[24] M. Sakawa and H. Yano, "An interactive method for multiobjective nonlinear programming problems with fuzzy parameters," in *Cybernetics and Systems '86*, R. Trappl, Ed. Dordrecht, The Netherlands: Reidel, 1986, pp. 607–614.
[25] M. Sakawa and H. Yano, "Interactive decision making for multiobjective linear programming problems with fuzzy parameters," in *Large-Scale Modeling and Interactive Decision Analysis*, G. Fandel, M. Grauer, A. Kurzhanski, and A. P. Wierzbicki, Eds. New York: Springer-Verlag, 1986, pp. 88–96.
[26] ——, "An interactive satisficing method for multiobjective nonlinear programming problems with fuzzy parameters," in *Optimization Models Using Fuzzy Sets and Possibility Theory*, J. Kacprzyk and S. A. Orlovski, Eds. Dordrecht, The Netherlands: Reidel, 1987, pp. 258–271.
[27] ——, "Interactive decision making for multiobjective nonlinear programming problems with fuzzy parameters," *Fuzzy Sets Syst.*, vol. 29, no. 3, pp. 315–326, 1989.
[28] ——, "An interactive fuzzy satisficing method for multiobjective nonlinear programming problems with fuzzy parameters," *Fuzzy Sets Syst.*, vol. 30, no. 3, pp. 221–238, 1989.
[29] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Cambridge, MA: MIT Press, 1992.
[30] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 2nd extended ed. New York: Springer-Verlag, 1994.
[31] ——, *Genetic Algorithms + Data Structures = Evolution Programs*, Third, revised and extended ed. New York: Springer-Verlag, 1996.

Dynamic Optimal Learning Rates of a Certain Class of Fuzzy Neural Networks and its Applications with Genetic Algorithm

Chi-Hsu Wang, Han-Leih Liu, and Chin-Teng Lin

***Abstract*—The stability analysis of the learning rate for a two-layer neural network (NN) is discussed first by minimizing the total squared error between the actual and desired outputs for a set of training vectors. The stable and optimal learning rate, in the sense of maximum error reduction, for each iteration in the training (back propagation) process can therefore be found for this two-layer NN. It has also been proven in this paper that the dynamic stable learning rate for this two-layer NN must be greater than zero. Thus it is guaranteed that the maximum error reduction can be achieved by choosing the optimal learning rate for the next training iteration. A dynamic fuzzy neural network (FNN) that consists of the fuzzy linguistic process as the premise part and the two-layer NN as the consequence part is then illustrated as an immediate application of our approach. Each part of this dynamic FNN has its own learning rate for training purpose. A genetic algorithm is designed to allow a more efficient tuning process of the two learning rates of the FNN. The objective of the genetic algorithm is to reduce the searching time by searching for only one learning rate, which is the learning rate of the premise part, in the FNN. The dynamic optimal learning rates of the two-layer NN can be found directly using our innovative approach. Several examples are fully illustrated and excellent results are obtained for the model car backing up problem and the identification of nonlinear first order and second order systems.**

***Index Terms*—Backpropogation, fuzzy neural networks, genetic algorithm, learning rate.**

Manuscript received November 26, 1999; revised September 29, 2000. This paper was recommended by Associate Editor T. Sudkamp.

C.-H. Wang and H.-L. Liu are with the School of Microelectronic Engineering, Griffith University, Queensland, Australia (e-mail: c.wang@me.gu.edu.au).

C.-T. Lin is with the Department of Electrical and Control Engineering, National Chiao-Tung University, Hsinchu, Taiwan, R.O.C.

Publisher Item Identifier S 1083-4419(01)02511-0.

I. Introduction

During the past decade, fuzzy neural networks (FNNs) have found a variety of applications in various fields [1]–[3]. Most notably, a FNN system has been applied to control nonlinear, ill-defined systems [4]. These systems used the back-propagation (BP) algorithm to tune the parameters of fuzzy sets and the weights of neural network (NN). Basically the BP algorithm is of descent type, which attempts to minimize the difference (or error) between the desired and actual outputs in an iterative manner. For each iteration, the parameters and weights are adjusted by the algorithm so as to reduce the error along a descent direction. In doing so, values, which are called learning rates, should be properly set in the BP algorithm. Authors in [5] proposed dynamic optimization of the learning rate using derivative information. In [5], it was shown that the relatively large or small learning rates may affect the progress of BP algorithm and even may lead to failure of the learning process. However, the analysis of stable learning rates was not discussed in [5]. Recently genetic algorithms (GAs) [6]–[10] have emerged as a popular family of methods for global optimization. GAs perform a search by evolving a population of potential solutions through the use of its operators. The authors in [9] proposed GAs to tune the parameters of the Gaussian membership functions. Although reasonable results have been obtained in [9], the analysis of stable learning rate was also not discussed at all.

In order to perform the stability analysis of the learning rate [11] in FNN, we start from the stability analysis of the learning rate for a two-layer neural network (NN) by minimizing the total squared error between the actual and desired outputs for a set of training vectors. The stable and optimal learning rate, in the sense of maximum error reduction, for each iteration during the back propagation process can be found for this two-layer NN. It is proven in this paper that the stable learning rate for this two-layer NN must be greater than zero. Following Theorem 1, it is guaranteed that the maximum error reduction can be achieved by choosing the optimal learning rate for the next training iteration. We then propose a dynamic fuzzy neural network that consists of the fuzzy linguistic process as the premise part and the two-layer NN as the consequence part. Each part has its own learning rate to be decided. The stable and optimal learning rate of the two-layer NN in the proposed FNN can also be found directly by our method, provided that the output of the premise part (or the input of the consequent part) remains the same during the training process of the consequent part. In order to find the best learning rate for the premise part, a new genetic search algorithm is proposed together with the stable and optimal learning rate in the consequent part. The major advantage of this new genetic algorithm is to reduce the searching time by searching only one learning rate, which is the learning rate of the premise part, in the dynamic FNN. In comparison with the searching process proposed in [9], our proposed GA has the benefit of reducing the searching complexity dramatically.

It is well known that backing up control of a truck is a very difficult exercise for all but the most skilled truck drivers since its dynamics are nonlinear and unstable. Based on our new methodology, a FNN controller for backing up a truck is successfully designed. Using the optimal learning rates, the trained FNN system can indeed let the truck reach its loading zone successfully. Also the nonlinear system identifications of first and second order systems are fully illustrated with excellent results. For the applicability of other FNN models, such as Horikawa *et al.* [12], we will consider them as future research topics.

1083–4419/01$10.00 © 2001 IEEE

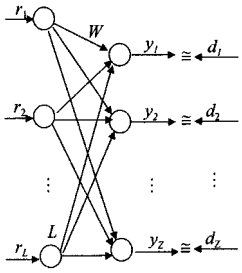


Fig. 1. Two-layer NN.

