ELSEVIER

Fuzzy Sets and Systems 111 (2000) 285–306

FUZZY sets and systems

www.elsevier.com/locate/fss

Water bath temperature control with a neural fuzzy inference network

Chin-Teng Lin*, Chia-Feng Juang, Chung-Ping Li

Department of Electrical and Control Engineering, National Chiao-Tung University, Hsinchu, Taiwan, ROC

Received July 1996; received in revised form March 1998

Abstract

Although multilayered backpropagation neural networks (BPNN) have demonstrated high potential in the nonconventional branch of adaptive control, its long training time usually discourages their applications in industry. Moreover, when they are trained on-line to adapt to plant variations, the overtuned phenomenon usually occurs. To overcome the weakness of the BPNN, we propose a neural fuzzy inference network (NFIN) in this paper suitable for adaptive control of practical plant systems in general, and for adaptive temperature control of a water bath system in particular. The NFIN is inherently a modified TSK (Takagi–Sugeno–Kang)-type fuzzy rule-based model possessing neural network's learning ability. In contrast to the general adaptive neural fuzzy networks, where the rules should be decided in advance before parameter learning is performed, there are no rules initially in the NFIN. The rules in the NFIN are created and adapted as on-line learning proceeds via simultaneous structure and parameter identification. The NFIN has been applied to a water bath temperature control system. As compared to the BPNN under the same training procedure, the control results show that not only can the NFIN greatly reduce the training time and avoid the overtuned phenomenon, but the NFIN also has perfect regulation ability. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Neural fuzzy network; Backpropagation network; TSK fuzzy rules; Recursive least square; Structure/parameter learning; Similarity measure; Water bath temperature control

1. Introduction

The advent of the fuzzy logic controllers (FLC) [4,8,9] and the neural controllers based on multilayered neural networks [15,30] has inspired new resources for the possible realization of better and more efficient control. They offer a key advantage over traditional adaptive control systems [1,16]. That is, they do not require mathematical models of the plants. In this paper, a neural fuzzy inference network (NFIN) is proposed to combine the advantages of fuzzy logic and neural networks. The NFIN is a fuzzy rule-based network possessing neural network's learning ability. A major characteristic of the network is that no preassignment and design of the rules are required. The rules are constructed automatically during the on-line operation.

* Corresponding author.

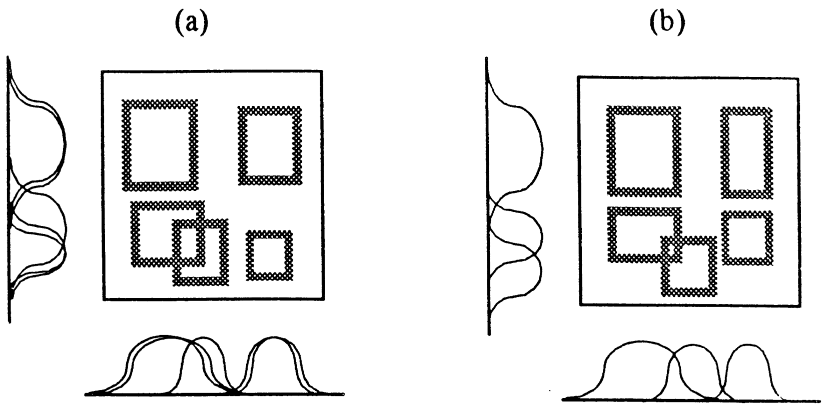

Fig. 1. Fuzzy partition of a two-dimensional input space (a) clustering-based partitioning. (b) Proposed aligned clustering-based partitioning.

Two learning phases, the structure as well as the parameter learning phases, are adopted on-line for the construction task. One important task in the structure identification of the NFIN is the partition of the input space, which influences the number of fuzzy rules generated. Unlike the general clustering-based partitioning method [22,29], where the formed membership functions projected from different clusters may be very similar as shown in Fig. 1(a), this paper proposes a novel on-line input space partitioning method which is an aligned clustering-based approach. This method can produce a partition result like the one shown in Fig. 1(b). Basically, it aligns the clusters formed in the input space, so it reduces not only the number of rules but also the number of membership functions under a prespecified accuracy requirement. The proposed method creates only the significant membership functions on the universe of discourse of each input variable based on a fuzzy measure. It can thus generate significant fuzzy rules from numerical data dynamically.

Another feature of the NFIN is that it can optimally determine the consequent part of fuzzy if–then rules during the structure learning phase. A fuzzy rule of the following form is adopted in our system initially,

$$\text{Rule } j\text{: IF } x_1 \text{ is } A_{i1} \text{ and } \dots \text{ and } x_n \text{ is } A_{in} \text{ THEN } y_i \text{ is } m_i, \tag{1}$$

where x_i and y_i are the input and output variables, respectively, A_{ij} is a fuzzy set, and m_i is the position of a symmetric membership function of the output variable with its width neglected during the defuzzification process. This type of fuzzy rule is used as the main body of the NFIN. We call a NFIN consisting of such kind of rules as a *basic NFIN*. By monitoring the change of the network output error, additional terms (the linear terms used in the consequent part of the TSK model) will be included when necessary to further reduce the output error. If it is decided that some auxiliary terms should be added to the consequent part during the on-line learning process, a projection-based correlation measure will be performed on each rule to select the most significant terms to be incorporated into the rule. This consequent identification process is employed in conjunction with the precondition identification process to reduce both the number of rules and the number of consequent terms. For the parameter identification scheme, the consequent parameters are tuned by the recursive least-squares (RLS) algorithm, and the precondition parameters are tuned by the backpropagation learning algorithm. Both the structure and parameter learning are done simultaneously to achieve fast learning.

The proposed NFIN is used for temperature control in this paper. Temperature control is an important factor in many process control systems. If the temperature is too high or too low, the final product is seriously affected. Therefore, it is necessary to reach some desired temperature points quickly and avoid large overshoot. Since the process control systems are often nonlinear and tend to change in an unpredictable way, it is not easy to control them accurately. To verify if the NFIN has good control performance on the temperature control system, we compare it to the backpropagation neural network (BPNN) under the same training process via a water bath temperature control system. In the training process, we adopt both off-line and on-line training schemes. First, a network (BPNN or NFIN) is trained off-line to learn the inverse dynamics model of a plant, and then the network is configured as a feedforward network controller to the plant. Next, a conventional on-line training scheme is used to adapt the network to the practical environment.

This paper is organized as follows. In Section 2, the structure and the learning algorithm of the NFIN are proposed. In Section 3, the configuration of the NFIN-based control and the training process are introduced. In Section 4, the control results of using the NFIN on the water bath temperature control problem are presented. Conclusion is made in Section 5.

2. Neural fuzzy inference network (NFIN)

In this section, the structure of the NFIN as shown in Fig. 2 is introduced. This six-layered network realizes a fuzzy model of the following form

$$\text{Ru } i\text{:}\;\; \text{IF } x_1 \text{ is } A_{i1} \text{ and } \dots \text{ and } x_n \text{ is } A_{in} \text{ THEN } y \text{ is } m_{i0} + a_{ij}x_j + \cdots,$$

where A_{ij} is a fuzzy set, m_{i0} is the center of a symmetric membership function on y, and a_{ij} is a consequent parameter. It is noted that unlike the traditional TSK model [23,27] where all the input variables are used in the output linear equation, only the significant ones are used in the NFIN, i.e., some a_{ij}'s in the above fuzzy rules are zero. With this six-layered network structure of the NFIN, we shall define the function of each node of the NFIN in Section 2.1, and the learning algorithm of the NFIN in Section 2.2.

2.1. Structure of the NFIN

The NFIN consists of nodes, each of which has some finite fan-in of connections represented by weight values from other nodes and fan-out of connections to other nodes. Associated with the fan-in of a node is an integration function f which serves to combine information, activation, or evidence from other nodes. This function provides the net input for this node

$$\textit{net-input} = f(u_1^{(k)}, u_2^{(k)}, \dots, u_p^{(k)};\; w_1^{(k)}, w_2^{(k)}, \dots, w_p^{(k)}),$$

where $u_1^{(k)}, u_2^{(k)}, \dots, u_p^{(k)}$ are inputs to this node, and $w_1^{(k)}, w_2^{(k)}, \dots, w_p^{(k)}$ are the associated link weights. The superscript (k) in the above equation indicates the layer number. This notation will also be used in the following equations. A second action of each node is to produce an activation value as a function of its net-input,

$$\textit{output} = o_i^{(k)} = a(\textit{net-input}) = a(f),$$

where $a(\cdot)$ denotes the activation function. We shall next describe the functions of the nodes in each of the six layers of the NFIN.

Layer 1: No computation is done in this layer. Each node in this layer, which corresponds to one input variable, only transmits input values to the next layer directly. That is,

$$f = u_i^{(1)} \quad \text{and} \quad a^{(1)} = f. \tag{2}$$

From the above equation, the link weight in layer one ($w_i^{(1)}$) is unity.

