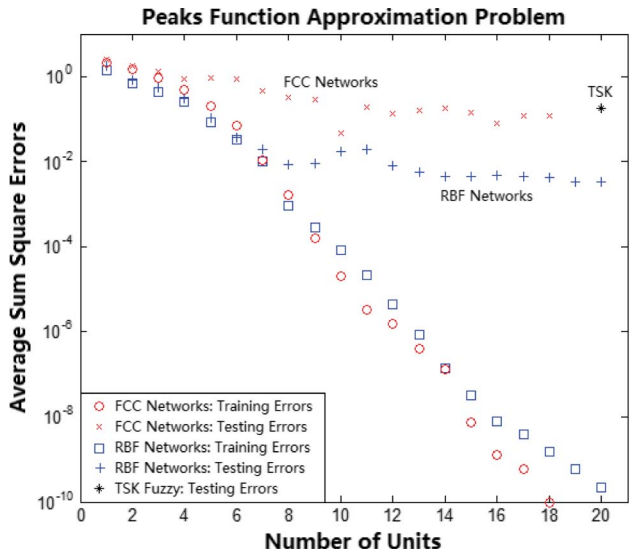


Fig. 12. Approximation results of three methods: for FCC networks, x-coordinate is the number of hidden neurons; for fuzzy inference systems, x-coordinate is equal to 20, as the number of membership functions; for RBF networks, x-coordinate is the number of RBF units.

combined with the ISO computation is applied for constructing and training RBF networks.

A. Generalization Ability

Peak problem comes from the MATLAB function *peaks*. The surface in Fig. 11(a) consists of 10 × 10 points. The purpose of peak problem is to use the 100 point in Fig. 11(a) to approximate the surface with 100 × 100 points in the same range [Fig. 11(b)].

For traditional neural networks, fully connected cascade (FCC) networks [49] and neuron-by-neuron algorithm [50] are applied to training. For each neural network topology, the training process is repeated for 10 times with randomly generated initial weights, and the results are obtained as the average values of the ten trials. For fuzzy inference systems, TSK architecture [23] and ten triangular membership functions in each direction (20 totally) are used for the approximation. Fig. 12 presents the approximation results of the three methods. Notice that there is no training process for fuzzy systems.

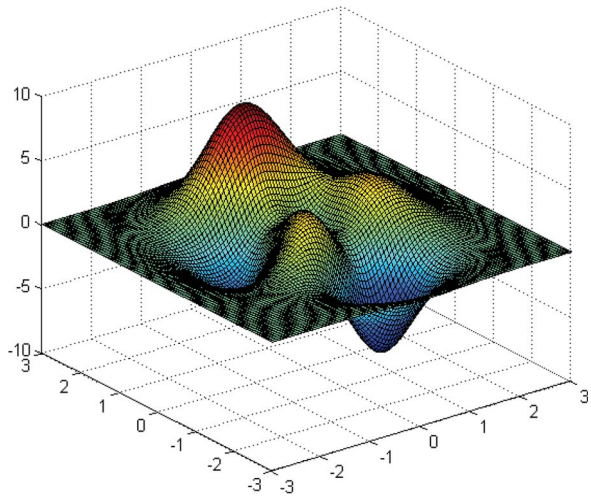


Fig. 13. Approximation result of fully connected cascade (FCC) neural network with 20 bipolar sigmoidal activation functions: $E_{\mathrm{Train}} = 1.920 \times 10^{-10}$ and $E_{\mathrm{Test}} = 2.922 \times 10^{-3}$.

Based on the experimental results presented in Fig. 12, there could be several observations as follows.

1) As the increase of hidden units, the training errors of both FCC networks and RBF networks are decreasing.
2) As the increase of hidden units, the testing errors of both FCC networks and RBF network are decreasing at first, and then get saturated. It is possible that, for a single trial, the generalization ability of FCC networks is better than RBF networks (Figs. 13 and 14), but for the average results shown in Fig. 12, the generalization ability of FCC networks is worse than RBF networks.
3) The TSK fuzzy architecture gets the smallest error for fitting the sampling points, but it requires 20 membership functions, and its generalization result is worse than both FCC networks and RBF networks with much less number of hidden units (Fig. 15).

Figs. 13–15, respectively show the generalization results of FCC networks (best one in ten trials), RBF networks, and TSK fuzzy systems, each of which has 20 activation/membership functions.