II. Dynamic Optimal Learning Rates for a Two-Layer NN

Consider the following simple two-layer NN in Fig. 1, which will form the consequent part of the FNN adopted in this paper where

$$\underline{r} = [r_1 \quad r_2 \quad \cdots \quad r_L]^T \in \mathcal{R}^L$$
the training data vector (1)

$$W = [\underline{w}_1 \quad \underline{w}_2 \quad \cdots \quad \underline{w}_Z] \in \mathcal{R}^{L\times Z}$$
the weighting matrix (2)

$$\underline{w}_i = [w_i^1 \quad w_i^2 \quad \cdots \quad w_i^L]^T \in \mathcal{R}^L$$
the ith weighting vector (3)

$$\underline{y} = [y_1 \quad y_2 \quad \cdots \quad y_Z]^T \in \mathcal{R}^Z$$
the actual output vector (4)

$$\underline{d} = [d_1 \quad d_2 \quad \cdots \quad d_Z]^T \in \mathcal{R}^Z$$
the desired output vector (5)

and "T" denotes matrix transpose.

Given a set of training vectors, which forms the training matrix R in (7), it is desired to use the back propagation technique to train the above NN so that the actual outputs converge to the desired outputs. The actual output y_z is defined as

$$y_z = \sum_{l=1}^{L} r_l w_z^l = \underline{r}^T \underline{w}_z. \tag{6}$$

Given P training vectors, there should be P desired output vectors. In matrix notations, we let

$$R = [\underline{r}_1 \quad \underline{r}_2 \quad \cdots \quad \underline{r}_P] \in \mathcal{R}^{L\times P}$$
the input training matrix (7)

$$Y = [\underline{y}_1 \quad \underline{y}_2 \quad \cdots \quad \underline{y}_P]^T \in \mathcal{R}^{P\times Z}$$
the actual output matrix (8)

$$D = [\underline{d}_1 \quad \underline{d}_2 \quad \cdots \quad \underline{d}_P]^T \in \mathcal{R}^{P\times Z}$$
the desired output matrix. (9)

The actual output matrix Y (8) can be shown as

$$Y = R^T W. \tag{10}$$

It is desired to update (or train) the weighting matrix W so that the actual output y_z will converge to a desired output d_z. To do so, we define the total squared error J as follows:

$$J = \frac{1}{2P\cdot Z}\sum_{p=1}^{P}\sum_{z=1}^{Z}(y_z^p - d_z^p)^2. \tag{11}$$

The above J can also be reorganized using matrix notation. To do so, we define error function E as

$$E = Y - D = R^T W - D. \tag{12}$$

Then we have

$$J = \frac{1}{2P\cdot Z} Tr(EE^T). \tag{13}$$

Equation (13) actually considers all the P training vectors to yield the total squared error. Other approaches [4], [14], [15], only considered the squared error for a single training vector.

To update W, we apply the back propagation method as follows:

$$W_{t+1} = W_t - \beta_t \left.\frac{\partial J}{\partial W}\right|_t \tag{14}$$

where t denotes the tth iteration. Using chain rule, we get

$$W_{t+1} = W_t - \beta_t \frac{1}{P\cdot Z} RE. \tag{15}$$

After training, assuming zero error, we should have matrix form $D = R^T W$. It should be noted we assume that the learning rate for each iteration during the back propagation process is different, i.e., the learning rates are not fixed. In order to find the optimal learning rate for β_t, we have the following theorem.

Theorem 1: The optimal learning rate β_t defined in (15) can be found from the minimum of a quadratic polynomial $A\beta^2 + B\beta = 0$, where $A(>0)$ and $B(<0)$ can be obtained from the training vector $\underline{r}$, desired output vector $\underline{d}$ and the weighting matrix W.

Proof: First, we must find the stable range for β_t. To do so, we define the Lyapunov function as

$$V = J^2 \tag{16}$$

where J is defined in (13). The change of the Lyapunov function is $\Delta V = J_{t+1}^2 - J_t^2$. It is well known that if $\Delta V < 0$, the response of the system is guaranteed to be stable. For $\Delta V < 0$ we have

$$J_{t+1} - J_t < 0. \tag{17}$$

Here we consider all the P training vectors as $\{\underline{r}_i = [r_i^1 \quad r_i^2 \quad \cdots \quad r_i^L]^T | i = 1, \cdots, P\}$. From (15) (for $w_z^l(t+1)$) and the fact that the training vectors remain the same during the training process, i.e., $r_l^p(t+1) = r_l^p(t) = r_l^p$), we have $\boldsymbol{J_{t+1}}$ [from (13)] as follows:

$$\begin{aligned}
J_{t+1} &= (2PZ)^{-1} Tr(E_{t+1}E_{t+1}^T) \\
&= (2PZ)^{-1} Tr[(R^T W_{t+1} - D)(R^T W_{t+1} - D)^T] \\
&= (2PZ)^{-1} Tr\{[(R^T(W_t - \beta_t(PZ)^{-1}RE_t) - D)] \\
&\quad \cdot [(R^T(W_t - \beta_t(PZ)^{-1}RE_t) - D)^T]\} \\
&= (2PZ)^{-1} Tr[(R^T W_t - \beta_t(PZ)^{-1}R^T RE_t - D) \\
&\quad \cdot (W_t^T R - \beta_t(PZ)^{-1}E_t^T R^T R - D^T)] \\
&= (2PZ)^{-1} Tr\{[(R^T W_t - D) - \beta_t(PZ)^{-1}R^T RE_t] \\
&\quad \cdot [(W_t^T R - D^T) - \beta_t(PZ)^{-1}E_t^T R^T R]\} \\
&= (2PZ)^{-1} Tr[(E_t - \beta_t(PZ)^{-1}R^T RE_t) \\
&\quad \cdot (E_t^T - \beta_t(PZ)^{-1}E_t^T R^T R)] \\
&= (2PZ)^{-1} Tr[E_t E_t^T - 2\beta_t(PZ)^{-1}R^T RE_t E_t^T \\
&\quad + \beta_t^2(PZ)^{-2}R^T RE_t E_t^T R^T R] \\
&= J_t + \beta_t(-PZ)^{-1} Tr[(PZ)^{-1}R^T RE_t E_t^T] \\
&\quad + \beta_t^2(2PZ)^{-1} Tr[(PZ)^{-2}R^T RE_t E_t^T R^T R] \\
&= J_t + \beta_t(-PZ)^{-1} Tr[(PZ)^{-1}E_t^T R^T RE_t] \\
&\quad + \beta_t^2(2PZ)^{-1} Tr[(PZ)^{-2}R^T RE_t E_t^T R^T R].
\end{aligned}$$

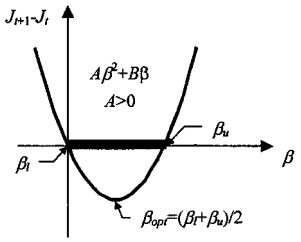


Fig. 2. Parabolic trajectory of $J_{t+1} - J_t$ (or $A\beta^2 + B\beta$) versus β.