Layer 2: Each node in this layer corresponds to one linguistic label (small, large, etc.) of one of the input variables in Layer 1. In other words, the membership value which specifies the degree to which an input

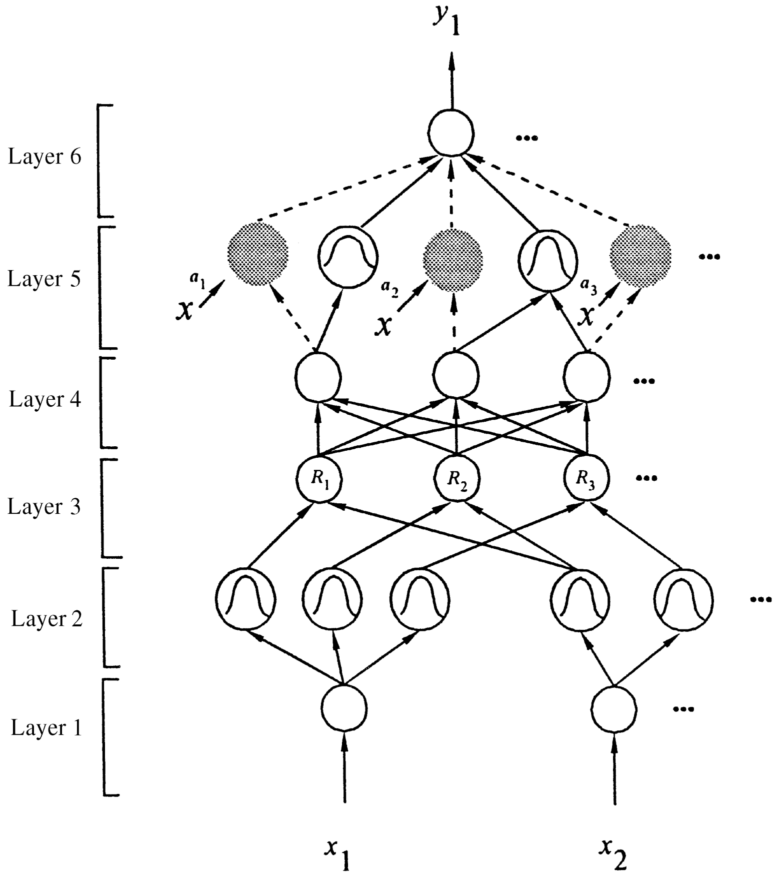

Fig. 2. The structure of the neural fuzzy inference network (NFIN).

value belongs to a fuzzy set is calculated in Layer 2. With the choice of Gaussian membership function, the operations performed in this layer are

$$f(u_{ij}^{(2)}) = -\frac{(u_i^{(2)} - m_{ij})^2}{\sigma_{ij}^2} \quad \text{and} \quad a^{(2)}(f) = e^f, \tag{3}$$

where m_{ij} and σ_{ij} are, respectively, the center (or mean) and the width (or variance) of the Gaussian membership function of the jth term of the ith input variable x_i. Hence, the link weight in this layer can be interpreted as m_{ij}. Unlike other clustering-based partitioning methods, where each input variable has the same number of fuzzy sets, the number of fuzzy sets of each input variable is not necessarily identical in the NFIN.

Layer 3: A node in this layer represents one fuzzy logic rule and performs precondition matching of a rule. Here, we use the following AND operation for each Layer-3 node,

$$f(u_i^{(3)}) = \prod_i u_i^{(3)} = \mathrm{e}^{-[D_i(\boldsymbol{x}-\boldsymbol{m}_i)]^{\mathrm{T}}[D_i(\boldsymbol{x}-\boldsymbol{m}_i)]} \quad \text{and} \quad a^{(3)}(f) = f, \tag{4}$$

where n is the number of Layer 2 nodes participating in the IF part of the rule, $D_i = diag(1/\sigma_{i1}, 1/\sigma_{i2}, \ldots, 1/\sigma_{in})$, and $\boldsymbol{m}_i = (m_{i1}, m_{i2}, \ldots, m_{in})^{\mathrm{T}}$. The link weight in Layer 3 ($w_i^{(3)}$) is then unity. The output f of a Layer-3 node represents the firing strength of the corresponding fuzzy rule.

Layer 4: The number of nodes in this layer is equal to that in Layer 3, and the firing strength calculated in Layer 3 is normalized in this layer by

$$f(u_i^{(4)}) = \sum_i u_i^{(4)} \quad \text{and} \quad a^{(4)}(f) = u_i^{(4)}/f. \tag{5}$$

Like Layer 3, the link weight ($w_i^{(4)}$) in this layer is unity, too.

Layer 5: This layer is called the consequent layer. Two types of nodes are used in this layer, and they are denoted as blank and shaded circles in Fig. 2, respectively. The node denoted by a blank circle (blank node) is the essential node representing a fuzzy set (described by a Gaussian membership function) of the output variable. Only the center of each Gaussian membership function is delivered to the next layer for the LMOM (local mean of maximum) defuzzification operation [2], and the width is used for output clustering only. Different nodes in Layer 4 may be connected to a same blank node in Layer 5, meaning that the same consequent fuzzy set is specified for different rules. The function of the blank node is

$$f = \sum_i u_i^{(5)} \quad \text{and} \quad a^{(5)}(f) = f \cdot a_{0i}, \tag{6}$$

where $a_{0i} = m_{0i}$, the center of a Gaussian membership function. As to the shaded node, it is generated only when necessary. Each node in Layer 4 has its own corresponding shaded node in Layer 5. One of the inputs to a shaded node is the output delivered from Layer 4, and the other possible inputs (terms) are the input variables from Layer 1. The shaded node function is

$$f = \sum_j a_{ji} x_j \quad \text{and} \quad a^{(5)}(f) = f \cdot u_i^{(5)}, \tag{7}$$

where the summation is over the significant terms connected to the shaded node only, and a_{ji} is the corresponding parameter. Combining these two types of nodes in Layer 5, we obtain the whole function performed by this layer as

$$a^{(5)}(f) = \left(\sum_j a_{ji} x_j + a_{0i} \right) u_i^{(5)}. \tag{8}$$

Layer 6: Each node in this layer corresponds to one output variable. The node integrates all the actions recommended by Layer 5 and acts as a defuzzifier with

$$f(u_i^{(6)}) = \sum_i u_i^{(6)} \quad \text{and} \quad a^{(6)}(f) = f. \tag{9}$$

2.2. Learning algorithms for the NFIN

Two types of learning, structure and parameter learning, are used concurrently for constructing the NFIN. The structure learning includes both the precondition and consequent structure identification of a fuzzy if–then rule. Here the precondition structure identification corresponds to the input space partitioning and can be formulated as a combinational optimization problem with the following two objectives: to minimize the number of rules generated and to minimize the number of fuzzy sets on the universe of discourse of each input variable. As to the consequent structure identification, the main task is to decide when a new membership function is generated for the output variable and which significant terms (input variables) should be added to the consequent part (a linear equation) when necessary. For the parameter learning, the parameters of the linear equations in the consequent parts are adjusted by the RLS algorithm, and the parameters in the precondition part are adjusted by the backpropagation algorithm to minimize a given cost function. There are no rules (i.e., no nodes in the network except the input/output nodes) in the NFIN initially. They are created dynamically as learning proceeds upon receiving on-line incoming training data by performing the following learning processes simultaneously,

A. Input/output space partitioning.
B. Construction of fuzzy rules.
C. Optimal consequent structure identification.
D. Parameter identification.

In the above, processes A, B, and C belong to the structure learning phase and process D belongs to the parameter learning phase. The details of these learning processes are described in the rest of this section.

A. Input/output space partitioning

The way the input space is partitioned determines the number of rules extracted from training data as well as the number of fuzzy sets on the universal of discourse of each input variable. For each incoming pattern $\boldsymbol{x}$, the strength a rule is fired can be interpreted as the degree the incoming pattern belongs to the corresponding cluster. For computational efficiency, we can use the firing strength derived in Eq. (4) directly as this degree measure,

$$F^i(\boldsymbol{x}) = \prod_i u_i^{(3)} = \mathrm{e}^{-[D_i(\boldsymbol{x}-\boldsymbol{m}_i)]^{\mathrm{T}}[D_i(\boldsymbol{x}-\boldsymbol{m}_i)]}, \tag{10}$$

where $F^i \in [0,1]$. Using this measure, we can obtain the following criterion for the generation of a new fuzzy rule. Let $\boldsymbol{x}(t)$ be the newly incoming pattern. Find

$$J = \arg \max_{1 \leqslant j \leqslant c(t)} F^j(\boldsymbol{x}), \tag{11}$$

where $c(t)$ is the number of existing rules at time t. If $F^J \leqslant \bar{F}(t)$, then a new rule is generated, where $\bar{F}(t) \in (0,1)$ is a prespecified threshold that decays during the learning process. Once a new rule is generated, the next step is to assign initial centers and widths of the corresponding membership functions. According to the first-nearest-neighbor heuristic [10], we set

$$\boldsymbol{m}_{(c(t)+1)} = \boldsymbol{x}, \tag{12}$$

$$D_{(c(t)+1)} = -\frac{1}{\beta}\, diag(1/\ln(F^J) \ldots 1/\ln(F^J)), \tag{13}$$

where $\beta \geqslant 0$ decides the overlap degree between two clusters.