One may conclude that RBF networks get the much more stable generalization ability than traditional neural networks and better generalization than TSK fuzzy systems.

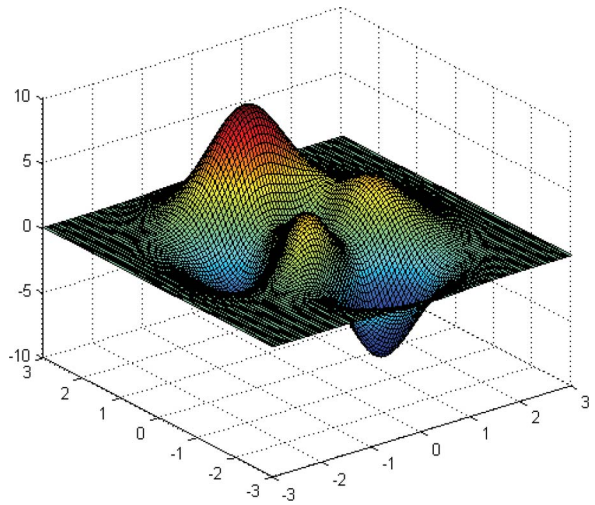


Fig. 14. Approximation result of RBF network with 20 RBF units: $E_{\mathrm{Train}} = 2.241 \times 10^{-10}$ and $E_{\mathrm{Test}} = 3.271 \times 10^{-3}$.

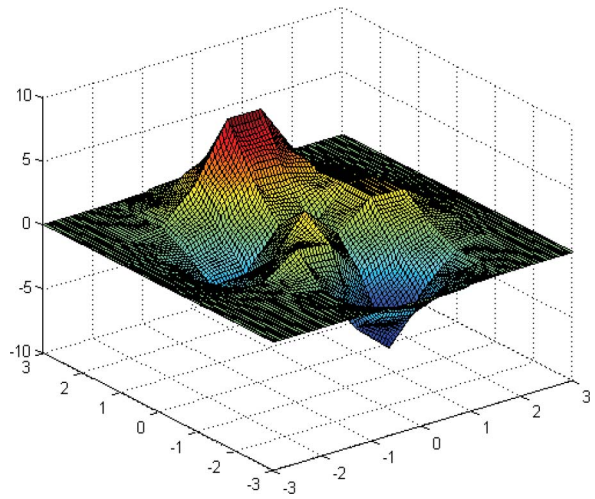


Fig. 15. Approximation result of TSK fuzzy system with ten membership functions in each direction (20 totally): $E_{\mathrm{Train}} = 1.688 \times 10^{-30}$ and $E_{\mathrm{Test}} = 1.761 \times 10^{-1}$.

B. Input Noise Rejection

Character image recognition problem is applied to test the input noise rejection ability of both RBF networks and traditional neural networks. As shown in Fig. 16, there are ten character images from "K" to "T" in each of the eight columns. Each character image consists of $8 \times 7 = 56$ pixels which are normalized in Jet degree between -1 to 1 (-1 for blue and 1 for red). The first column (from left) is the original character image data without noise and used as training patterns; while the remaining 7 seven columns are noised and used as testing patterns. The strength of noise is calculated by

$$NP_i = P_0 + i \times \delta \tag{18}$$

where P_0 is the original character image data in the 1st column (from left); NP_i is the image data with ith level noise and i is the noise level from 1 to 7, related with the noised images from the second column to the eighth column (left to right) in Fig. 16. δ is the randomly generated noise between $[-0.5, 0.5]$.

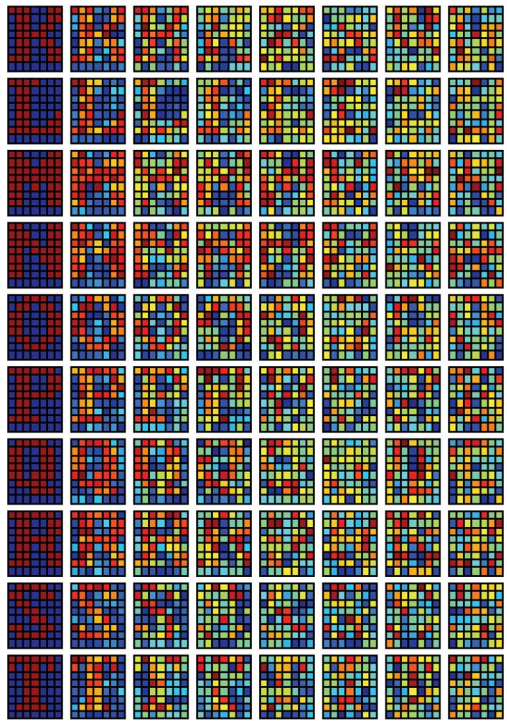


Fig. 16. Character images with different noise levels from 0 to 7 in left-to-right order (one data set in 100 groups).