Hence

$$\begin{aligned} J_{t+1} - J_t &= \beta_t(-PZ)^{-1}Tr[(PZ)^{-1}E_t^T R^T R E_t] \\ &\quad + \beta_t^2(2PZ)^{-1}Tr[(PZ)^{-2}R^T R E_t E_t^T R^T R] \\ &= A\beta_t^2 + B\beta_t \end{aligned} \tag{18}$$

where

$$\begin{aligned} A &= \tfrac{1}{2}(PZ)^{-3}Tr[R^T R E_t E_t^T R^T R] \\ &= \tfrac{1}{2}(PZ)^{-3}\sum_{p=1}^{P}\sum_{z=1}^{Z} \\ &\quad \cdot \left(\sum_{l=1}^{L} r_l^p(t)\sum_{i=1}^{P} r_l^i(t)(y_z^i - d_z^i)\right)^2 \end{aligned} \tag{19}$$

$$\begin{aligned} B &= -(PZ)^{-2}Tr[E_t^T R^T R E_t] \\ &= -(PZ)^{-2}\sum_{p=1}^{P}\sum_{z=1}^{Z} \\ &\quad \left((y_z^p - d_z^p)\sum_{l=1}^{L} r_l^p(t)\sum_{i=1}^{P} r_l^i(t)(y_z^i - d_z^i)\right). \end{aligned} \tag{20}$$

It is obvious that (19) and (20) contain quadratic matrices, therefore, the A should be greater than zero and B should be less than zero. Therefore we have

$$J_{t+1} - J_t = A\beta^2 + B\beta < 0.$$

Fig. 2 shows the parabolic trajectory of $A\beta^2 + B\beta$ versus β. In order to satisfy (17), we must have $A\beta^2 + B\beta < 0$. Since $A > 0$, it is obvious that the stable range of β is (β_l, β_u), where β_l and β_u are the two roots of $A\beta^2 + B\beta = 0$. From Fig. 2, we also know that the optimal $\beta(= \beta_{opt})$ is the median of β_l and β_u, i.e., when

$$\beta_{opt} = (\beta_u + \beta_l)/2 \tag{21}$$

$A\beta_{opt}^2 + B\beta_{opt}$ is at its minimum. This is due to the symmetrical property of the parabola in Fig. 2. The β_{opt} will not only guarantee the stability of the training process, but also has the fastest speed of convergence. **Q.E.D.**

By inspecting (19) and (20), it is obvious that the stable range of β is a function of $\underline{r}$, $\underline{d}$ and W. Theorem 2 shows that the stable learning rate should be positive in the two-layer NN with a set of fixed training vectors.

Theorem 2: For the two-layer NN defined in Fig. 1, the stable learning rate should be positive, i.e., $\beta > 0$.

Proof: From Theorem 1, we know that $A > 0$ and $B < 0$. Therefore $A\beta^2 + B\beta < 0$ implies that $B\beta < -A\beta^2 < 0$. Since $B < 0$, we have the end result of $\beta > 0$. **Q.E.D.**

Algorithm I shows the overall computational aspects for the back propagation training process of the above two layer NN.

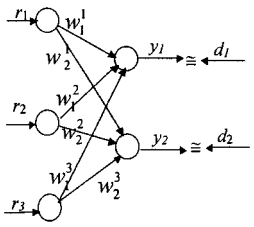


Fig. 3. Two-layer NN with three inputs and two outputs.

Algorithm I: Dynamic Optimal Learning Rates for a Two-Layer NN

Step1: Given the initial weighting matrix W_0, training matrix R and desired output matrix D, find the initial actual output matrix Y_0 (10) and optimal learning rate β_0 (Theorem 1).

Step2: Start the back propagation training process. Iteration count $t = 0$.

Step3: Find if the D and Y_t (10) are close enough or not? If Yes, GOTO Step 7.

Step4: Update the weighting matrix to yield W_{t+1} by (15).

Step5: Find the optimal learning rate β_{t+1} (Theorem 1) for the next iteration.

Step6: $t = t + 1$. GOTO Step 3.

Step7: End.

The following Example 1 illustrates the major concept in this section.

Example 1: Fig. 3 shows a two-layer NN with three inputs and two outputs.

Given input training matrix R, desired output matrix D (defined in (7) and (9)) as

$$\begin{aligned} R &= [\underline{r}_1 \quad \underline{r}_2 \quad \underline{r}_3 \quad \underline{r}_4] \\ &= \begin{bmatrix} -3.0852 & 1.0449 & 2.9027 & 5.0642 \\ -4.1030 & -4.3199 & 0.5842 & 1.4118 \\ -5.0811 & 6.31611 & -0.9816 & 1.2853 \end{bmatrix}_{L(=3)\times P(=4)} \end{aligned}$$

$$D = \begin{bmatrix} -0.9346 & -0.0882 \\ 1.0108 & 0.1857 \\ 0.0664 & -0.9783 \\ 0.4995 & -1.3264 \end{bmatrix}_{P(=4)\times Z(=2)}.$$

The initial weighting matrix W_i is chosen to be

$$W_i = \begin{bmatrix} -0.0531 & 0.1050 \\ -1.7333 & 1.3398 \\ -0.9498 & -1.2728 \end{bmatrix}_{L(=3)\times Z(=2)}.$$

The initial J is 28.1832. After 30 iterations, the stable range of learning rate β for each iteration can be found from (14)–(20) and are listed in Table I.

After finding the stable range of each iteration, we choose $0.5\beta_u$ to be the real learning rate for that iteration and perform the update of the weighting matrix W. Fig. 4 shows the trajectory of total squared error J. It is obvious that the values of total squared error decreased as expected.

The final weighting matrix W_f is

$$W_f = \begin{bmatrix} 0.0657 & -0.3169 \\ -0.0039 & 0.0852 \\ 0.1461 & 0.1396 \end{bmatrix}_{L(=3)\times Z(=2)}.$$

TABLE I
COEFFICIENT B, THE LEARNING RATE β, ITS STABLE RANGES, AND J

t	B	β_l	β_{opt}	β_u	J
1	-398.2670	0	0.1260	0.2520	3.0903
2	-34.3532	0	0.1240	0.2481	0.9596
3	-5.6185	0	0.1834	0.3668	0.4444
⋮	⋮	⋮	⋮	⋮	
28	-9.2024e-8	0	0.1422	0. 2844	1.29765e-4
29	-3.5148e-8	0	0.1806	0. 3612	1.29762e-4
30	-2.1644e-8	0	0.1422	0. 2844	1.29761e-4

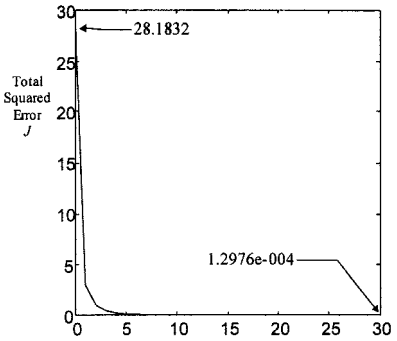


Fig. 4. Total squared error J via iteration t.

In the end, we have

$$D = \begin{bmatrix} -0.9346 & -0.0882 \\ 1.0108 & 0.1857 \\ 0.0664 & -0.9783 \\ 0.4995 & -1.3264 \end{bmatrix} \approx R^T \times W_f = \begin{bmatrix} -0.9295 & -0.0814 \\ 1.0083 & 0.1823 \\ 0.0452 & -1.0071 \\ 0.5153 & -1.3050 \end{bmatrix}.$$

III. FNN WITH DYNAMIC STABLE LEARNING RATE

The FNN in Fig. 5 was proposed in [4] for the control of a model car to follow a specified path, but without the stability analysis of learning rates. Here we adopt the identical structure as shown in [4] but replace the B-spline membership functions with Gaussain membership functions. Fig. 5 contains the premise part and consequent part. Each part has its own learning rate. The learning rate in the premise part is to fine-tune the Gaussian membership functions, whereas the learning rate in the consequent part is to adjust the weighting factors. Fig. 6 redraws the consequent part of Fig. 5, which clearly shows that the two-layer NN in Fig. 1 is the consequent part of Fig. 5. The stability analysis in Section II will be used to analyze the stability of the FNN and then a more efficient GA is devised in Section IV to tune this FNN.