After a rule is generated, the next step is to decompose the multidimensional membership function formed in Eqs. (12) and (13) to the corresponding one-dimensional membership function for each input variable. For the Gaussian membership function used in the NFIN, the task can be easily done as

$$\mathrm{e}^{-[D_i(\boldsymbol{x}-\boldsymbol{m}_i)]^{\mathrm{T}}[D_i(\boldsymbol{x}-\boldsymbol{m}_i)]} = \prod_j \mathrm{e}^{-(x_j-m_{ij})^2/\sigma_{ij}^2}, \tag{14}$$

where m_{ij} and σ_{ij} are, respectively, the projected center and width of the membership function in each input dimension. To reduce the number of fuzzy sets of each input variable and to avoid the existence of redundant ones, we should check the similarities between the newly projected membership function and the existing ones in each input dimension. Before going to the details on how this overall process works, let us consider the similarity measure first. Since bell-shaped membership functions are used in the NFIN, we use the formula of the similarity measure of two fuzzy sets with bell-shaped membership functions derived previously in [7,11–14]. Suppose the fuzzy sets to be measured are fuzzy sets A and B with membership

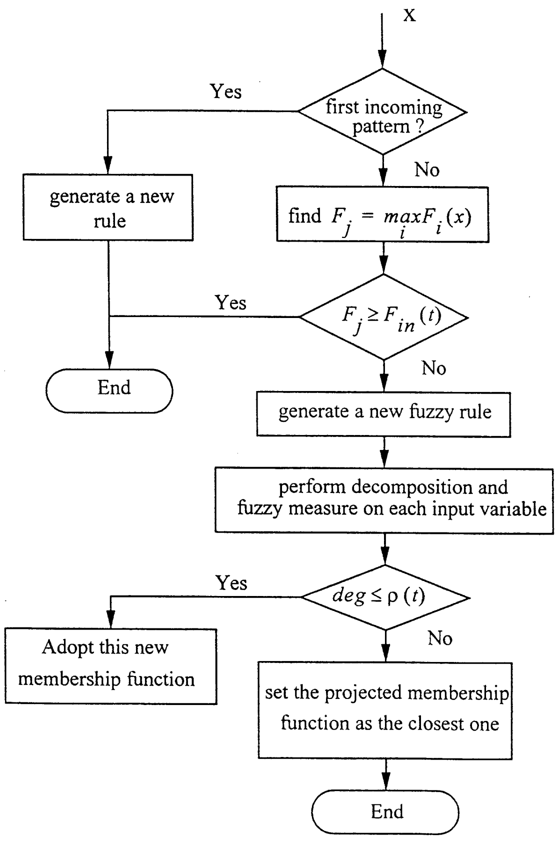
x
first incoming pattern ?
Yes
No
generate a new rule
find $F_j = \max_i F_i(x)$
$F_j \geq F_{in}(t)$
Yes
No
End
generate a new fuzzy rule
perform decomposition and fuzzy measure on each input variable
$deg \leq \rho(t)$
Yes
No
Adopt this new membership function
set the projected membership function as the closest one
End

Fig. 3. Flow chart of the algorithm for input/output space partitioning in NFIN.

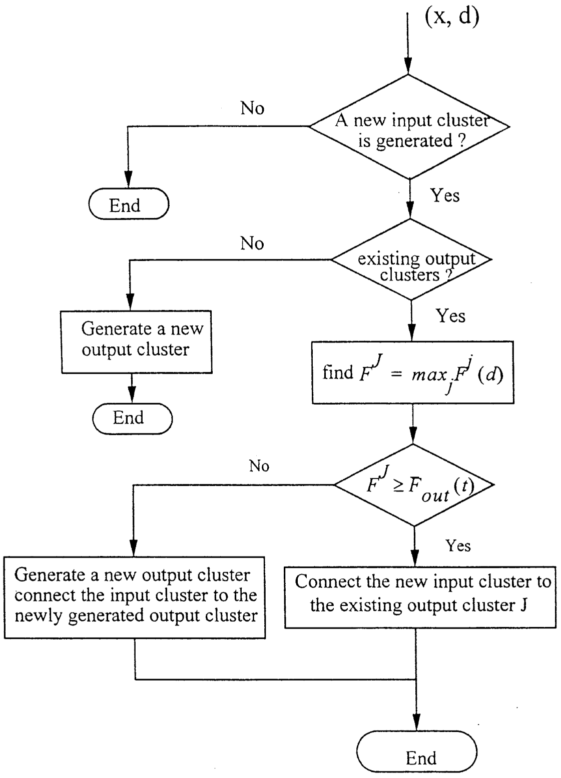
(x, d)
A new input cluster is generated ?
No
Yes
End
existing output clusters ?
No
Yes
Generate a new output cluster
End
find $F^J = \max_j F^j(d)$
$F^J \geq F_{out}(t)$
No
Yes
Generate a new output cluster connect the input cluster to the newly generated output cluster
Connect the new input cluster to the existing output cluster J
End

Fig. 4. Flow chart of the algorithm for rule construction in NFIN.

function $\mu_A(x) = \exp\{-(x-m_1)^2/\sigma_1^2\}$ and $\mu_B(x) = \exp\{-(x-m_2)^2/\sigma_2^2\}$, respectively. Assume $m_1 \geqslant m_2$, we can compute $|A \cap B|$ by:

$$|A \cap B| = \frac{1}{2}\frac{h^2(m_2 - m_1 + \sqrt{\pi}(\sigma_1+\sigma_2))}{\sqrt{\pi}(\sigma_1+\sigma_2)} + \frac{1}{2}\frac{h^2(m_2 - m_1 + \sqrt{\pi}(\sigma_1-\sigma_2))}{\sqrt{\pi}(\sigma_2-\sigma_1)} + \frac{1}{2}\frac{h^2(m_2 - m_1 - \sqrt{\pi}(\sigma_1+\sigma_2))}{\sqrt{\pi}(\sigma_1-\sigma_2)}, \tag{15}$$

where $h(x) = \max\{0, x\}$. So the approximate similarity measure is

$$E(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{\sigma_1\sqrt{\pi} + \sigma_2\sqrt{\pi} - |A \cap B|}, \tag{16}$$

where we use the fact that $|A| + |B| = |A \cap B| + |A \cup B|$.

Let $\mu(m_i, \sigma_i)$ represent the Gaussian membership function with center m_i and width σ_i. The whole algorithm for the generation of new fuzzy rules as well as fuzzy sets in each input dimension is as follows, and the corresponding flowchart is shown in Fig. 3.

Suppose no rules are existent initially.

```
  IF $\boldsymbol{x}$ is the first incoming pattern THEN do
PART 1. {Generate a new rule,
            with center $\boldsymbol{m}_1 = \boldsymbol{x}$, width $D_1 = diag(1/\sigma_{init}, \ldots, 1/\sigma_{init})$,
                where $\sigma_{init}$ is a prespecified constant.
            After decomposition, we have $n$ one-dimensional membership functions,
              with $m_{1i} = x_i$ and $\sigma_{1i} = \sigma_{init}$, $i = 1 \cdots n$.
        }
ELSE for each newly incoming $\boldsymbol{x}$, do
PART 2. {find $J = \arg\max_{1 \leqslant j \leqslant c(t)} F^j(\boldsymbol{x})$,
              IF $F^J \geqslant \bar{F}_{in}(t)$
                do nothing
              ELSE
              {$c(t+1) = c(t) + 1$,
              generate a new fuzzy rule, with
                $\boldsymbol{m}_{c(t+1)} = \boldsymbol{x}$, $D_{c(t+1)} = -1/\beta \cdot diag(1/\ln(F^J) \ldots 1/\ln(F^J))$.
              After decomposition, we have
              $m_{new\text{-}i} = x_i$, $\sigma_{new\text{-}i} = -\beta \cdot \ln(F^J)$, $i = 1 \ldots n$.
              Do the following fuzzy measure for each input variable $i$:
                {$degree(i,t) \equiv \max_{1 \leqslant j \leqslant k_i} E[\mu(m_{new\text{-}i}, \sigma_{new\text{-}i}), \mu(m_{ji}, \sigma_{ji})]$,
                where $k_i$ is the number of partitions of the $i$th input variable.
                IF $degree(i,t) \leqslant \alpha(t)$,
                THEN adopt this new membership function, and set $k_i = k_i + 1$,
                ELSE set the projected membership function as the closest one.}
        }
```

In the above algorithm, $\alpha(t)$ is a scalar similarity criterion which is monotonically decreasing such that higher similarity between two fuzzy sets is allowed in the initial stage of learning. For the output space partitioning, the same measure in Eq. (11) is used. Since the criterion for the generation of a new output cluster is related to the construction of a rule, we shall describe it together with the rule construction process in learning process *B* below.

B. Construction of fuzzy rules

As mentioned in learning process A, the generation of a new input cluster corresponds to the generation of a new fuzzy rule, with its precondition part constructed by the learning algorithm in Process A. At the same time, we have to decide the consequent part of the generated rule. Suppose a new input cluster is formed after the presentation of the current input–output training pair ($\boldsymbol{x}$, $\boldsymbol{d}$), then the consequent part is constructed by the following algorithm (see the flowchart in Fig. 4):

```
IF there are no output clusters,
  do {PART 1 in Process A, with $\boldsymbol{x}$ replaced by $\boldsymbol{d}$}
ELSE
  do {
    find $J = \arg\max_j F^j(\boldsymbol{x})$.
    IF $F^J \geqslant \bar{F}_{\mathrm{out}}(t)$
    connect input cluster $c(t+1)$ to the existing output cluster $J$,
    ELSE
    generate a new output cluster,
    do the decomposition process in PART 2 of Process A,
    connect input cluster $c(t+1)$ to the newly generated output cluster.
    }.
```

The algorithm is based on the fact that different preconditions of different rules may be mapped to the same consequent fuzzy set. Since only the center of each output membership function is used for defuzzification, the consequent part of each rule may simply be regarded as a singleton. Compared to the general fuzzy rule-based models with singleton output, where each rule has its own individual singleton value, fewer parameters are needed in the consequent part of the NFIN, especially in the case with a large number of rules.

C. Optimal consequent structure identification

Up to now, the NFIN contains fuzzy rules in the form of Eq. (1). Even though such a basic NFIN can be used directly for system modeling, a large number of rules are necessary for modeling sophisticated systems under a tolerable modeling accuracy. To cope with this problem, we adopt the spirit of the TSK model [27] in the NFIN. In the TSK model, each consequent part is represented by a linear equation of the input variables. It is reported in [23] that the TSK model can model a sophisticated system using a few rules. However, even for the TSK model, if the dimension of the input or output space is high, the number of terms used in the linear equation is large even though some terms are in fact of little significance. Hence, instead of using the linear combination of all the input variables as the consequent part, only the most significant input variables are used as the consequent terms of the NFIN. The significant terms will be chosen and added to the network incrementally any time when the parameter learning cannot improve the network output accuracy any more during the on-line learning process.