In this experiment, both traditional networks and RBF networks will be built based on the training patterns (first column), and then tested by noised patterns (from second column to eighth column). For each noise level, the testing will be repeated for 100 times with randomly regenerated noise.

Using traditional neural networks, the MLP network, 56-10, is applied for training. Table II below shows the testing results of the trained MLP network. One may notice that incorrect recognition happens when images with second level noise are tested.

The RBF network used for solving this problem consists of ten RBF units with initial centers corresponding to the ten images without noise (first column), respectively. After training process, noised images are applied to test the trained RBF network. The performance of trained RBF network is shown in Table III below. One may notice that recognition error appears when fourth level noised patterns are tested.

Fig. 17 shows the average success rates of two types of network architectures in the character image recognition problem. One may notice that RBF networks (red solid line) perform more robust and have better input noise rejection ability than traditional neural networks (blue dash line).

C. Online Training

For problems where training data change dynamically, online training is necessary. Algorithm has the online training ability if it is designed hierarchically. In the experiment, the online updating of the designed RBF networks for new patterns is illustrated by the forward kinematics problem [51].

TABLE II
SUCCESS RATES OF THE TRAINED MLP NETWORK FOR CHARACTER IMAGE RECOGNITION

Data / Char	Noise level 1	Noise level 2	Noise level 3	Noise level 4	Noise level 5	Noise level 6	Noise level 7
"K"	100%	84%	53%	44%	42%	24%	24%
"L"	100%	96%	71%	51%	32%	26%	25%
"M"	100%	97%	79%	61%	45%	41%	39%
"N"	100%	79%	55%	44%	36%	32%	29%
"O"	100%	95%	80%	59%	53%	42%	35%
"P"	100%	100%	97%	87%	82%	75%	57%
"Q"	100%	100%	100%	100%	99%	96%	93%
"R"	100%	100%	97%	86%	75%	72%	58%
"S"	100%	95%	75%	55%	34%	30%	25%
"T"	100%	97%	71%	54%	43%	31%	30%

TABLE III
SUCCESS RATES OF THE TRAINED RBF NETWORK FOR CHARACTER IMAGE RECOGNITION

Data / Char	Noise level 1	Noise level 2	Noise level 3	Noise level 4	Noise level 5	Noise level 6	Noise level 7
"K"	100%	100%	100%	100%	91%	88%	83%
"L"	100%	100%	100%	100%	99%	95%	94%
"M"	100%	100%	100%	99%	96%	85%	82%
"N"	100%	100%	100%	97%	89%	83%	80%
"O"	100%	100%	100%	100%	100%	96%	96%
"P"	100%	100%	100%	98%	97%	94%	86%
"Q"	100%	100%	100%	100%	100%	98%	94%
"R"	100%	100%	100%	97%	93%	87%	76%
"S"	100%	100%	100%	100%	100%	98%	97%
"T"	100%	100%	100%	100%	100%	100%	96%

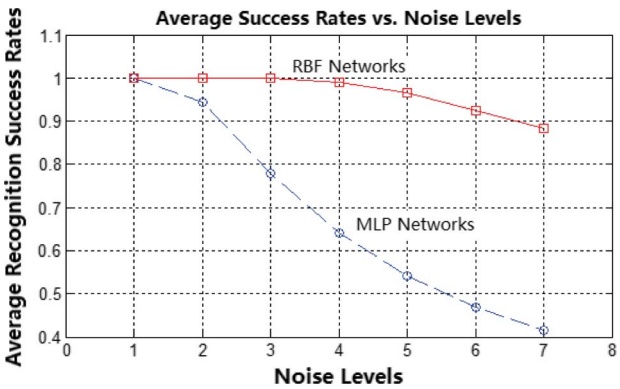


Fig. 17. Average recognition success rates of the trained MLP network and RBF network under different levels of noised inputs.