The reasoning rule can be established by the following:

Rule l: If x_1 is F_1^l and $\cdots$ and x_N is F_N^l then y_1 is w_1^l and $\cdots$ and y_Z is w_Z^l

where $l = 1, 2, \cdots, L$, F_q^l's are fuzzy membership functions of the antecedent part, and $w_z^l \in W$ are neural network weights of the consequent part. The F_q^l's, whose functions are Gaussian functions, and μ_l are

$$F_q^l(x_q) = \exp\left(-\left(\frac{x_q - \eta_q}{\sigma_q}\right)^2\right) \tag{22}$$

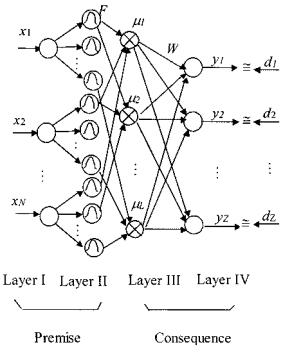


Fig. 5. Proposed FNN in this paper.

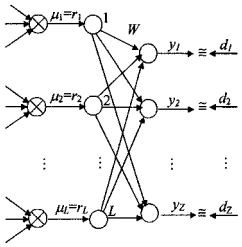


Fig. 6. Another look at the consequent part in Fig. 5.

$$r_l = \mu_l = \prod_{i=1}^{N} F_i^l(x_i) \tag{23}$$

where μ_l is the truth value of the premise of the lth rule. The output y_z of the fuzzy reasoning can be derived from the following equation:

$$\begin{aligned} y_z &= a/b \\ a &= \sum_{i=1}^{L} w_z^i \mu_i, \quad b = \sum_{i=1}^{L} \mu_i \qquad (z = 1, 2, \cdots, Z) \\ \underline{y} &= [y_1 \quad y_2 \quad \cdots \quad y_Z]^T. \end{aligned} \tag{24}$$

By adjusting the weighting factors and the parameters of the Gaussian functions of the neural network, the learning algorithm can be derived to minimize the total squared error J defined in (11). To update η_i^l, we use

$$\eta_i^l(t+1) = \eta_i^l(t) - \alpha_t \left.\frac{\partial J}{\partial \eta}\right|_t. \tag{25}$$

Using the chain rule, we get

$$\begin{aligned} \eta_i^l(t+1) = \eta_i^l(t) - \alpha_t \frac{1}{P \cdot Z} \sum_{p=1}^{P} \sum_{z=1}^{Z} (y_z^p - d_z^p) \\ \cdot \frac{w^l - y_z^p}{b} \mu^l \frac{(x_i^p - \eta_i^l(t))}{(\sigma_i^l(t))^2} \end{aligned} \tag{26}$$

where α_t is a current learning rate for tuning η. Again, using a similar method, we have the following for σ_i^l, w_z^l:

$$\begin{aligned} \sigma_i^l(t+1) = \sigma_i^l(t) - \alpha_t \frac{1}{P \cdot Z} \sum_{p=1}^{P} \sum_{z=1}^{Z} (y_z^p - d_z^p) \\ \cdot \frac{w^l - y_z^p}{b} \mu^l \frac{(x_i^p - \eta_i^l(t))^2}{(\sigma_i^l(t))^3}, \end{aligned} \tag{27}$$

$$w_z^l(t+1) = w_z^l(t) - \beta_t \frac{1}{P \cdot Z} \sum_{p=1}^{P} \frac{y_z^p - d_z^p}{b} \mu^l \tag{28}$$

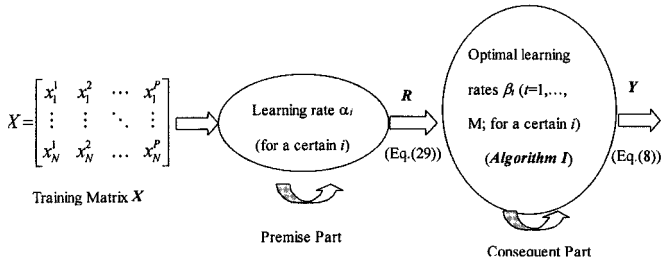


Fig. 7. Training process of the proposed FNN.

where β_t is a current learning rate for tuning w_z^l, b defined in (24). Hence the input matrix R of the consequent part, i.e., the two layer NN, becomes

$$R = \begin{bmatrix} \mu_1^1 \Big/ \sum_{i=1}^{L} \mu_i^1 & \mu_1^2 \Big/ \sum_{i=1}^{L} \mu_i^2 & \cdots & \mu_1^P \Big/ \sum_{i=1}^{L} \mu_i^P \\ \mu_2^1 \Big/ \sum_{i=1}^{L} \mu_i^1 & \mu_2^2 \Big/ \sum_{i=1}^{L} \mu_i^2 & \cdots & \mu_2^P \Big/ \sum_{i=1}^{L} \mu_i^P \\ \vdots & \vdots & \ddots & \vdots \\ \mu_L^1 \Big/ \sum_{i=1}^{L} \mu_i^1 & \mu_L^2 \Big/ \sum_{i=1}^{L} \mu_i^2 & \cdots & \mu_L^P \Big/ \sum_{i=1}^{L} \mu_i^P \end{bmatrix}_{L \times P}. \tag{29}$$

For each iteration during the back propagation training process of the premise part (with a chosen learning rate), we can have the above R matrix for the consequent part. Then we can apply the results of Theorems 1 and 2 to find the dynamic optimal learning rates for all the iterations during the training process of the consequent part. The following Fig. 7 shows the proposed training process of the whole FNN in Fig. 5.

The number of iterations M in the consequent part of Fig. 7 depends upon the convergent rate set by the designer. In order to find the optimal learning rate of the premise part, we rely on a genetic search algorithm. The following section will explain the details of the proposed new genetic search algorithm based on Fig. 7.

IV. Tuning FNN using a Genetic Algorithm

GAs are iterative search algorithms based on an analogy with the process of natural selection (Darwinism) and evolutionary genetics. The main goal is to search for a solution, which optimizes a user-defined function called the fitness function. To perform this task, it maintains a population or a gene pool of randomly encoded chromosomes (or individuals, solution candidates), $Pop_t = \{\alpha_t^1, \cdots, \alpha_t^{Pop\_size}\}$ for each generation t. Each α_t^i is selected randomly following a uniform distribution over search space and can be binary strings or a real value. It represents a potential solution to the problem at hand and is evaluated. Then, a new population (generation $t+1$) is formed by selecting the more fit chromosomes. Some members of the new population undergo transformation by means of genetic operators to form new solutions. After some generations, it is hoped that the best chromosome represents a near-optimal solution.

There are three operators: selection, crossover, and mutation. The selection decides which of the chromosomes in a population are selected for further genetic operations. Each chromosome i in a population is assigned a value φ_i of fitness. The fitness values are used to assign a probability value ρ_i to each chromosome. The probability value ρ_i is defined as

$$\rho_i = \varphi_i \Big/ \sum_{k=1}^{Pop\_size} \varphi_k. \tag{30}$$

The chromosome with a larger fitness value has a larger probability of selection. The crossover operation combines the features of two parent chromosomes to form two similar offspring by swapping corresponding segments of the parents. The parameters defining the crossover operation are the probability of crossover Pc and the crossover position. Mutation is a process of occasional alternation of some gene values in a chromosome by a random change with a probability less than the mutation rate Pm.