In the choice of the significant terms participated in the consequent part, since the dependence between the candidates $u_i^{(5)} \cdot x_i$ and the desired output y_m is linear ($y_m = \sum_i u_i^{(5)}(\sum_j a_{ji}^m x_j)$), we can consider the training sequences $u_i^{(5)}x_i(1), u_i^{(5)}x_i(2), \ldots, u_i^{(5)}x_i(t)$ and $y_m(1), y_m(2), \ldots, y_m(t)$ as vectors and find the correlation between $\hat{\boldsymbol{x}}_i = u_i^{(5)}[x_i(1), \ldots, x_i(t)]^{\mathrm{T}}$ and $[y_m(1), \ldots, y_m(t)]^{\mathrm{T}}$. The correlation between two vectors $\hat{\boldsymbol{x}}$ and $\boldsymbol{y}$, is estimated by the cosine value of their angle θ, $Deg = \cos^2(\theta) = (\hat{\boldsymbol{x}}^{\mathrm{T}}\boldsymbol{y})^2/(\hat{\boldsymbol{x}}^{\mathrm{T}}\hat{\boldsymbol{x}})(\boldsymbol{y}^{\mathrm{T}}\boldsymbol{y})$. If vectors $\hat{\boldsymbol{x}}$ and $\boldsymbol{y}$ are dependent then $Deg = 1$, otherwise if $\hat{\boldsymbol{x}}$ and $\boldsymbol{y}$ are orthogonal then $Deg = 0$. The main idea of the proposed choice scheme is as follows. Suppose we have chosen $k-1$ vectors from n candidates to form a space $P_{k-1} = \boldsymbol{p}_1 \oplus \boldsymbol{p}_2 \oplus \cdots \oplus \boldsymbol{p}_{k-1}$. To find the next important vector from the remaining $n-k+1$ vectors, we first project each of the remaining $n-k+1$ vectors to the null space of P_{k-1}, find the correlation value Deg between the $n-k+1$ projected vectors and $\boldsymbol{y}$, then choose the maximum one which is the kth important term of the n candidates, and finally set $P_k = \boldsymbol{p}_1 \oplus \boldsymbol{p}_2 \oplus \cdots \oplus \boldsymbol{p}_k$. Here, $\boldsymbol{p}_1 = \hat{\boldsymbol{x}}_0$ is the vector formed by the essential singleton values. To find the projected vector $\boldsymbol{p}_k$, the Gram–Schmidt orthogonalization procedure [18] is adopted as

$$\alpha_{ik} = \boldsymbol{p}_i^{\mathrm{T}}\hat{\boldsymbol{x}}_k/(\boldsymbol{p}_i^{\mathrm{T}}\boldsymbol{p}_i), \tag{17}$$

$$\boldsymbol{p}_k = \hat{\boldsymbol{x}}_k - \sum_{i=1}^{k-1} \alpha_{ik}\boldsymbol{p}_i. \tag{18}$$

If there are c rules, then we have cn candidate vectors, a large number that may lead to high computation load in the calculation of the projected vectors in the above. To reduce the computation cost and to keep the parallel-processing advantage assumed in fuzzy rule-based systems, the terms in the consequent part are selected independently for each rule; i.e. the projection operation is done only for the n vectors in each rule, not for other rules. This assumption is based upon the local property of a fuzzy rule-based system, so the vectors from different rules can be regarded as being orthogonal.

For on-line learning, to calculate the correlation degree, we have to store all the input/output sequences before these degrees are calculated. The memory required is of order $O(nt + Mt)$, where n and M are, respectively, the number of input and output variables, for every rule and output, and is huge for large t. To cope with this problem, instead of storing the input–output sequences, we store the correlation values only. Let $C_{x_i y_m}$ denote the correlation between the sequence $u_j^{(5)} x_i$ and y_m, and $C_{x_i x_p}$ the correlation between the sequence $u_j^{(5)} x_i$ and $u_j^{(5)} x_p$. For each incoming datum, these values are on-line calculated, respectively, for each rule j by

$$C^j_{x_i y_m}(t+1) = C^j_{x_i y_m}(t) + \Gamma(t)(u_j^{(5)}(t+1)x_i(t+1)y_m(t+1) - C^j_{x_i y_m}(t)), \tag{19}$$

$$C^j_{x_i x_p}(t+1) = C^j_{x_i x_p}(t) + \Gamma(t)(u_j^{(5)}(t+1)x_i(t+1)u_j^{(5)}(t+1)x_p(t+1) - C^j_{x_i x_p}(t)), \tag{20}$$

where $i, p = 0, \dots, n$, $m = 1, \dots, M$, and $C^j_{x_i y_m}(0)$ and $C^j_{x_i x_p}(0)$ are equal to zero initially. For normal correlation computation, $\Gamma(t) = 1/(t+1)$ is used, but for computation efficiency and for changing environment where the recent calculations dominate, a constant value, say $0 < \Gamma < 1$, can be used. Using the stored correlation values in Eqs. (19) and (20), we can compute the correlation values and choose the significant ones. The algorithm for this computation is described as follows.

Projection-based Correlation Measure Algorithm.

```
For each rule do,
{
For $k = 1 \dots K$, where $K$ denotes the number of terms to be selected from the $n$ candidates.
  For $i = 1 \dots n$, $i \neq i_1, \dots, i_{k-1}$, where $i_1, \dots, i_{k-1}$ denotes the terms already selected.
  Compute
```

$$\alpha(m,i) = \frac{A(m,i)}{B(m,m)}, \quad 0 \leqslant m \leqslant k-1, \tag{21}$$

$$E(i,j) = C_{x_i y_j} - \sum_{m=0}^{k-1} \alpha(m,i) E(i_m, j), \tag{22}$$

$$G(i,i) = C_{x_i x_i} - 2\sum_{m=0}^{k-1} \alpha(m,i) A(m,i) + \sum_{m=0}^{k-1}\sum_{q=0}^{k-1} \alpha(m,i)\alpha(q,i) B(m,q), \tag{23}$$

$$Deg_k(i) = \frac{E^2(i,j)}{G(i,i) C_{y_j y_j}}, \tag{24}$$

where

$$A(m,i) = C_{x_{i_m} x_i} - \sum_{l=0}^{m-1} \alpha(l, i_m) A(l,i), \tag{25}$$

$$B(l,n) = \begin{cases} C_{x_0 x_0}, & l = n = 0, \\ G(i_l, i_l), & l = n \neq 0, \\ A(n, i_l) - \sum_{m=0}^{l-1} \alpha(m, i_l) B(m,n), & 0 \leqslant n \leqslant l-1, \\ B(n,l), & 0 \leqslant n \leqslant n-1. \end{cases} \tag{26}$$

Then find $i_k \in \{1,2,\ldots,n\}$ such that

$$Deg_k(i_k) = \max_{i=1,\ldots,n;\, i\neq i_1,\ldots,\, i\neq i_{k-1}} (Deg_k(i)). \tag{27}$$

}

The procedure is terminated at the Kth step, when

$$Deg_K(i_K) \leqslant \gamma, \tag{28}$$

where $0 \leqslant \gamma \leqslant 1$ is the tolerable dependence degree, and K terms are added to the consequent part of the rule.

The consequent structure identification scheme in the NFIN is a kind of node growing method in neural networks. For the node growing method, in general there is a question of when to perform node growing. The criterion used in the NFIN is by monitoring the learning curve. When the effect of parameter learning diminished (i.e., the output error does not decrease over a period of time), then it is the time to apply the above algorithm to add additional terms to the consequent part.

D. Parameter identification

After the network structure is adjusted according to the current training pattern, the network then enters the parameter identification phase to adjust the parameters of the network optimally based on the same training pattern. Notice that the following parameter learning is performed on the whole network after structure learning, no matter whether the nodes (links) are newly added or are existent originally. The idea of backpropagation is used for this supervised learning. Considering the single-output case for clarity, our goal is to minimize the error function

$$E = \tfrac{1}{2}(y(t) - y^{\mathrm{d}}(t))^2, \tag{29}$$

where $y^{\mathrm{d}}(t)$ is the desired output, and $y(t)$ is the current output. The parameters, a_{ji}, in layer 5 are tuned by RLS as

$$P(t+1) = \frac{1}{\lambda}\left[P(t) - \frac{P(t)\boldsymbol{u}^{(5)^{\mathrm{T}}}(t+1)\boldsymbol{u}^{(5)}(t+1)P(t)}{\lambda + \boldsymbol{u}^{(5)^{\mathrm{T}}}(t+1)P(t)\boldsymbol{u}^{(5)}(t+1)}\right], \tag{30}$$

$$\boldsymbol{a}(t+1) = \boldsymbol{a}(t) + P(t+1)\boldsymbol{u}^{(5)}(t+1)(y^{\mathrm{d}}(t) - y(t)), \tag{31}$$

where $0 < \lambda \leqslant 1$ is the forgetting factor, $\boldsymbol{u}^{(5)}$ is the current input vector for nodes in layer 5, $\boldsymbol{a}$ is the corresponding parameter vector, and P is the covariance matrix. The initial parameter vector $\boldsymbol{a}(0)$ is determined in the structure learning phase and $P(0) = \sigma I$, where σ is a large positive constant. To cope with changing environment, in general, $0.9 < \lambda < 1$ is used. Also, to avoid the unstable effect caused by a small λ, we may reset $P(t)$ as $P(t) = \sigma I$ after a period of learning. As to the free parameters m_{ij} and σ_{ij} of the input membership functions in layer 2, they are updated by the backpropagation algorithm. Using the chain rule, we have

$$\frac{\partial E}{\partial m_{ij}^{(2)}} = \frac{\partial E}{\partial y}\sum_k \frac{\partial y}{\partial a_k^{(3)}}\frac{\partial a_k^{(3)}}{\partial m_{ij}^{(2)}}, \tag{32}$$

where

$$\frac{\partial E}{\partial y} = y(t) - y^{\mathrm{d}}(t), \tag{33}$$

$$\frac{\partial y}{\partial a_k^{(3)}} = a_k^{(5)} - \frac{y}{\sum_i a_i^{(3)}}, \tag{34}$$

$$\frac{\partial a_k^{(3)}}{\partial m_{ij}^{(2)}} = \begin{cases} a_k^{(2)} \dfrac{2(x_i - m_{ij})}{\sigma_{ij}^2} & \text{if term node } j \text{ is connected to rule node } k \\ 0 & \text{otherwise,} \end{cases} \tag{35}$$

and $m_{ij}^{(2)}$ is updated by

$$m_{ij}^{(2)}(t+1) = m_{ij}^{(2)}(t) - \eta \frac{\partial E}{\partial m_{ij}^{(2)}} \tag{36}$$

$$= m_{ij}^{(2)}(t) - \eta(y(t) - y^{\mathrm{d}}(t)) \sum_k \frac{\partial y}{\partial a_k^{(3)}} \frac{\partial a_k^{(3)}}{\partial m_{ij}^2}. \tag{37}$$