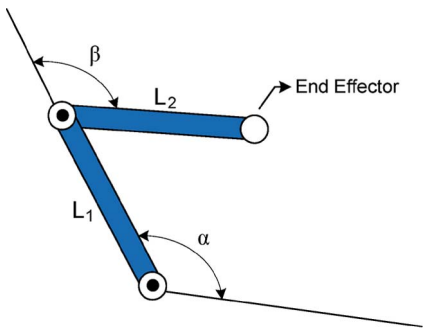


Fig. 18. Tow-link planar manipulator.

The forward kinematics problem is purposed to simulate the movement of robot's end effectors and locate the position when joint angles changes. Fig. 18 shows the two-link planar manipulator.

As shown in Fig. 18, for 2-D forward kinematics problem, the coordinates of end effector are calculated by

$$x = L_1 \cos\alpha + L_2 \cos(\alpha + \beta) \tag{19}$$

$$y = L_1 \sin\alpha + L_2 \sin(\alpha + \beta) \tag{20}$$

where (x, y) is the coordinate of the end effector marked in the Fig. 18. α and β are joint angles. L_1 and L_2 are the lengths of arms. In this experiment, let us set $L_1 = L_2 = 1$ and all training/testing data are generated by (20).

To emphasize the online learning ability of RBF networks, the experiment is organized in two steps: (1) Generate 49 training patterns and 961 testing patterns, with parameters α and β uniformly distributed in range [0, 3]; (2) Extend the range of parameters α and β from [0, 3] to [0, 6], so that 120 new training patterns and 2760 testing patterns are generated (uniformly distributed) and combined with the original training patterns and testing patterns, respectively.

All the training and testing patterns are visualized in Figs. 19 and 20 below. Only y-dimension is considered in the experiment.

First, by applying the ErrCor algorithm, the training/testing average sum square error trajectories of step 1 are obtained as shown in Fig. 21.

From Fig. 21, it can be seen that, when the number of RBF units increases to 3 (point **C** in Fig. 21), the RBF network can reach the desired accuracy approximation, 0.01.

For the step (2) of the experiment, two training procedures are performed. The one is the online training process, starting from the trained network with 3 RBF units in step (1), marked as point **C** in Fig. 21; the other is the offline training process, starting from scratch. The error trajectories of both online and offline training processes are presented in Fig. 22.

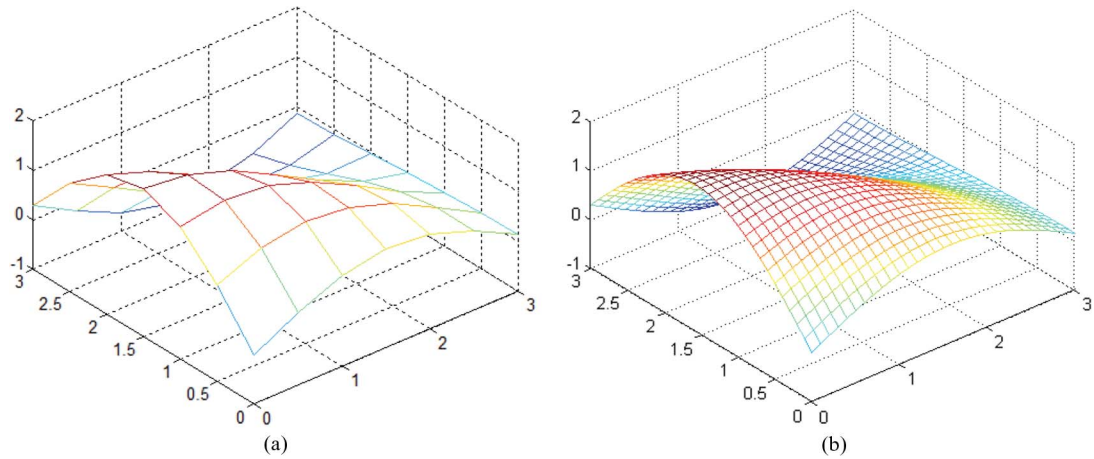


Fig. 19. Forward kinematics, step (1). (a) Training data set, 49 points. (b) Testing data set, 961 points. Parameters α and β are uniformly distributed in range [0, 3].