GAs [10] are used to maximize a function or to do a minimization. In our application, the error function J needs to be scaled and transformed into another function to meet the fitness evaluation requirement. For a given J, $J = \psi 10^{\lambda}$, $1 < \psi < 10$, the fitness function $\psi(J)$ is defined as [8]

$$\psi(J) = \varphi(\psi 10^{\lambda} = \begin{cases} -\lambda + 1 - \dfrac{\psi}{10}, & \text{if } \lambda < 0 \\ 10^{-(\lambda+1)} + \dfrac{1 - \psi/10}{10^{(\lambda+1)}}, & \text{if } \lambda \geq 0. \end{cases} \tag{31}$$

Equation (31) finds a larger fitness value for smaller J. In other words, if the value of J is larger, it will be mapped to a smaller fitness value and vice versa. For example, if J is 0.007, then $\lambda = -3$ and (31) will yield a fitness value of 3.3. If J is 10238, then $\lambda = 4$ and (31) will be mapped to 1.8976e-005.

Following the training process as explained in Fig. 7, we start with an initial learning rate α_0 in the premise part and proceed to train the NN with the dynamic optimal rates obtained from Theorems 1 and 2 in the consequent part. By choosing the optimal β_{opt} in each iteration in the training process of the NN, the total squared error J can be found for this initial α_0. The search must then be continued to yield the optimal α_{opt} such that the total squared error J is a minimum. It is obvious that we only have to search for α_{opt} in the FNN. The determination of β_{opt} is from Theorems 1 and 2. Otherwise, the FNN with two learning rates (to be searched for by GAs, [9]) will require much more searching time. The overall search algorithm, which summarizes the whole concept, is listed below.

Algorithm II: Tuning of FNN via Genetic Algorithm

Step 1: Initialize weighting matrix W randomly. Initialize centers η's, and widths σ's. Set values to *Iteration*, α_l, α_u, Pop_size, Max_gen, and *Threshold*.

Step 2:

For $t = 1$: *Iteration*

Initialize population $Pop = \{\alpha_i\}$, $\alpha_i \in (\alpha_l, \alpha_u)$, $i = 1, \cdots, Pop\_size$.

For *generation* = 1: Max_gen % GA

For $i = 1$: Pop_size

Get ith α.

Compute centers ith η_{t+1}'s in (26), and widths ith σ_{t+1}'s in (27).

Establish a new matrix R

While $|J_{t,\min} - J_{t-2,\min}|/J_{t-2,\min} >$ *Threshold* %update W_{t+1}

Compute matrix E in (12).

Compute A in (19), B in (20).

Compute ith β_{opt} in (21), ith W_{t+1} in (28), and ith $J_{t+1,\min}$ in (11).

end

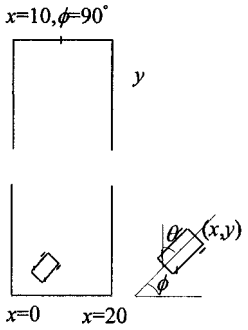


Fig. 8. Diagram of simulated truck and loading zone.

Put ith $J_{t+1,\min}$ into fitness vector.
End
Perform selection, crossover, and mutation. % for next generation
End
Optimal α_{opt} is found.
For premise part: centers (η_{t+1}), widths (σ_{t+1}), and matrix R are found.
For consequent part: $(\beta_l = 0, \beta_u)$ of β, β_{opt}, W_{t+1}, and $J_{t+1,\min}$. are found.
End

The performance of the algorithm will be illustrated using three popular examples.

V. Examples

The applications of the above GAs will be fully illustrated in this section. Example 2 is the truck back up problem. Examples 3 and 4 are nonlinear system identifications.

Example 2: Truck Back Up Problem: The well-known problem of backing up a truck into a loading dock via the FNN controller [9], [13], [14] will be considered in this section. The FNN in Fig. 5 will be fully utilized and tuned by our GAs. Fig. 8 shows the truck and loading zone.

The truck is located by three variables x, y, and ϕ, where ϕ is the angle of the truck with the horizontal axis and $0 \le x \le 20, -115^\circ \le \phi \le 295^\circ$. The steering angle θ is within $[-40^\circ, 40^\circ]$, which is to control the truck. The truck moves backward by a fixed unit distance at every step. Because we assume enough clearance between the truck and the loading zone, y is not considered. We must first prepare many pairs of data for x, ϕ, and θ as the training data such that the final state (x_f, ϕ_f) is equal or close to (10, 90°). In this simulation, we normalized $[-40^\circ, 40^\circ]$ into [0, 1].

To cover the whole situation, the following 14 initial states are used to generate desired input-output (I/O) pairs: (x_0, ϕ_0) = (1, 0°), (1, 90°), (1, -90°), (7, 0°), (7, 90°), (7, 180°), (7, −90°), (13, 0°), (13, 90°), (13, 180°), (13, 270°), (19, 90°), (19, 180°), and (19, 270°). Also, the following approximate kinematics are used:

$$x_{t+1} = x_t + \cos(\phi_t + \theta_t) + \sin(\phi_t)\sin(\theta_t) \tag{32}$$

$$y_{t+1} = y_t + \sin(\phi_t + \theta_t) - \sin(\phi_t)\sin(\theta_t) \tag{33}$$

$$\phi_{t+1} = \phi_t - \sin^{-1}\left(\frac{2\sin(\theta_t)}{l}\right) \tag{34}$$

where l is the length of truck. In this simulation, we assumed $l = 4$. Equations (32)–(34) will be used to obtain the next state when the present state and control are given. Since y is not considered, (33) will not be used. Further we let the Ts be the fuzzy membership functions (Gaussian functions) of steering angle θ. The centers of T1, T2, T3, T4, T5, T6, and T7 are -40°, -20°, -7°, 0°, 7°, 20°, and 40°, respectively. Table II shows the initial fuzzy sets of x (Q1 ∼ Q5) which are represented by the centers and widths of Gaussian functions.

TABLE II
Center and Width for Fuzzy Sets of x

	Initial		Final	
	Center	Width	Center	Width
Q1	1.5	2.1	1.49968	2.09907
Q2	7	1	7.00496	1.00494
Q3	10	0.3	9.98299	0.34094
Q4	13	1	12.98543	1.02156
Q5	18.5	2.1	18.49925	2.10107

TABLE III
Center and Width for Fuzzy Sets of ϕ

	Initial		Final	
	Center	Width	Center	Width
R1	-65	17	-64.99998	17.00004
R2	0	15	0.00002	15.00005
R3	52.5	14	52.50135	14.00273
R4	90	3	90.00288	3.00629
R5	127.5	14	127.50041	14.00024
R6	180	15	179.99998	15.00007
R7	245	17	245.00001	17.00008

TABLE IV
Fuzzy Rules

	Q1	Q2	Q3	Q4	Q5
R1	T2×0.5651	T2×0.8279	0×0.8480	0×0.3946	0×0.6629
R2	T2×0.6847	T1×0.8553	T1×0.7884	T1×0.1012	0×0.9979
R3	T5×0.9738	T3×1.0657	T2×1.1486	T1×0.2397	T2×0.9525
R4	T6×1.0639	T6×1.0785	T4×1.0118	T2×1.4621	T2×0.8656
R5	T7×0.9739	T7×0.9330	T6×0.9005	T5×0.9724	T3×1.0264
R6	0×0.1407	T7×0.8863	T7×0.8771	T7×0.8454	T6×1.0043
R7	0×0.4147	0×0.8072	0×0.9578	T7×0.8545	T6×0.9961

The centers and widths of membership functions of ϕ(R1 ∼ R7) are listed in Table III. Table IV shows the fuzzy rules.