Similarly, we have

$$\frac{\partial E}{\partial \sigma_{ij}^{(2)}} = \frac{\partial E}{\partial y} \sum_k \frac{\partial y}{\partial a_k^{(3)}} \frac{\partial a_k^{(3)}}{\partial \sigma_{ij}^{(2)}}, \tag{38}$$

where

$$\frac{\partial a_k^{(3)}}{\partial \sigma_{ij}^{(2)}} = \begin{cases} a_k^{(2)} \dfrac{2(x_i - m_{ij})^2}{\sigma_{ij}^3} & \text{if term node } j \text{ is connected to rule node } k, \\ 0 & \text{otherwise,} \end{cases} \tag{39}$$

and $\sigma_{ij}^{(2)}$ is updated by

$$\sigma_{ij}^{(2)}(t+1) = \sigma_{ij}^{(2)}(t) - \eta \frac{\partial E}{\partial \sigma_{ij}^{(2)}}. \tag{40}$$

3. NFIN-based adaptive control

Consider a classical discrete-time single-input–single-output (SISO) plant

$$y_p(k+1) = f[y_p(k), y_p(k-1), \ldots, y_p(k-m+1), u(k), u(k-1), \ldots, u(k-n)], \tag{41}$$

where u denotes the input, y_p is the output, k is the discrete-time index, $m, n \geqslant 0$ and $m, n \in Z$, and the function $f(\cdot)$ is $f : \Re^{m+n} \to \Re$. In many practical systems, the plant input is limited in magnitude, i.e., there exist $u_{\max}$ and $u_{\min}$ such that, for any k,

$$u_{\min} \leqslant u(k) \leqslant u_{\max}. \tag{42}$$

In this paper, the task is to control the plant described in Eq. (41) with respect to a specified reference output $y_{\mathrm{ref}}(k)$ and there is no a priori knowledge regarding its dynamics, i.e., the function $f(\cdot)$ is unknown.

3.1. Network controller

Assume that the plant described in Eq. (41) is invertible, i.e., there exists a function $g(\cdot)$ such that

$$u(k) = g[y_p(k+1), y_p(k), \ldots, y_p(k-m+1), u(k-1), \ldots, u(k-n)]. \tag{43}$$

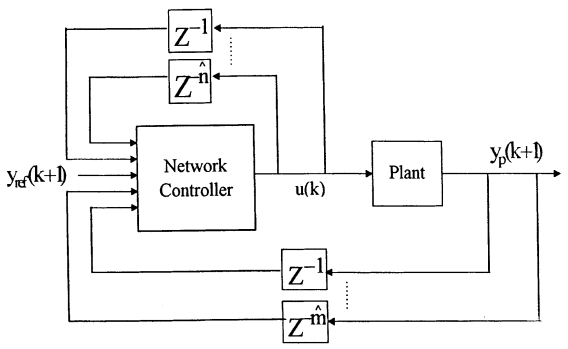

Fig. 5. Schematic diagram of the network controller.

Consider a network (BPNN or NFIN) with input vector $I(k)$, single output $\hat{u}(k)$, and an input–output relationship represented by

$$\hat{u}(k) = N(I(k)), \tag{44}$$

where

$$I(k) = [y_p(k+1), y_p(k), \dots, y_p(k - \hat{m} + 1), u(k-1), \dots, u(k - \hat{n})]^{\mathrm{T}}, \tag{45}$$

and function $N(\cdot)$ denotes the input–output mapping of the network. If the output of $N(\cdot)$ approximates the output of $g(\cdot)$ for the same input, the network can be viewed as a controller in the feedforward control path. At any instant k, the control input to the plant can be obtained from Eq. (44) by setting the input vector as

$$I'(k) = [y_{\mathrm{ref}}(k+1), y_p(k), \dots, y_p(k - \hat{m} + 1), u(k-1), \dots, u(k - \hat{n})]^{\mathrm{T}}, \tag{46}$$

where reference output $y_{\mathrm{ref}}(k+1)$ is used instead of the unknown $y_p(k+1)$ in Eq. (45).

Therefore, if the network can be trained to make its output approximates the output of the inverse dynamics model of the plant described in Eq. (43) for the same input, it can be viewed as a controller, called network controller. A schematic diagram of the network controller is shown in Fig. 5. Before a network controller is used, however, one must perform sufficient training on the network. In the following subsection, a training procedure for the network is presented.

3.2. Off-line and on-line combined training procedure

In the NFIN, we adopt both the off-line and on-line training schemes to train a network controller. For the off-line training, the general inverse-modeling learning scheme proposed by Psaltis et al. [20] is used. When we perform the off-line training, we have to decide the training patterns in advance. We always hope that the training patterns are sufficient and proper; however, there are no procedures or rules suitable to all cases. In some approaches [5,6], the training patterns are obtained by injecting specified input signals to the plant. In practical applications, such a technique is rather impractical, because we normally do not know what plant input u will cause the reference output y_{ref}. Therefore, in our scheme, the training patterns are obtained by probing the plant input using random signals. That is, a sequence of random input signals $u_{\mathrm{rd}}(k)$ under the magnitude limits of the plant input is injected directly to the plant, and then an open-loop input–output characteristic of the plant can be obtained. According to the input–output characteristic of the plant, proper

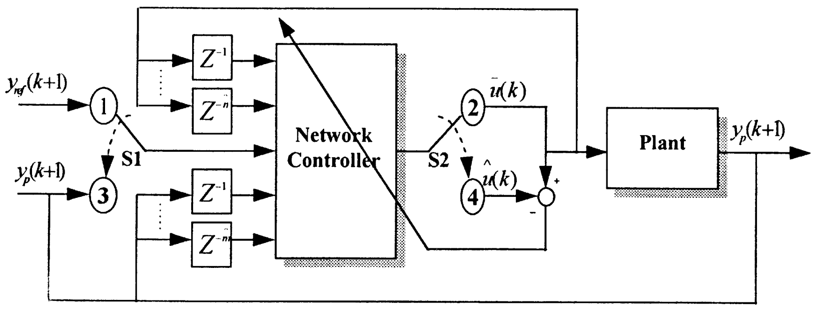

Fig. 6. Conventional on-line training scheme.

training patterns are selected to cover the entire reference output space. Using the collected training patterns with the values of the selected input variables as the input pattern and the corresponding control signal $u_{\mathrm{rd}}(k)$ as the target pattern, the network can be updated supervisedly to minimize an error function E defined by

$$E = \sum_{k=1}^{k_n} \frac{1}{2}[u_{\mathrm{rd}}(k) - \hat{u}(k)]^2, \tag{47}$$

where k_n is the number of training patterns.

After sufficient off-line training, the trained network is configured as a network controller to the plant as shown in Fig. 5. Generally speaking, such controller realizes fair but not perfect control. To achieve perfect control performance, on-line training is usually required. To do this, when the trained network is used to control the plant, the training patterns near the specific operating points are further gathered to train the network to suit the current control environment. In addition, in order to adapt to parameter variations in the plant and changes in the environment, the on-line training is also necessary. Hence, in our training procedure, we also include the on-line training scheme.

For the on-line training, a conventional on-line training scheme is used. Fig. 6 is a block diagram for the conventional on-line training scheme. In executing this scheme, we follow two phases, application phase and training phase. In the application phase, the switch S1 and S2 are connected to node 1 and node 2, respectively, to form a control loop. In this loop, the control signal $\bar{u}(k)$ is generated according to the input vector $I'(k) = [y_{\mathrm{ref}}(k+1), y_p(k), \ldots, y_p(k-\hat{m}+1), u(k-1), \ldots, u(k-\hat{n})]^{\mathrm{T}}$ (see Eq. (46)). In the training phase, the switches S1 and S2 are connected to nodes 3 and 4, respectively, to form a training loop. In this loop, we can define a training pattern with input vector $I(k) = [y_p(k+1), y_p(k), \ldots, y_p(k-\hat{m}+1), u(k-1), \ldots, u(k-\hat{n})]^{\mathrm{T}}$ (see Eq. (45)) and desired output $\bar{u}(k)$, where the input vector of network controller is the same as that used in the off-line training scheme. With this training pattern, the network controller can be trained supervisedly to minimize the error function $E(k+1)$ defined by

$$E(k+1) = \tfrac{1}{2}[\bar{u}(k) - \hat{u}(k)]^2, \tag{48}$$

where $\hat{u}(k)$ is the actual output of the network controller when it receives the input vector $I(k)$ in the training phase.

In the next section, the off-line and on-line combined training procedure will be used to train the BPNN and the proposed NFIN to control a water bath temperature control system.