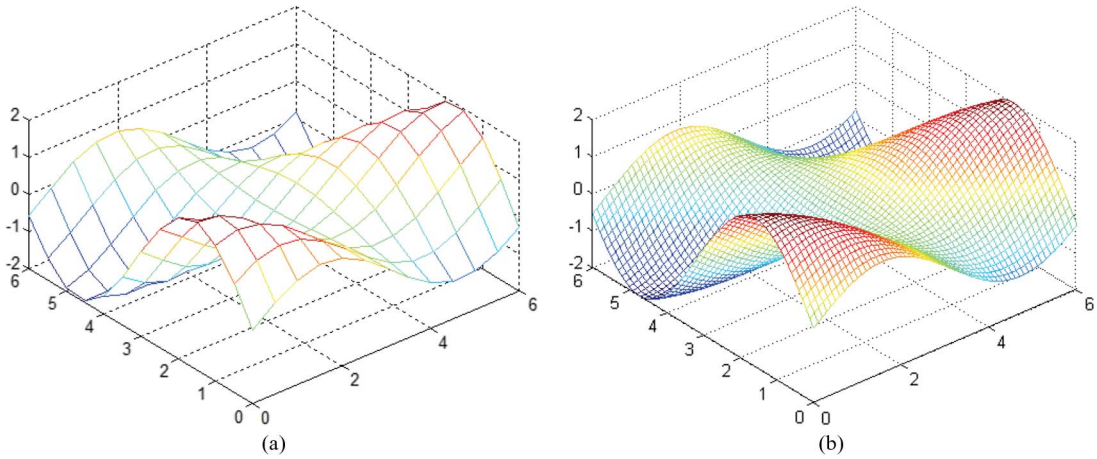


Fig. 20. Forward kinematics, step (2). (a) Training data set, 169 points. (b) Testing data set, 3721 points. Parameters α and β are uniformly distributed in range [0, 6].

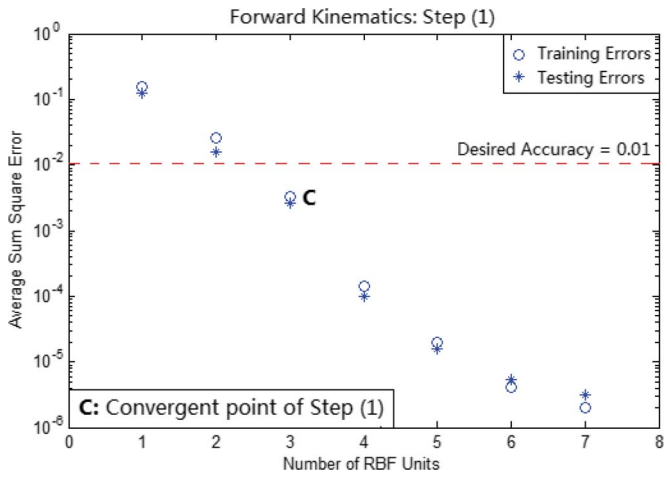


Fig. 21. Step (1): error trajectories as the increase of RBF units. The marked point **C** is the convergent result of step (1) and the trained RBF network at this point will be used as the initial condition of step (2) for online training.

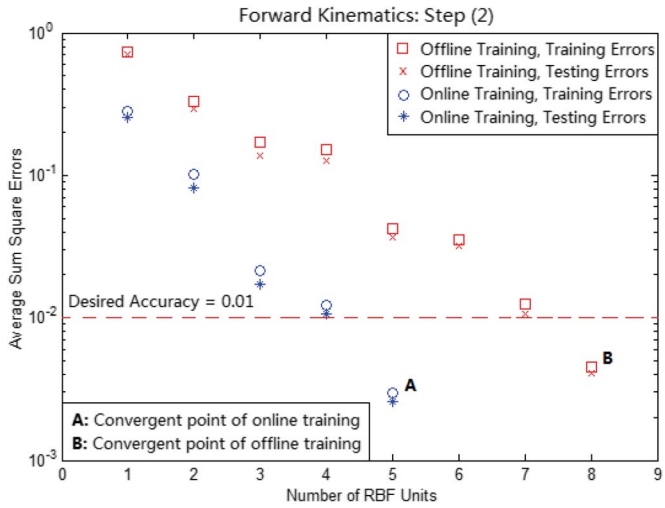


Fig. 22. Step (2): blue circles and stars present the error trajectories of the online training process, while red squares and marks show the error trajectories of the offline training process.