We use 16 bits to form the chromosome pattern. The chromosomes will be mapped to the real values in range (α_l, α_u). To increase the efficiency, we define mutation rate Pm and crossover rate Pc [9] as

$$Pm = \exp^{(0.05k/Max\_gen)} - 1 \tag{35}$$

$$Pc = \exp^{(-k/Max\_gen)} \tag{36}$$

where k denotes the kth generation. Table V shows all the parameters in the GAs process.

The value of initial J is 0.051 76. After five iterations, we have an excellent result as shown in Fig. 9. Fig. 9 also shows the performance comparison with other cases ($\boldsymbol{b} \sim \boldsymbol{d}$) in which the learning rates are fixed.

Tables II and III show that final centers and widths of the membership functions have not been changed a lot from the initial ones. The optimal learning rates α_{opt}, β_{opt}, (β_l, β_u) and J of 5 iterations are shown in Table VI.

From the above table, the values of α_{opt} is very close to one, which is α's upper bound α_u, and β_{opt} is derived from (21). The final weighting factors of the fuzzy rules are shown in Table IV. During the simulation, the tolerated ranges of x and ϕ are defined as [9.85, 10.15], [89°, 91°], respectively. The truck trajectories of this simulation are shown in Figs. 10 and 11 with different initial positions (case: $\boldsymbol{a} \sim \boldsymbol{j}$).

Example 3: Nonlinear System Identification Second Order System: The plant to be identified is described by the second-order difference equation [14]–[17]

$$y(k+1) = g[y(k), y(k(1)] + u(k) \tag{37}$$

TABLE V
PARAMETERS FOR GA

Pop_size	*Max_gen*	Chromosome size (bits)	α_l	α_u	*Threshold (Algorithm II)*
14	16	16	0.01	1	0.1

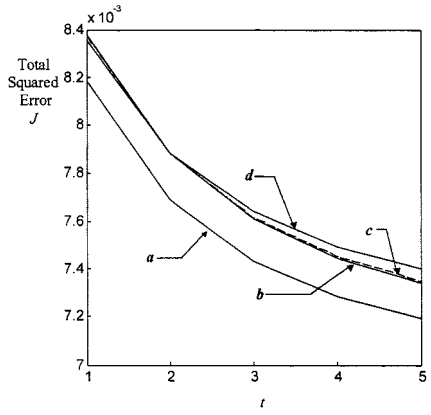


Fig. 9. Performance comparison for Example 2. *Case* ***a***: Our optimal α, β; *Case* ***b***: $\alpha = 0$, $\beta = 0.6$. *Case* ***c***: $\alpha = 0.2$, $\beta = 0.7$; *Case* ***d***: $\alpha = 0.9$, $\beta = 0.1$.

TABLE VI
LEARNING RATE α_{opt}, β_{opt}, (β_l, β_u), AND J

t	α_{opt}	β_l	β_{opt}	β_u	J
1	0.9992	0	33.5857	67.1714	0.00818
2	0.9698	0	54.7069	109.4139	0.00769
3	0.9356	0	33.9742	67.9483	0.00743
4	0.9974	0	55.3503	110.7006	0.00729
5	0.9945	0	34.1053	68.2106	0.00719

where

$$g[y(k), y(k(1)] = \frac{y(k)y(k-1)[y(k)+2.5]}{1+y^2(k)+y^2(k-1)}. \tag{38}$$

A series-parallel FNN identifier [14], [15] described by the following equation

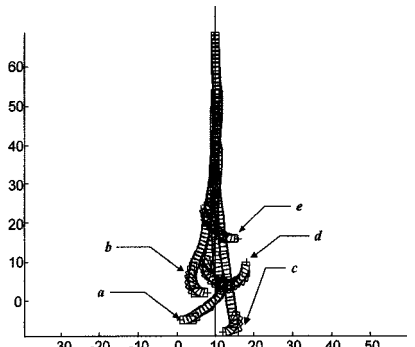
$$\hat{y}(k+1) = \hat{f}[y(k), y(k-1)] + u(k) \tag{39}$$

will be adopted, where $\hat{f}[y(k), y(k-1)]$ is in the form of (24) with two fuzzy variables $y(k)$ and $y(k-1)$. Training data of 500 points are generated from the plant model, assuming a random input signal $u(k)$ uniformly distributed in the interval $[-2, 2]$. The data are used to build fuzzy model for $\hat{f}$. The $y(k)$ and $y(k-1)$ are allocated with four fuzzy sets (Q's and R's) in Tables VIII and IX, respectively. Hence 16(= 4^2) fuzzy rules are required. The initial centers and widths of $y(k)$ and $y(k-1)$ are shown in Tables VII and VIII. The mutation rate Pm in (35) and crossover rate Pc in (36) are applied again. The parameters for the GA are listed in Table IX.

The initial value of J is 3.2268. After 606 iterations, the trajectory of J of first 29 iterations via iteration t is shown in Fig. 12. It also shows the performance comparison with other cases ($\boldsymbol{b} \sim \boldsymbol{e}$).

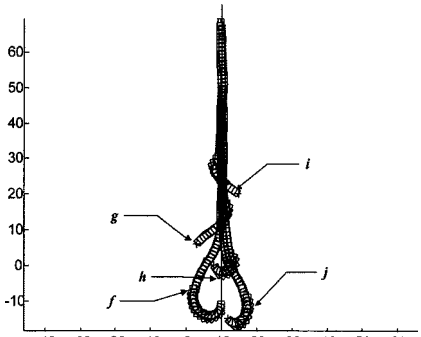
After the training process is finished, the model is tested by applying a sinusoidal input signal $u(k) = \sin(2\pi k/25)$ to the fuzzy model. Fig. 13 shows the output of both fuzzy model and the actual model. The total squared error J using 120 data items is 0.0023. This example once again shows the tremendous effect of using optimal learning rates.

The final centers and widths of membership functions for $y(k)$ and $y(k-1)$ are also listed in Tables VIII and IX, respectively. Table X shows the fuzzy rules and final weighting matrix.



	Initial (x, y, ϕ)→Final (x, y, ϕ)	No of Steps
Case *a*	(1, -5, 2°)→(9.8724, 35.5243, 89.1725°)	44
Case *b*	(8, 2, 180°)→(10.0321, 68.9107, 90.9788°)	68
Case *c*	(12, -8, 0°)→(9.8595, 50.6700, 89.4093°)	59
Case *d*	(18, 10, 269°)→(10.0559, 54.0981, 90.8841°)	60
Case *e*	(16, 16, 185°)→(9.9185, 68.1484, 90.8965°)	55

Fig. 10. Truck trajectories using fuzzy neural network controller.



	Initial (x, y, ϕ)→Final (x, y, ϕ)	No of Steps
Case *f*	(10, -10, 270°)→(10.0864, 48.5556, 90.8445°)	71
Case *g*	(3, 6, 45°)→(9.8727, 46.9418, 89.2060°)	42
Case *h*	(8, 0, -60°)→(9.8668, 34.2177, 90.9225°)	39
Case *i*	(15, 20, 145°)→(10.1420, 67.5033, 90.9323°)	48
Case *j*	(12, -15, -45°)→(9.8667, 69.2272, 89.1791°)	89

Fig. 11. Another five truck trajectories using fuzzy neural network controller.

TABLE VII
CENTER AND WIDTH FOR FUZZY SETS OF $y(k)$

	Center		Width	
	Initial	Final	Initial	Final
Q1	-1	-0.4340	0.5	1.0162
Q2	0	0.7933	0.3	0.6060
Q3	1.5	1.7889	0.4	0.9086
Q4	3	2.6810	0.6	1.1325

TABLE VIII
CENTER AND WIDTH FOR FUZZY SETS OF $y(k-1)$

	Center		Width	
	Initial	Final	Initial	Final
R1	-1	-0.5926	0.5	0.9207
R2	0	0.4880	0.3	0.8982
R3	1.5	1.7165	0.4	0.6837
R4	3	2.8133	0.6	0.9412

TABLE IX
PARAMETERS OF GA

Pop_size	*Max_gen*	Chromosome size (bits)	α_l	α_u	*Threshold (Algorithm II)*
20	30	14	0.01	4	0.08

The optimal learning rates α_{opt}, β_{opt}, (β_l, β_u) and J of certain iterations are shown in Table XI.

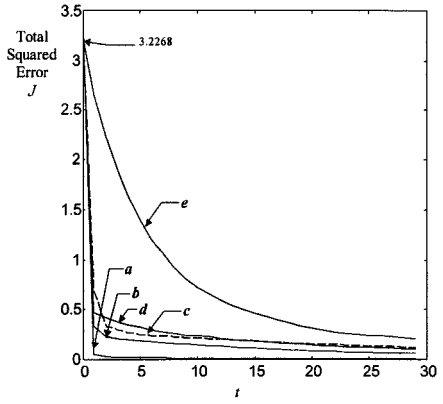


Fig. 12. Performance comparison for Example 3. *Case* ***a***: Our optimal α, β. *Case* ***b***: $\alpha = 0.5$, $\beta = 0.8$. *Case* ***c***: $\alpha = 0.4$, $\beta = 0.4$. *Case* ***d***: $\alpha = 0$, $\beta = 0.6$. *Case* ***e***: $\alpha = 0.1$, $\beta = 0.2$.

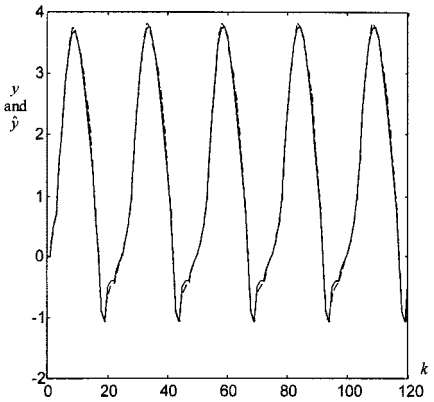


Fig. 13. Outputs of the plant y (solid line) and the identification model $\hat{y}$ (dashed line).

Example 4: Nonlinear System Identification First Order System: In Example 3, the input is seen to occur linearly in the difference equation describing the plant. In this example [15], the plant to be identified is of the following nonlinear form:

$$y(k+1) = g[y(k), u(k)] \tag{40}$$

where the unknown function g has the following nonlinear form:

$$g(x_1, x_2) = \frac{x_1}{1+x_1^2} + x_2^3 \tag{41}$$

and $u(k) = \sin(2\pi k/25) + \sin(2\pi k/10)$, $y(k)$ is distributed in range $[-7.3713, 7.4410]$. The series-parallel identification model is

$$\hat{y}(k+1) = \hat{f}[y(k), u(k)] \tag{42}$$

where $\hat{f}$ is in the form of (24) with two fuzzy variables $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. The fuzzy variables $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are defined to have five fuzzy sets respectively. Hence $25(= 5^2)$ fuzzy rules are required. Also 60 training data items are generated for training purposes. The initial centers and widths of two fuzzy variables x_1 and x_2 are shown in Tables XII and XIII. Mutation rate Pm in (35) and crossover rate Pc in (36) are applied again. The parameters for GAs are listed in Table XIV.

After 589 iterations, the trajectory of J of first 29 iterations via iteration t is shown in Fig. 14. It also shows the performance comparison with other cases ($\boldsymbol{b} \sim \boldsymbol{e}$) in which the learning rates are fixed.

TABLE X
FUZZY RULES AND FINAL WEIGHTING FACTORS w_i

	R1	R2	R3	R4
Q1	0.4117	0.4045	0.2002	0.2612
Q2	-1.1727	1.1728	0.9774	0.6328
Q3	-1.7721	1.6982	1.9583	1.1646
Q4	-1.9223	1.3623	2.1867	3.0141

TABLE XI
LEARNING RATE α_{opt}, β_{opt}, (β_l, β_u), AND J

t	α_{opt}	β_l	β_{opt}	β_u	J
1	0.0158	0	13.0182	26.0363	0.0536
2	0.0102	0	12.8502	25.7005	0.0290
3	3.9992	0	13.1265	26.2530	0.0198
⋮	⋮	⋮	⋮	⋮	
604	3.7506	0	22.7768	45.5536	7.5045e-003
605	3.5131	0	6.3826	12.7652	7.5041e-003
606	3.9377	0	24.2665	48.5329	7.5037e-003

TABLE XII
CENTER AND WIDTH FOR FUZZY SETS OF x_1

	Center		Width	
	Initial	Final	Initial	Final
Q1	-5	-5.8543	0.5	4.5812
Q2	-3	-2.3761	0.3	2.0577
Q3	0	0.4524	0.3	1.7701
Q4	3	3.1177	0.4	0.5958
Q5	5	5.2009	0.6	2.2813

TABLE XIII
CENTER AND WIDTH FOR FUZZY SETS OF x_2

	Center		Width	
	Initial	Final	Initial	Final
R1	-2	-2.0591	0.6	0.9166
R2	-1	-1.2908	0.6	0.7372
R3	0	-0.0064	0.6	0.6909
R4	1	1.1998	0.6	0.7644
R5	2	2.0915	0.6	0.8010

Fig. 15 shows the outputs and the model after the identification procedure was terminated. After recall, the total squared error J using 120 data items is 0.0131.

The final centers and widths of membership functions of x_1 and x_2 are listed in Tables XII and XIII, respectively. Table XV shows the fuzzy rules and final weighting matrix.

The optimal learning rates α_{opt}, β_u and J of certain iterations are shown in Table XVI. Due to $\beta_{opt} = [\beta_l(= 0) + \beta_u]/2$, we avoid showing the value of β_{opt}.

VI. CONCLUSION

The stability analysis of a dynamic learning rate in a simple two-layer neural network is first explored in this paper. It has been found that the dynamic stable optimal learning rates in the sense of maximum error reduction must be positive. This result can be used in any dynamic FNN that includes the simple two-layer NN as the consequent part. In order to demonstrate this effectiveness, a genetic algorithm is devised to fully utilize this result to fine-tune the two learning rates in a dynamic FNN. In this case, we only have to search for the optimal learning rate in the premise part of the FNN. The other optimal learning rate for the consequent part can be determined immediately from the proposed innovative approach for the two-layer NN in this paper. Several popular examples are considered and it has been found that in all the examples the FNNs are trained in a convergent way as expected. Performance comparisons with different

TABLE XIV
PARAMETERS FOR GA

Pop_size	*Max_gen*	Chromosome size (bits)	α_l	α_u	*Threshold*
20	30	18	0.01	8	0.08

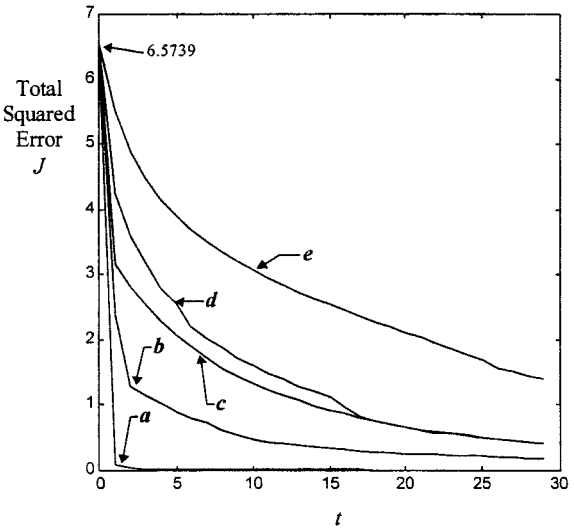


Fig. 14. Performance comparison for Example 4. *Case* ***a***: Our optimal α, β. *Case* ***b***: $\alpha = 0.5$, $\beta = 0.8$. *Case* ***c***: $\alpha = 0$, $\beta = 0.6$. *Case* ***d***: $\alpha = 0.4$, $\beta = 0.4$. *Case* ***e***: $\alpha = 0.1$, $\beta = 0.2$.

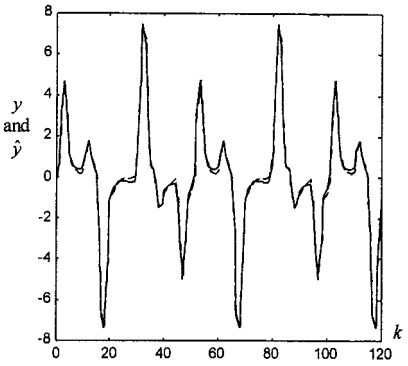


Fig. 15. Outputs of the plant y (solid line) and the identification model $\hat{y}$ (dashed line).

TABLE XV
FUZZY RULES AND FINAL WEIGHTING FACTORS w_i

	R1	R2	R3	R4	R5
Q1	-12.4725	3.4060	-3.0671	-2.0815	3.0663
Q2	-11.2106	0.6830	-1.0486	-1.3534	2.3886
Q3	-8.4582	1.1653	0.9901	0.6677	11.0607
Q4	-2.5946	2.9471	-1.1356	0.6394	8.4535
Q5	-1.1667	6.8722	3.5256	-1.1096	10.8100

learning rates in all examples are also presented. It is believed that the new results presented in this paper can be applied to any other applications that utilize the dynamic FNNs, which includes two-layer NNs.

TABLE XVI
LEARNING RATE α_{opt}, β_{opt}, (β_l, β_u), AND J

t	α_{opt}	β_l	β_{opt}	β_u	J
1	0.0646	0	12.9452	25.8905	0.0936
2	0.0694	0	14.4257	28.8514	0.0409
3	0.6018	0	10.4365	20.8731	0.0225
⋮	⋮	⋮	⋮	⋮	⋮
439	0.0375	0	8.2795	16.5590	5.8976e-3
440	7.9881	0	16.1084	32.2168	5.8626e-3
441	0.1206	0	10.4802	20.9604	5.8464e-3
⋮	⋮	⋮	⋮	⋮	⋮
587	0.2126	0	9.9095	19.8190	5.5508 e-3
588	0.2517	0	10.3565	20.7129	5.5493 e-3
589	0.2202	0	10.0279	20.0558	5.5473e-3

REFERENCES

[1] C. T. Lin and Y. C. Lu, "A neural fuzzy system with fuzzy supervised learning," *IEEE Trans. Syst., Man, Cybern. B*, vol. 26, pp. 744–763, Oct. 1996.

[2] W. Y. Wang *et al.*, "Function approximation using fuzzy neural networks with robust learning algorithm," *IEEE Trans. Syst., Man, Cybern. B*, vol. 27, pp. 740–747, Aug. 1997.

[3] J. T. Spooner and K. M. Passino, "Stable adaptive control using fuzzy systems and neural networks," *IEEE Trans. Fuzzy Syst.*, vol. 4, pp. 339–359, Aug. 1996.

[4] C. H. Wang *et al.*, "Fuzzy B-spline membership function (BMF) and its applications in fuzzy-neural control," *IEEE Trans. Syst., Man, Cybern.*, vol. 25, pp. 841–851, May 1995.

[5] X. H. Yu *et al.*, "Dynamic learning rate optimization of the backpropagation algorithm," *IEEE Trans. Neural Networks*, vol. 6, pp. 669–677, May 1995.

[6] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs.*, 3rd ed. New York: Springer-Verlag, 1996.

[7] K. S. Tang *et al.*, "Structured genetic algorithm for robust H^{∞} control systems design," *IEEE Trans. Ind. Electron.*, vol. 43, pp. 575–582, Oct. 1996.

[8] C.-C. Hsu *et al.*, "Digital redesign of continuous systems with improved suitability using genetic algorithms," *IEE Electron. Lett.*, vol. 33, no. 15, pp. 1345–1347, July 1997.

[9] T. L. Seng *et al.*, "Tuning of a neural-fuzzy controller by genetic algorithm," *IEEE Trans. Syst., Man, Cybern. B*, vol. 29, pp. 226–236, Apr. 1999.

[10] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Cambridge, MA: MIT Press, 1992.

[11] V. Solo and X. Kong, *Adaptive Signal Processing Algorithms Stability and Performance*. Englewood Cliffs, NJ: Prentice- Hall, 1995.

[12] S. Horikawa *et al.*, "On fuzzy modeling using fuzzy neural networks with back-propagation algorithm," *IEEE Trans. Neural Networks*, vol. 3, pp. 801–806, Sept. 1992.

[13] B. Kosko, *Neural Network and Fuzzy Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1992.

[14] L. X. Wang, *Adaptive Fuzzy Systems and Control: Design and Stability Analysis*. Englewood Cliffs, NJ: Prentice-Hall, 1994.

[15] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4–26, Mar. 1990.

[16] S. Barada and H. Singh, "Generating optimal adaptive fuzzy-neural models of dynamical systems with applications to control," *IEEE Trans. Syst., Man, Cybern. B*, vol. 28, pp. 371–390, Aug. 1998.

[17] W. A. Farag *et al.*, "A genetic-based neural-fuzzy approach for modeling and control of dynamical systems," *IEEE Trans. Neural Networks*, vol. 9, pp. 756–767, Sept. 1998.