4. NFIN for water bath temperature control

4.1. Problem statement

To see whether the proposed NFIN can achieve good performance and overcome the disadvantages of the BPNN, we compare it with the BPNN under the same aforementioned training procedure on a water bath temperature control system. Consider a discrete-time SISO temperature control system

$$y_p(k+1) = A(T_s)y_p(k) + \frac{B(T_s)}{1+e^{1/2y_p(k)-\gamma}}u(k) + [1-A(T_s)]y_0, \tag{49}$$

where

$$A(T_s) = e^{-aT_s}, \tag{50}$$

$$B(T_s) = \frac{b}{a}(1-e^{-aT_s}). \tag{51}$$

The above equation models a real water bath temperature control system given in [25]. The parameters are set as $a = 1.00151e^{-4}$, $b = 8.67973e^{-3}$, $\gamma = 40.0$ and $y_0 = 25.0°C$. The plant input $u(k)$ is limited between 0 and 5 V, and the sampling period is $T_s = 30$ s. The task is to control the water bath system to follow three set-points:

$$y_{\text{ref}}(k) = \begin{cases} 35°\text{C} & \text{for } k \leqslant 40, \\ 55°\text{C} & \text{for } 40 < k \leqslant 80, \\ 75°\text{C} & \text{for } 80 \leqslant k \leqslant 120. \end{cases}$$

4.2. Control performance

In implementing the off-line training scheme, a sequence of random input signals $u_{\text{rd}}(k)$ limited between 0 and 5 is injected directly to the water bath system described in Eq. (49). Then an open-loop input–output characteristic of the system is obtained as shown in Fig. 7. It is observed that the system behaves linearly up to about 70.0°C and then becomes nonlinear and saturates at about 80.0°C. From the input–output characteristic of the water bath system, 90 training patterns are selected to cover the entire reference output space. Comparing Eqs. (41) and (49), it is clear that we have $m = 1$ and $n = 0$ in Eq. (41) now. Hence, the input vector of the network controller can be decided as $\hat{m} = 1$ and $\hat{n} = 0$ in Eq. (45) to get a perfect matching.

For the BPNN, a four-layer feedforward network with two hidden layers is used. The hidden and output nodes have hyperbolic tangent sigmoid activation functions. In general, the choice of the number of hidden nodes is a fundamental question often raised in the application of BPNN and is usually decided by trials and errors. Therefore, to find a suitable number of hidden nodes, three networks: BPNN(5, 5), BPNN(10, 10) and BPNN(15, 15) are chosen, where the notation BPNN(a, b) denotes that the number of nodes in the first- and second-hidden-layer are a and b, respectively. To increase the convergence speed, a modified form of the generalized delta rule [17] is used. According to this form, the weights from the second-hidden-layer to the output-layer connections are updated by

$$\Delta w_{iq}(k+1) = \eta\delta_{oi}Z_q + \alpha\Delta w_{iq}(k) + \beta\Delta w_{iq}(k-1), \tag{52}$$

where

$$\delta_{oi} = ea'(net_i), \tag{53}$$

and Z_q is the output of the qth node in the second-hidden-layer. The parameter e is the difference between the target output and the actual output of the BPNN, η is the learning rate, and α and β are the momentum

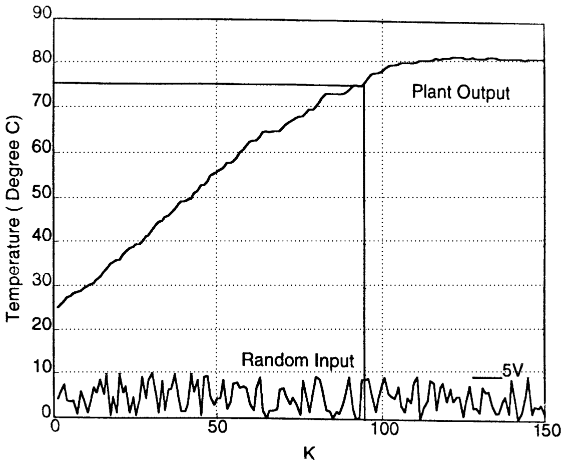

Fig. 7. Input–output characteristic of the water bath temperature training using control system

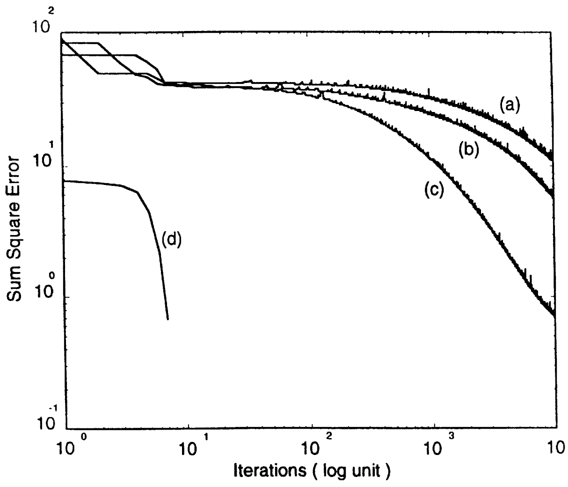

Fig. 8. Convergence curves of the control system through off-line (a) BPNN(5, 5), (b) BPNN(10, 10), (c) BPNN(15, 15) and (d) NFIN.

and acceleration coefficients, respectively. The value δ_{oi} is the error signal and its double subscript indicates the ith node in the output layer. The value net_i is the net input to node i in the output layer, and $a'(net_i) = \partial a(net_i)/\partial net_i$.

The weights from the first-hidden-layer to the second-hidden-layer connections are updated by

$$\Delta s_{qp}(k+1) = \eta\delta_{hq}L_p + \alpha\Delta s_{qp}(k) + \beta\Delta s_{qp}(k-1), \tag{54}$$

where

$$\delta_{hq} = a'(net_q)\sum_i \delta_{oi}w_{iq}, \tag{55}$$

and L_p is the output of the pth node in the first hidden layer. The value δ_{hq} is the error signal of node q in the second hidden layer. The value net_q is the net input to node q in the second hidden layer, and $a'(net_q) = \partial a(net_q)/\partial net_q$.

The weights from the input-layer to the first-hidden-layer connections are updated by

$$\Delta v_{pj}(k+1) = \eta\delta_{hp}X_j + \alpha\Delta v_{pj}(k) + \beta\Delta v_{pj}(k-1), \tag{56}$$

where

$$\delta_{hp} = a'(net_p)\sum_q \delta_{hq}s_{qp}, \tag{57}$$

and X_j is the output of the jth node in the input layer. The value δ_{hp} is the error signal of node p in the first hidden layer. The value net_p is the net input to node p in the first hidden layer, and $a'(net_p) = \partial a(net_p)/\partial net_p$. The learning rate, momentum and acceleration coefficients are set to 0.01, 0.9 and -0.1, respectively. The weights are initialized at small random values.

For the NFIN, the learning parameters are set as $\eta = 0.005$, $\beta = 0.7$, $\bar{F}_{\text{in}} = 0.1$, $\bar{F}_{\text{out}} = 0.7$, $\alpha = 0.4$ and $\lambda = 0.9$. Using the selected 90 training patterns with input vector $[y_p(k+1), y_p(k)]^{\mathrm{T}}$ and target pattern

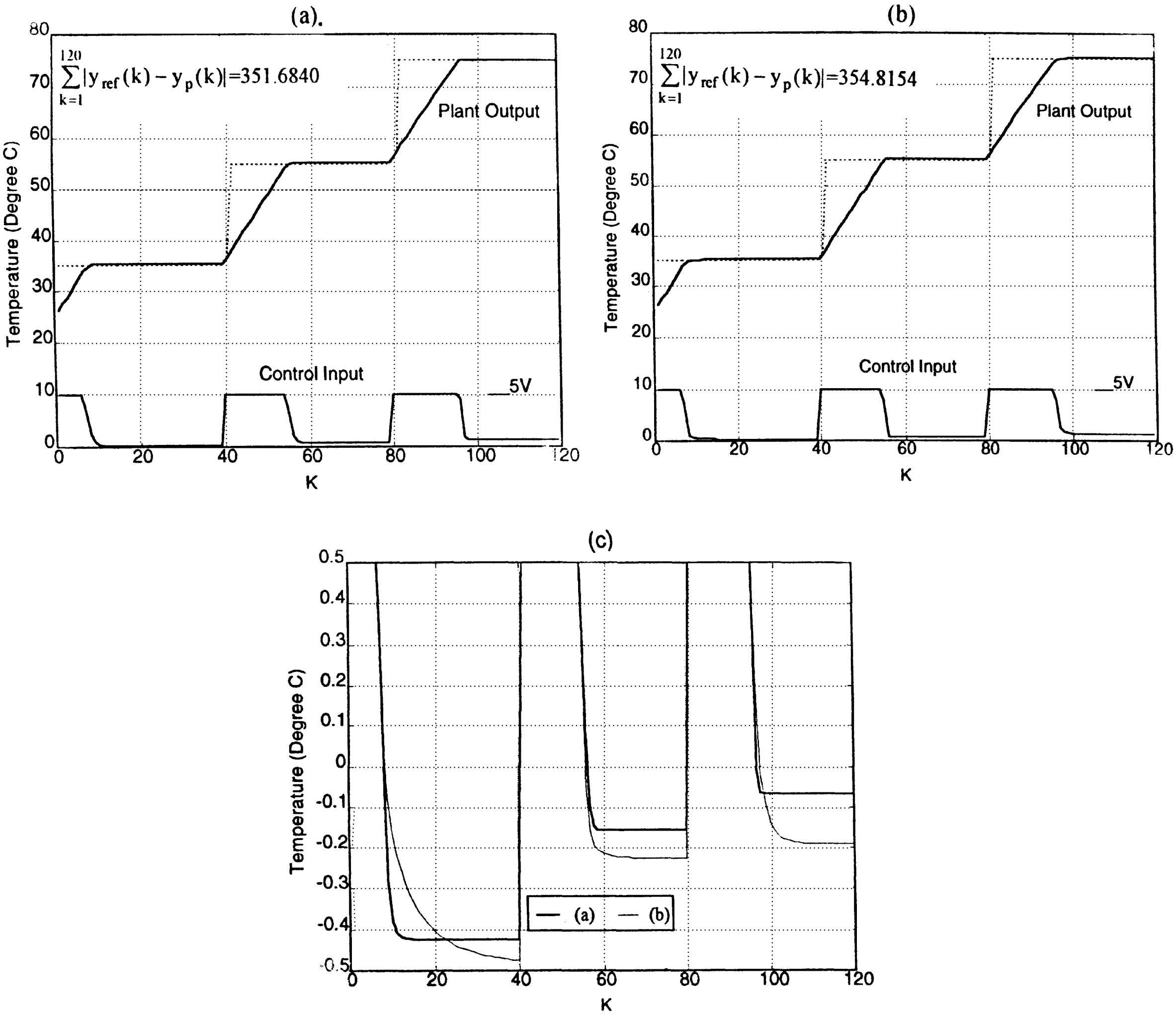

Fig. 9. Regulation performance of the control system after off-line training using (a) NFIN and (b) BPNN(15, 15). (c) The corresponding errors of (a) and (b).

$u_{\mathrm{rd}}(k)$, each of the NFIN and the three BPNNs described above is trained to minimize the error function E defined as

$$E=\sum_{k=1}^{90}\frac{1}{2}[u_{\mathrm{rd}}(k)-\hat{u}(k)]^2. \tag{58}$$

In the off-line training, the convergence curves of the NFIN and the three BPNNs are shown in Fig. 8. The curves show the sum square error per iteration as a function of the number of iterations. For the three BPNNs, we find that the BPNN(15, 15) shows the highest convergence speed of all. However, its convergence speed is still not satisfied. After 10 000 iterations, its error only reach about 0.7031. For the NFIN, as expected, its convergence speed is higher than that of the BPNN(15, 15). We find that its error reaches a relatively small value at about 0.6731 after only 5 iterations. After the off-line learning, either of the NFIN

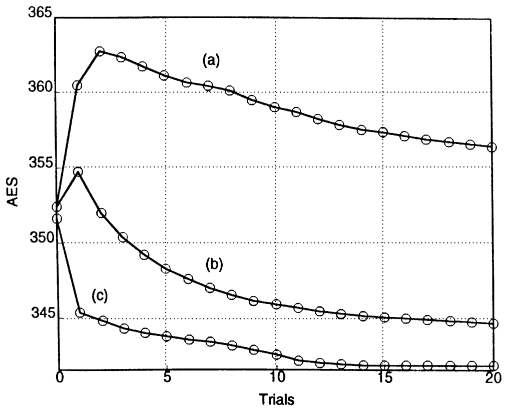

Fig. 10. Regulation performance of each trial of the control system through on-line training using (a) BPNN(15, 15) based on the conventional on-line training scheme, (b) BPNN(15, 15) based on the new on-line training scheme, and (c) NFIN based on the conventional on-line training scheme.

and the BPNN(15, 15) is configured as a direct feedforward controller to the water bath system. The regulation performance of these two network controllers and the corresponding errors are shown in Figs. 9(a)–(c). The curves in Fig. 9(c) shows the error between the reference output and the actual output of the control system. We find that the NFIN controller achieves better performance than the BPNN(15, 15) controller. However, their errors are still too large, meaning that off-line training along is not enough and on-line training is necessary to achieve high accuracy.

In the on-line training, the NFIN and BPNN(15, 15) controllers are trained by the conventional on-line training scheme shown in Fig. 6. Moreover, the new on-line training method that performs multiple updating operations during each sampling period [25] is also applied to the BPNN(15, 15) for comparison. In the new on-line training method, we choose additional 20 adjacent training patterns per sampling period. To test their regulation performance in each trial, a performance index, sum of absolute error (SAE), is defined by

$$SAE = \sum_k |y_{\mathrm{ref}}(k) - y_p(k)|, \tag{59}$$

where $y_{\mathrm{ref}}(k)$ and $y_p(k)$ are the reference output and the actual output of control system, respectively. The performance index SAE is calculated for k ranging from 1 to 120, which is called a trial.

After 20 trials of on-line training, the regulation performance of each trial of the aforementioned three cases is shown in Fig. 10, and the final regulation performance and the corresponding errors in the 20th trial of the on-line training are shown in Figs. 11(a)–(d). The curves in Fig. 10 show the sum of absolute error per trial as a function of the number of trials. For the BPNN(15, 15) based on the conventional on-line training scheme, as expected, its global learning ability leads to seriously overtuned phenomenon. From the corresponding errors in Fig. 11(d), it is found that it only performs good regulation at the upper set-point but deteriorate at others. Furthermore, we find its performance index value after 20 trials of on-line training is larger than its initial. For the BPNN(15, 15) based on the new on-line training scheme, although such method increases the on-line convergence speed and improves control performance as expected, the overtuned phenomenon still exists. From the corresponding errors in Fig. 11(d), it is also observed that it only improves the performance at the upper and middle set-points, but not at the lower set-points. Furthermore, the additional training of the adjacent patterns increases the computational load per sampling period. The NFIN based on the conventional on-line training scheme shows the highest on-line convergence speed and, owing to the local

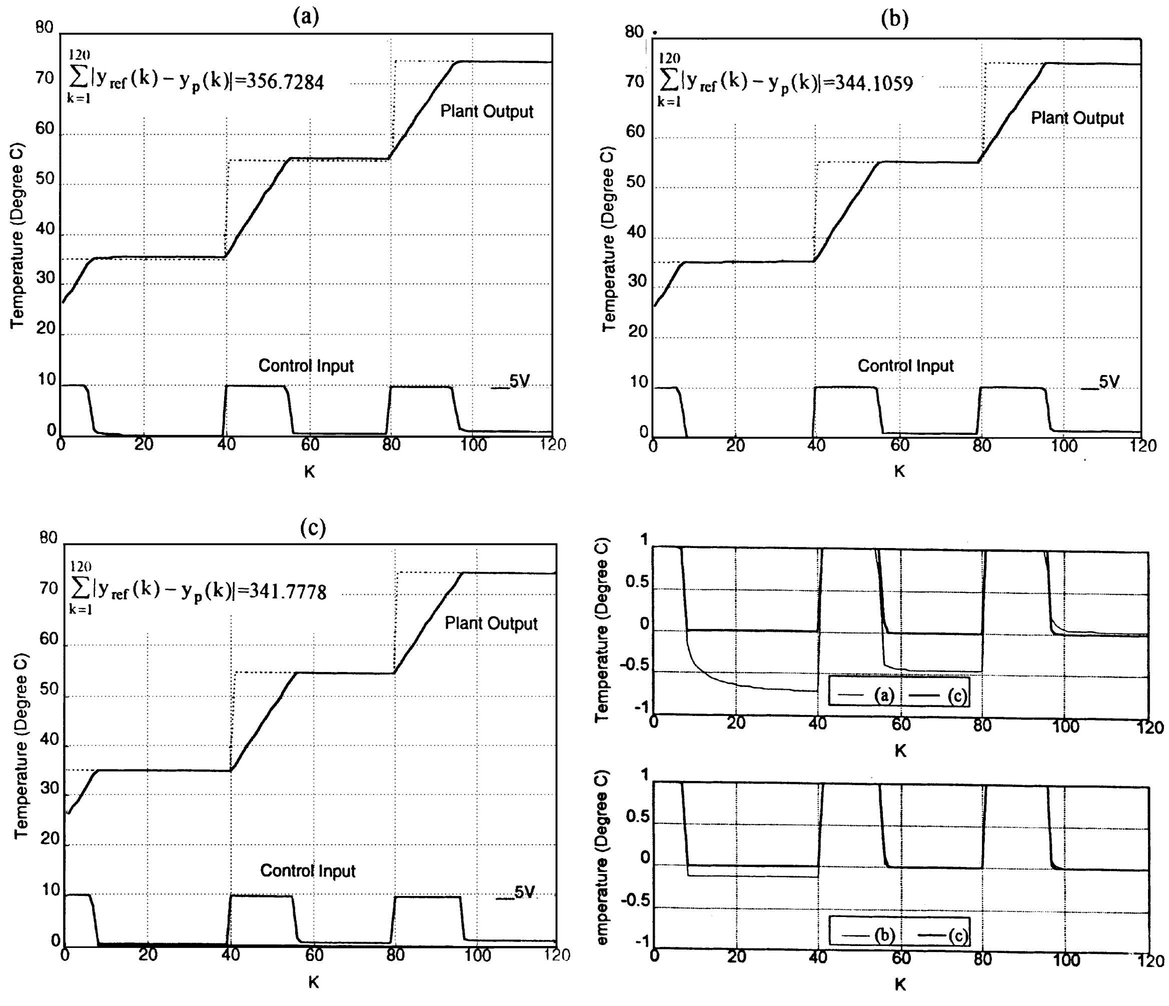

Fig. 11. Regulation performance of the control system in the 20th trial of on-line training using (a) BPNN(15, 15) based on the conventional on-line training scheme, (b) BPNN(15, 15) based on the new on-line training scheme, and (c) NFIN based on the conventional on-line training scheme. (d) The corresponding errors of (a), (b) and (c).

tuning property of fuzzy rule-based systems, there is no overtuned phenomenon after 20 trials of on-line training. According to the corresponding errors in Fig. 11(d), the NFIN shows good regulation ability at all set-points and little errors exist. The NFIN also shows better regulation performance after 4 trials of on-line training than the two other cases after 20 trials and the best performance is achieved after about 13 trials of on-line training. According to the performance index, among the three cases, the NFIN shows the best regulation control performance for the overall process. Fig. 12(a) shows the final assignment of fuzzy rules of the NFIN after 20-trials of on-line training in the $[y(k), y(k+1)]$ plain. The number of generated rules is 7, and the numbers of fuzzy sets on the $y(k)$ and $y(k+1)$ dimensions are 4 and 4, respectively, as shown in Fig. 12(b). In total, the number of network structure parameters is 31, but that of the BPNN(15, 15) is 270.

From the above results, we verify that the NFIN can achieve good control performance and overcome the disadvantages of the BPNN. Not only can it reduce the long off-line and on-line training time and avoid the overtuned phenomenon in the on-line training, but it also shows good regulation control capability on the

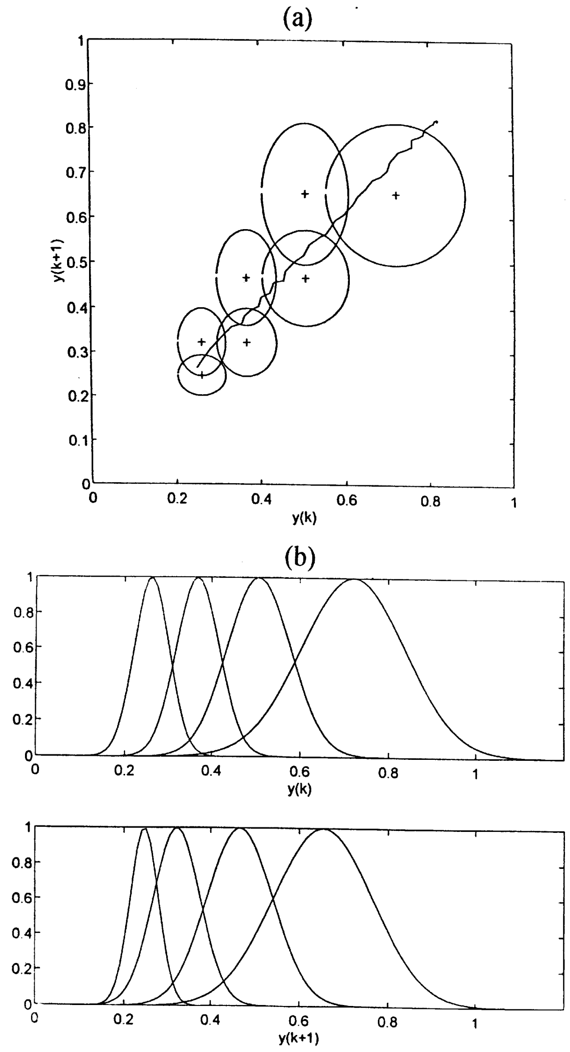

Fig. 12. Final structure of the NFIN after 20 trials of on-line training. (a) The final assignment of fuzzy rules in the $[y(k), y(k+1)]$ plain. (b) The corresponding membership functions on the $y(k)$ and $y(k+1)$ dimensions.

water bath system. Moreover, in comparing the number of network structure parameters, the NFIN also use less parameters than the BPNN.

5. Conclusion

Based on the studies of a water bath temperature control system, a summary of comparisons between the two networks, NFIN and BPNN(15, 15), is shown in Table 1. In comparing the number of network structure parameters, there are 270 structure parameters to be tuned for BPNN(15, 15), but only 31 structure parameters to be tuned for the NFIN. As to the convergence speed of the two networks, we have found that the NFIN not only has higher off-line convergence speed than the BPNN(15, 15), but the NFIN based on the conventional

Table 1
Summary of comparisons among the two network controllers on the water bath temperature control system

Criteria	NFIN	BPNN(15, 15)	BPNN(15, 15) (Using new on-line training scheme [16])
Network structure parameters	Light	Heavy	
Convergence speed (off-line)	Fast	Slow	
Regulation performance (off-line) $\mathrm{SAE} = \Sigma_{k=1}^{120} \lvert y_{\mathrm{ref}}(k) - y_p(k) \rvert$	SAE = 351.6840	SAE = 354.8154	
Convergence speed (on-line)	Fast	Slow	Medium
The final regulation performance $\mathrm{SAE} = \Sigma_{k=1}^{120} \lvert y_{\mathrm{ref}}(k) - y_p(k) \rvert$	SAE = 341.7778	SAE = 356.7284	SAE = 344.1059
Computational load (on-line)	Light	Light	Heavy

on-line training scheme also has a higher on-line convergence speed than the BPNN(15, 15) based on the conventional or the new on-line training scheme proposed in [25]. For the regulation performance after off-line training, the NFIN controller shows a better regulation capability than the BPNN(15, 15) controller. Moreover, the final regulation performance of the NFIN controller after 20 trials of the conventional on-line training also shows the best regulation control capability of all cases. According to these comparisons, we find the NFIN not only overcome the disadvantages of the BPNN but also owns good control capability in the water bath temperature control system. Hence, it is promising to apply the NFIN controller to other real cases in industries in the near future.

References

[1] K.J. Astrom, B. Wittenmark, Adaptive Control, Addison-Wesley, Reading, MA, 1989.

[2] H.R. Berenji, P. Khedkar, Learning and tuning fuzzy logic controllers through reinforcements, IEEE Trans. Neural Networks 3 (5) (1992) 724–740.

[3] M. Jordan, D.E. Rumelhart, Internal world models and supervised learning, in: Machine Learning, Proc. 8th Internat. Workshop, 1991.

[4] C.L. Karr, E.J. Gentry, Fuzzy control of pH using genetic algorithms, IEEE Trans. Fuzzy System 1 (1993) 46–53.

[5] M. Khalid, S. Omatu, A neural network controller for a temperature control system, IEEE Control System Mag. 12 (3) (1992) 58–64.

[6] M. Khalid, S. Omatu, R. Yusof, MIMO furnace control with neural networks, IEEE Trans. Control System Technol. 1 (4) (1993) 238–245.

[7] B. Kosko, Neural Networks and Fuzzy Systems, Prentice-Hall, Englewood Cliffs, NJ, 1992.

[8] C.C. Lee, Fuzzy logic in control systems: fuzzy logic controllers – Parts I, II, IEEE Trans. System Man Cybernet. 20 (1990) 404–435.

[9] C. Ling, T. Edgar, A new fuzzy gain scheduling algorithm for process control, in: Proc. ACC'92, Chicago, IL, 1992, pp. 2284–2290.

[10] C.T. Lin, C.S.G. Lee, Neural-network-based fuzzy logic control and decision system, IEEE Trans. Comput. 40 (12) (1991) 1320–1336.

[11] C.T. Lin, C.S.G. Lee, Neural Fuzzy Systems: A Neural-Fuzzy Synergism to Intelligent Systems (with disk), Prentice-Hall, Englewood Cliffs, NJ, 1996.

[12] C.T. Lin, Neural Fuzzy Control Systems with Structure and Parameter Learning, World Scientific, Singapore, 1994.

[13] C.J. Lin, C.T. Lin, Reinforcement learning for ART-based fuzzy adaptive learning control networks, IEEE Trans. Neural Networks 7 (3) (1996).

[14] C.T. Lin, A neural fuzzy control system with structure and parameter learning, Fuzzy Sets and Systems 70 (1995) 183–212.

[15] W.T. Miller III, R.S. Sutton, P.J. Werbos (Ed.), Neural Networks for Control, MIT Press, Cambridge, MA, 1990.

[16] K.S. Narendra, R. Ortega, P. Dorato (Eds.), Advances in Adaptive Control, IEEE Press, New York, 1991.

[17] S. Nagata, M. Sekiguchi, K. Asakawa, Mobile robot control by a structured hierarchical neural network, IEEE Control System Mag. 10 (3) (1990) 69–76.

[18] B. Noble, J.W. Daniel, Applied Linear Algebra, 3rd ed., Prentice-Hall, Englewood Cliffs, NJ, 1988.
[19] K. Ogata, Discrete-Time Control Systems, Prentice-Hall, Englewood Cliffs, NJ, p. 202.
[20] D. Psaltis, A. Sideris, A. Yamamura, A multilayered neural network controller, IEEE Control System Mag. 10 (3) (1989) 44–48.
[21] D.E. Rumelhart et al., Learning internal representation by error propagation, in Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. I, MIT Press, Cambridge, MA, 1986.
[22] E.H. Ruspini, Recent development in fuzzy clustering, Fuzzy Set and Possibility Theory, North-Holland, New York, 1982, pp. 113–147.
[23] M. Sugeno, K. Tanaka, Successive identification of a fuzzy model and its applications to prediction of a complex system, Fuzzy Sets and Systems 42 (3) (1991) 315–334.
[24] C.T. Sun, J.S. Jang, A neuro-fuzzy classifier and its applications, Proc. IEEE Internat. Conf. Fuzzy Systems, San Francisco, CA, vol. I, 1993, pp. 94–98.
[25] J. Tanomaru, S. Omatu, Process control by on-line trained neural controllers, IEEE Trans. Ind. Electron. 39 (6) (1992) 511–521.
[26] K. Tanaka, M. Sano, H. Watanabe, Modeling and control of carbon monoxide concentration using a neuro-fuzzy technique, IEEE Trans. Fuzzy Systems 3 (3) (1995) 271–279.
[27] T. Takagi, M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, IEEE Trans. System Man Cybernet. 15 (1) (1985) 116–132.
[28] L.X. Wang, J.M. Mendel, Fuzzy basis functions, universal approximation, and orthogonal least-squares learning, IEEE Trans. Neural Networks 3 (5) (1992) 807–814.
[29] L. Wang, R. Langari, Building Sugeno-type models using fuzzy discretization and orthogonal parameter estimation techniques, IEEE Trans. Fuzzy Systems 3 (4) (1995) 454–458.
[30] T. Yabuta, T. Yamada, Learning control using neural networks, in Proc. 1991 IEEE Internat. Conf. Robotics and Automation, Sacramento, CA, April 1991.
[31] L.A. Zazeh, Fuzzy sets, Inform. and Control 8 (1965) 338–352.
[32] L.A. Zazeh, Outline of a new approach to the analysis of complex systems and decision processes, IEEE Trans. System Man Cybernet. 3 (1973) 28–44.