As the experimental results shown in Fig. 22, it can be noticed that, for step (2), the online training process works quite well, and it reaches the desired accuracy (0.01) when the fifth RBF unit is added (point **A** in Fig. 22). For the offline training process, 8 RBF units are required for convergence (point **B** in Fig. 22). Notice that, even though totally 8 RBF units are required for both training procedures, the special

TABLE IV
COMPARISON OF NEURAL NETWORKS, RBF NETWORKS, AND FUZZY INFERENCE SYSTEMS

Methods / Properties	Neural Networks	Fuzzy Inference Systems	RBF Networks
Network construction	Complex; no clues for network selection	Easy; Fixed design procedure	Easy; Fixed three-layer architecture
Parameter adjustment	Complex	Not required	Easier than neural networks
Generalization Ability	Smooth surface; very good generalization but not stable	Raw surface	Smooth surface; Stable and good generalization
Input noise tolerance	Acceptable	Not compared	Very good
Online learning	Yes	Yes	Yes

online expertise makes the ErrCor algorithm quite suitable for building dynamic systems [18]–[21].

VII. CONCLUSION

For nonlinear compensation in dynamic systems, networks should have good generalization ability and strong tolerance to input noise. Furthermore, according to the study on the recent literatures, the online learning behavior is attracting more and more attentions in designing time-variant adaptive control systems. The paper is aimed to recommend RBF networks for dynamic system design, by comparing with traditional neural networks and fuzzy inference systems.

In this paper, the recently developed ErrCor algorithm was introduced as a robust method to build very compact RBF networks. Combining with the ISO computation, the design procedure becomes more efficient.

Based on the comparison in Section III and experimental results in Section VI, Table IV concludes the properties of neural networks, fuzzy inference systems, and RBF networks.

With the advantages of easy design, stable and good generalization ability, good tolerance to input noise, and online learning ability, RBF networks are strongly recommended as an efficient and reliable way of designing dynamic systems.

The ErrCor algorithm is implemented in the training tool which can be downloaded freely from the following website: http://www.eng.auburn.edu/~wilambm/nnt/index.htm.

REFERENCES

[1] R. J. Wai, J. D. Lee, and K. L. Chuang, "Real-time PID control strategy for Maglev transportation system via particle swarm optimization," *IEEE Trans. Ind. Electron.*, vol. 58, no. 2, pp. 629–646, Feb. 2011.

[2] M. A. S. K. Khan and M. A. Rahman, "Implementation of a wavelet-based MRPID controller for benchmark thermal system," *IEEE Trans. Ind. Electron.*, vol. 57, no. 12, pp. 4160–4169, Dec. 2010.

[3] R. Muszynski and J. Deskur, "Damping of torsional vibrations in high-dynamic industrial drives," *IEEE Trans. Ind. Electron.*, vol. 57, no. 2, pp. 544–552, Feb. 2010.

[4] K. Kiyong, P. Rao, and J. A. Burnworth, "Self-tuning of the PID controller for a digital excitation control system," *IEEE Trans. Ind. Appl.*, vol. 46, no. 4, pp. 1518–1524, Jul./Aug. 2010.

[5] A. Cuenca, J. Salt, A. Sala, and R. Piza, "A delay-dependent dual-rate PID controller over an ethernet network," *IEEE Trans. Ind. Informat.*, vol. 7, no. 1, pp. 18–29, Feb. 2011.

[6] Y. Z. Li and K. M. Lee, "Thermohydraulic dynamics and fuzzy coordination control of a microchannel cooling network for space electronics," *IEEE Trans. Ind. Electron.*, vol. 58, no. 2, pp. 700–708, Feb. 2011.

[7] M. Suetake, I. N. Silva, and A. Goedtel, "Embedded DSP-based compact fuzzy system and its application for induction-motor V/f speed control," *IEEE Trans. Ind. Electron.*, vol. 58, no. 3, pp. 750–760, Mar. 2011.

[8] R. H. Abiyev and O. Kaynak, "Type 2 fuzzy neural structure for identification and control of time-varying plants," *IEEE Trans. Ind. Electron.*, vol. 57, no. 12, pp. 4147–4159, Dec. 2010.

[9] B. M. Wilamowski, "Human factor and computational intelligence limitations in resilient control systems," in *Proc. 3rd ISRCS*, Idaho Falls, ID, Aug. 10–12, 2011, pp. 5–11.

[10] A. Bhattacharya and C. Chakraborty, "A shunt active power filter with enhanced performance using ANN-based predictive and adaptive controllers," *IEEE Trans. Ind. Electron.*, vol. 58, no. 2, pp. 421–428, Feb. 2011.

[11] N. Cotton and B. M. Wilamowski, "Compensation of nonlinearities using neural networks implemented on inexpensive microcontrollers," *IEEE Trans. Ind. Electron.*, vol. 58, no. 3, pp. 733–740, Mar. 2011.

[12] B. M. Wilamowski, "Neural network architectures and learning algorithms: How not to be frustrated with neural networks," *IEEE Ind. Electron. Mag.*, vol. 3, no. 4, pp. 56–63, Dec. 2009.

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

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

[15] J. Lin and R. J. Lian, "Intelligent control of active suspension systems," *IEEE Trans. Ind. Electron.*, vol. 58, no. 2, pp. 618–628, Feb. 2011.

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

[17] M. Pucci and M. Cirrincione, "Neural MPPT control of wind generators with induction machines without speed sensors," *IEEE Trans. Ind. Electron.*, vol. 58, no. 1, pp. 37–47, Jan. 2011.

[18] Q. N. Le and J. W. Jeon, "Neural-network-based low-speed-damping controller for stepper motor with an FPGA," *IEEE Trans. Ind. Electron.*, vol. 57, no. 9, pp. 3167–3180, Sep. 2010.

[19] C. Xia, C. Guo, and T. Shi, "A neural-network-identifier and fuzzy-controller-based algorithm for dynamic decoupling control of permanent-magnet spherical motor," *IEEE Trans. Ind. Electron.*, vol. 57, no. 8, pp. 2868–2878, Aug. 2010.

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

[21] T. Orlowska-Kowalska, M. Dybkowski, and K. Szabat, "Adaptive sliding-mode neuro-fuzzy control of the two-mass induction motor drive without mechanical sensors," *IEEE Trans. Ind. Electron.*, vol. 57, no. 2, pp. 553–564, Feb. 2010.

[22] B. M. Wilamowski and R. C. Jaeger, "Implementation of RBF type networks by MLP networks," in *Proc. IEEE Int. Conf. Neural Netw.*, Washington, DC, Jun. 3–6, 1996, pp. 1670–1675.

[23] T. T. Xie, H. Yu, and B. M. Wilamowski, "Replacing fuzzy systems with neural networks," in *Proc. IEEE HSI Conf.*, Rzeszow, Poland, May 13–15, 2010, pp. 189–193.

[24] J. S. Roger and C. T. Sun, "Functional equivalence between radial basis function networks and fuzzy inference systems," *IEEE Trans. Neural Netw.*, vol. 4, no. 1, pp. 156–159, Jan. 1993.

[25] Y. Jin and B. Sendhoff, "Extracting interpretable fuzzy rules from RBF networks," *Neural Process. Lett.*, vol. 17, no. 2, pp. 149–164, Apr. 2003.

[26] W. Li and Y. Hori, "An algorithm for extracting fuzzy rules based on RBF neural network," *IEEE Trans. Ind. Electron.*, vol. 53, no. 4, pp. 1269–1276, Jun. 2006.

[27] J. Moody and C. J. Darken, "Learning with localized receptive fields," in *Proc. Connectionist Models Summer School*, D. Touretzky, G. Hinton, and T. Sejnowski, Eds., 1988, pp. 133–142.

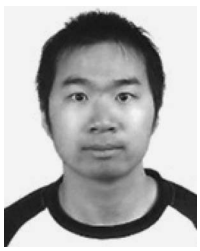
[28] S. Wu and T. W. S. Chow, "Induction machine fault detection using SOM-based RBF neural networks," *IEEE Trans. Ind. Electron.*, vol. 51, no. 1, pp. 183–194, Feb. 2004.

[29] 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.

[30] Y. S. Hwang and S. Y. Bang, "An efficient method to construct a radial basis function neural network classifier," *Neural Netw.*, vol. 10, no. 8, pp. 1495–1503, Nov. 1997.

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

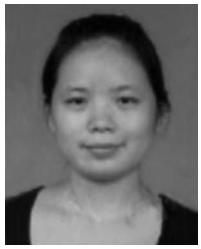
[32] B. M. Wilamowski, "Modified EBP algorithm with instant training of the hidden layer," in *Proc. IEEE IECON*, New Orleans, LA, Nov. 9–14, 1997, pp. 1097–1101.
[33] Z. Hong, "Algebraic feature extraction of image for recognition," *Pattern Recognit.*, vol. 24, no. 3, pp. 211–219, 1991.
[34] E. S. 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.
[35] N. B. Karayiannis, "Reformulated radial basis neural networks trained by gradient descent," *IEEE Trans. Neural Netw.*, vol. 10, no. 3, pp. 657–671, May 1999.
[36] D. Simon, "Training radial basis neural networks with the extended Kalman filter," *Neurocomputing*, vol. 48, no. 1–4, pp. 455–475, Oct. 2002.
[37] 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.
[38] B. M. Wilamowski and H. Yu, "Neural network learning without backpropgation," *IEEE Trans. Neural Netw.*, vol. 21, no. 11, pp. 1793–1803, Nov. 2010.
[39] C. Blake and C. Merz, *UCI Repository of Machine Learning Databases*, Dept. Inform. Comput. Sci., Univ. California, Irvine, 1998.
[40] 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. B, Cybern.*, vol. 34, no. 6, pp. 2284–2292, Dec. 2004.
[41] N. Sundararajan, P. Saratchandran, and Y. W. Li, *Radial Basis Function Neural Networks With Sequential Learning: MRAN and Its Applications*. Singapore: World Scientific, 1999.
[42] V. Kadirkamanathan and M. Niranjan, "A function estimation approach to sequential learning with neural networks," *Neural Comput.*, vol. 5, no. 6, pp. 954–975, Nov. 1993.
[43] J. Platt, "A resource-allocating network for function interpolation," *Neural Comput.*, vol. 3, no. 2, pp. 213–225, Jun. 1991.
[44] N. Chaiyaratana and A. M. S. Zalzala, "Evolving hybrid RBF-MLP networks using combined genetic/unsupervised/supervised learning," in *Proc. UKACC Int. Conf. Control*, Swansea, U.K., Sep. 1–4, 1998, vol. 1, pp. 330–335.
[45] W. Kaminski and P. Strumillo, "Kernel orthonormalization in radial basis function neural networks," *IEEE Trans. Neural Netw.*, vol. 8, no. 5, pp. 1177–1183, Sep. 1997.
[46] R. Neruda and P. Kudová, "Learning methods for radial basis function *networks*," *Future Gener. Comput. Syst.*, vol. 21, no. 7, pp. 1131–1142, Jul. 2005.
[47] 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.
[48] 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.
[49] H. Yu and B. M. Wilamowski, "Efficient and reliable training of neural networks," in *Proc. IEEE HSI Conf.*, Catania, Italy, May 21–23, 2009, pp. 109–115.
[50] B. M. Wilamowski, N. J. Cotton, O. Kaynak, and G. Dundar, "Computing gradient vector and Jacobian matrix in arbitrarily connected neural networks," *IEEE Trans. Ind. Electron.*, vol. 55, no. 10, pp. 3784–3790, Oct. 2008.
[51] A. Malinowski and H. Yu, "Comparison of embedded system design for industrial applications," *IEEE Trans. Ind. Informat.*, vol. 7, no. 2, pp. 244–254, May 2011.



Hao Yu (S'10) received the M.S. degree in electrical engineering from Huazhong University of Science and Technology, Wuhan, China, in 2006. He is currently working toward the Ph.D. degree in electrical engineering at Auburn University, Auburn, AL.

He is a Research Assistant in the Department of Electrical and Computer Engineering, Auburn University. His current research interests include computational intelligence, neural networks, and computer-aided design.

Mr. Yu serves as a Reviewer for the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS and IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS.



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

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

Stanisław Paszczyñski (SM'05) received the M.S., Ph.D., and D.Sc. degrees in electronics from Warsaw University of Technology, Warsaw, Poland, in 1972, 1979, and 1991, respectively.

In 1985, he joined the Microelectronics Institute, Catholic University of Leuven, Leuven, Belgium. From 1989 to 1991, he was a Visiting Assistant Professor in the Science Department, University of Texas, San Antonio. From 1992 to 1999, he was a Professor in the Electrical Engineering Department, Rzeszów University of Technology, Rzeszów, Poland. He is currently an Associate Professor in the Department of Distributed Systems, University of Information Technology and Management, Rzeszów, where he works on network traffic analysis and modeling, as well as on agent technology use in network throughput increase.

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

He was with 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 (2000–2004) and the President of the IEEE Industrial Electronics Society (2004–2005). He served as an Associate Editor for numerous journals. He was the Editor-in-Chief of the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS from 2007 to 2010, and he currently serves as the Editor-in-Chief of IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS.