A New Learning Algorithm for a Fully Connected Neuro-Fuzzy Inference System

C. L. Philip Chen, Fellow, IEEE, Jing Wang, Student Member, IEEE, Chi-Hsu Wang, Fellow, IEEE, and Long Chen, Member, IEEE

Abstract—A traditional neuro-fuzzy system is transformed into an equivalent fully connected three layer neural network (NN), namely, the fully connected neuro-fuzzy inference systems (F-CONFIS). The F-CONFIS differs from traditional NNs by its dependent and repeated weights between input and hidden layers and can be considered as the variation of a kind of multilayer NN. Therefore, an efficient learning algorithm for the F-CONFIS to cope these repeated weights is derived. Furthermore, a dynamic learning rate is proposed for neuro-fuzzy systems via F-CONFIS where both premise (hidden) and consequent portions are considered. Several simulation results indicate that the proposed approach achieves much better accuracy and fast convergence.

Index Terms—Fully connected neuro-fuzzy inference systems (F-CONFIS), fuzzy logic, fuzzy neural networks, gradient descent, neural networks (NNs), neuro-fuzzy system, optimal learning.

I. INTRODUCTION

NEURO-FUZZY systems have been applied to many engineering applications in decades related to pattern recognition, intelligent adaptive control, regression and density estimation, systems modeling, and so on [1]–[6]. A neuro-fuzzy system possesses characteristics of neural network (NN), linguistic description and logic control [7]–[10]. Although the significant progress has been made by combining different learning algorithms with neuro-fuzzy system [11]–[16], there are still problems that need to be solved for practical implementations for instance, finding the optimal learning rates for both the premise and consequent parts to increase convergence speed, or updating the parameters of membership functions (MFs). In a neuro-fuzzy system, in general, the rule layer is a product layer instead of a summing layer in a conventional feedforward NN. As a result, it is not concise to apply learning algorithms in turning premise parameters. Therefore, to design a systematic learning for the neuro-fuzzy system, a traditional neuro-fuzzy system is reformulated as an equivalent fully connected three-layer NN, i.e., the fully connected fuzzy inference systems (F-CONFIS) [17]. Though some literatures have proved the functional equivalence between a fuzzy system and a NN, they are nonconstructive [9]. The F-CONFIS provides constructive steps to build the equivalence between a neuro-fuzzy system and NN. The F-CONFIS is different with the classical multiple layer NNs by its repeated link weights. With some special arrangements, we can derive the training algorithm for the F-CONFIS, thereafter for a neuro-fuzzy system efficiently and effectively.

Choosing a proper learning rate is a critical issue in the gradient descent algorithm for a neuro-fuzzy system. It is a time consuming process to select manually and the result may lack of generality. In a neuro-fuzzy system, it needs to adjust two learning rates corresponding to adjustable parameters in the premise and the consequent parts. To improve the convergence rate, although many kinds of dynamic learning rate methods have been proposed for neuro-fuzzy systems [16], [18]–[22], the convergence rate was still not improved satisfactorily because only the optimal learning rate of the consequent part is derived. Although the learning rate of the premise part can be obtained by genetic search techniques in [18], of which is time consuming and the optimal result is not guaranteed. The optimal learning rate of the consequent part can be obtained analytically, but unfortunately, the analytical expression of the premise part cannot be solved in a similar way because the premise part is usually a transcendental equation. Due to the complexity of the premise part of neuro-fuzzy systems, there is no existing learning algorithm about the optimal learning rate of the premise portion that has been discussed. In this paper, by transforming a neuro-fuzzy system to F-CONFIS, a dynamic learning algorithm is proposed.

The major contributions of the paper are summarized as follows.

1) We derive the ordinal indices for hidden neurons of the repeated weights and, thereafter, the gradient descent-based training algorithm of F-CONFIS.
2) The dynamic optimal learning rates of neuro-fuzzy system are derived first time not only in the consequent part, but also in the premise part. The optimal learning rate for the parameters of MFs can greatly boost the training speed of the neuro-fuzzy systems. Such kind of speedup is extremely valuable for the online applications.
3) In addition, this paper explicitly provides formulations to update the parameters of fuzzy MFs in the premise part.
Section V. Finally, this paper is concluded in Section VI.

The fuzzy Mamdani model [23] represented by above configuration of a neuro-fuzzy system contains following rules.

\[ r_i: \text{the ordinal index of } r \text{th MFs for fuzzy variable } x_i \]
\[ \mu_{l}^{A_{1r_{l}}(x)}(x_{1}) \times \cdots \times A_{N_{r_{N}}}(x_{N}) \]
\[ \{l = 0, 1, 2, \ldots, L - 1; r_{l} = 0, 1, 2, \ldots, R_{l} - 1\}. \] (1)

There are two difficulties in developing a learning algorithm for the neuro-fuzzy system. One is that, in a neuro-fuzzy system, the links between the MF layer, and the fuzzy rule layer are not fully connected, so it is selective learning. The other is that the operators in the fuzzy rule layer are product-form rather than summation form.

This kind of four-layer neuro-fuzzy system can be transformed into an equivalent F-CONFIS [17]. For convenience and completeness, we briefly describe the derivation of F-CONFIS here. In order to overcome the above mentioned difficulties, the MF layer II of a neuro-fuzzy system in Fig. 1 can be redrawn, and substituted for new links between Layer I and Layer III, so that we can have a fully connected three-layer NN as shown in Fig. 2. The exponential function was taken as the activation function in F-CONFIS. Therefore, the new link weight \( V_{ij} \) for the neuro-fuzzy system was discussed in [17].

\[ V_{ij} = L \ln(A_{r_{l}(j)}(x_{i})) \] (2)

where \( \ln \) is the natural logarithm, and \( r_{l}(j) \) \((i = 1, \ldots, N)\) are ordinal indices of MFs corresponding to rule \( j \).

Fig. 2 depicts the equivalent F-CONFIS, and it is equivalent to the original neuro-fuzzy system shown in Fig. 1. The F-CONFIS has three layers as shown in Fig. 2 and its hidden layer is also the fuzzy rule layer.

B. Special Properties of F-CONFIS

The F-CONFIS differs from traditional multilayer NNs. For a normal multilayer feedforward NN, every weight is updated only once to reduce the error in the process of each epoch. While in F-CONFIS, the weights may be updated only once to reduce the error in the process of each epoch.
more than one time. For example, see the configuration of a neuro-fuzzy system in Fig. 3 and its equivalent F-CONFIS shown in Fig. 4. The configuration of the neuro-fuzzy system (or the F-CONFIS) has two input variables \( x_1 \) and \( x_2 \), two output variables \( y_1 \) and \( y_2 \), and two MFs \( A_{10}(x_1), A_{11}(x_1) \) for \( x_1 \) and three MFs \( A_{20}(x_2), A_{21}(x_2), A_{22}(x_2) \) for \( x_2 \), \( R_1 = 2, R_2 = 3 \). For instance, in Fig. 4, the links \( V_{10}, V_{12}, \) and \( V_{14} \) are with the same weight of \( \ln(A_{10}(x_1)) \) from the first MF of \( x_1 \) to hidden neurons as shown in Table I. This kind of multiple updates should be handled carefully and will be discussed next.

With above description, it is important to find the number of repeated links \( RW(i) \) for each fuzzy variable \( x_i \) in premise part so that the training algorithm can be properly carried out for the F-CONFIS. The following proposition 1 shows a precise formula of finding the number of repeated links \( RW(i) \) in the F-CONFIS.

### Proposition 1

In a F-CONFIS, each input variable \( x_i \) has the number of repeated links \( RW(i) \), every same value of weights \( A_{i_1}(x_i) \) will be repeated for \( RW(i) \) times between input layer and hidden layer, \( RW(i) \) is given as

\[
RW(i) = \prod_{i=1}^{n} R_i \quad (i = 1, \ldots, N)
\]

where \( R_i \) is the number of MFs for input variable \( x_i \), \( A_{i_1}(x_i) \) is the \( r_i \)th MF of \( x_i \).

From (1), we have

\[
\mu_l = \prod_{i=1}^{n} A_{i_1}(x_i)
\]

\[|l = 0, 1, 2, \ldots, L - 1; r_i = 0, 1, 2, \ldots, R_i - 1\].

To be identical with \( l \) defined in [24], the ordinal index \( l \) of the \( l \)th fuzzy rule is

\[
l = r_1 + \sum_{i=1}^{N} \left( \prod_{j=1}^{i-1} R_j \right) r_i.
\]

From (4) and (5), the total number of fuzzy rules is

\[
L = R_1 \times R_2 \times \cdots \times R_i \times \cdots \times R_N.
\]

In F-CONFIS, there are \( L \) links from every node of the input layer to all of nodes of the fuzzy rule layer. However, there are only \( R_i \) different MFs for \( x_i \) \([A_{iq} q = 0, 1, 2, \ldots, R_i - 1\].

From (4) and (6), it is obvious that for each fuzzy variable \( x_i \) to have the number of repeated links, \( RW(i) \), with the same weights

\[
RW(i) = \frac{R_1 \times R_2 \times \cdots \times R_i \times \cdots \times R_n}{R_i} = \frac{\prod_{i=1}^{n} R_i}{R_i}.
\]

For instance, in Fig. 4, the number of repeated links of variables \( x_2 \) is as follows:

\[
RW(2) = \prod_{i=1}^{3} R_i = \frac{R_1 \times R_2}{R_2} = \frac{6}{3} = 2
\]

which are \( V_{20} = V_{21} = \ln(A_{20}(x_2)), V_{22} = V_{23} = \ln(A_{21}(x_2)), \) and \( V_{24} = V_{25} = \ln(A_{22}(x_2)) \) for fuzzy variable \( x_2 \) as shown in Table I.

The derivation of a new learning algorithm for F-CONFIS is discussed in the following sections.

### III. NEW OPTIMAL LEARNING OF F-CONFIS

The F-CONFIS is a new type of NN that has dependent and repeated links between input and hidden layers. With Proposition 1 that shows a precise formula of finding the number of repeated links, \( RW(i) \), in F-CONFIS, an explicit learning algorithm considering the dependent and repeated weights is proposed next.

A gradient decent algorithm, may be the most commonly used, with fixed learning rates may lead to slow convergence [25]–[30]. The choosing of proper learning rate becomes a critical issue in a gradient decent-based algorithm. To deal with
this issue, many kinds of dynamic learning rate methods have been proposed [12]–[16]. However, only the optimal learning rate of the consequent part is derived for neuro-fuzzy systems [18]–[22]. Although the learning rate for the premise part is obtained by genetic search techniques [18], it is generally time consuming and the searched result may not be optimal. Due to the complexity of the premise part of neuro-fuzzy systems, no existing learning algorithm about the optimal learning of the premise portion has been discussed. In this section, based on the F-CONFIS, we derive the new optimal learning rate for both the premise part and the consequent part.

A popular centroid defuzzification method is used in the output layer as

$$y_k = \frac{\sum_{j=0}^{L-1} \mu_j w_{jk}}{\sum_{j=0}^{L-1} \mu_j}, (k = 1, 2, \ldots, M) \tag{7}$$

where the output vectors is \(\{y_k, k = 1, 2, \ldots, M\}\). The number of training patterns is \(P\). The total training error [17] is given as

$$E(W) = \frac{1}{2P} \sum_{p=1}^{P} \sum_{q=1}^{M} (y^p_q - d^p_q)^2 \tag{8}$$

where \(y^p_q\) is the \(k\)th actual output and \(d^p_q\) is the \(k\)th desired output. In F-CONFIS, the Gaussian MFs are adopted

$$A_{iq}(x) = \exp \left[ \frac{-(x-m_{iq})^2}{2\sigma_{iq}^2} \right] \quad i = 1, \ldots, N; \quad q = 0, \ldots, R_i - 1 \tag{9}$$

where \(m_{iq}\) and \(\sigma_{iq}\) denote the center and width of the MFs, respectively.

In F-CONFIS and its corresponding neuro-fuzzy system, two types of adjustable parameters should be adjusted, i.e., the adjustable parameters, \(W\) consist of the parameters of MFs in the premise part and weight factors in consequent part. Therefore, weighting vector \(W\) is defined as

$$W = [m_{iq} \quad \sigma_{iq} \quad w_{jk}]$$

$$i = 1, \ldots, N; \quad q = 0, \ldots, R_i - 1; \quad k = 1, \ldots, M \tag{10}$$

where \(w_{jk}\) are the link weights between the fuzzy rule layer and output layer. The gradient of \(E(W)\) with respect to \(W\) is given by

$$g = \left[ \frac{\partial E}{\partial m_{iq}}, \frac{\partial E}{\partial \sigma_{iq}}, \frac{\partial E}{\partial w_{jk}} \right]$$

$$i = 1, \ldots, N; \quad q = 0, \ldots, R_i - 1$$

$$j = 0, \ldots, L - 1; \quad k = 1, \ldots, M. \tag{11}$$

The weighting vector \(W\) can be tuned as

$$W(\alpha, \beta) = W - \left[ \alpha \frac{\partial E}{\partial m_{iq}}, \alpha \frac{\partial E}{\partial \sigma_{iq}}, \beta \frac{\partial E}{\partial w_{jk}} \right]$$

$$i = 1, \ldots, N; \quad q = 0, \ldots, R_i - 1$$

$$j = 0, \ldots, L - 1; \quad k = 1, \ldots, M. \tag{12}$$

From (5), all the \(r_j\)'s for a specific rule number \(l\) are represented as

$$r_1(l) = l\%R_i; \quad r_j(l) = \left( i - 1 \right) \sum_{k=1}^{j} R_k \% R_i, \quad i = 2, \ldots, N. \tag{13}$$

From Fig. 5, we know that the repeated weight for the \(q\)th MF of \(x_i\) is equal to \(A_{iq} = A_{iq}(x_i)\). Therefore, we have

$$q = r_1(\ell_j) \Rightarrow \{\ell_j = h_j(i, q); \quad l = 1, \ldots, RW(i)\} \tag{14}$$

where \(\ell_j = h_j(i, q); \quad l = 1, \ldots, RW(i)\) in (14) are the links with repeated weight \(A_{iq}\) for the \(q\)th MF of \(x_i\). For convenience, we define following \(s_i\):

$$s_i = 1, \quad i = 1$$

$$s_i = \prod_{k=1}^{i-1} R_k, \quad i > 1 \quad \text{or} \quad s_{i+1} = s_i R_i, \quad i = 1, \ldots, N - 1 \tag{15}$$

so that we can rewrite (13) to find the \(l\)th fuzzy rule in hidden layer as

$$l(r_1, r_2, \ldots, r_N) = \sum_{n=1}^{N} r_n s_n. \tag{16}$$

Let \(A = [r_1, r_1-l, \ldots, q, r_{i+1}, \ldots, r_N] [0 < r_n < R_n, n = 1, 2, \ldots, i-1, i+1, \ldots, N] \) and \(B = [h_j(i, q)l; l = 1, \ldots, RW(i)]\). Then, it is obvious that the following equation will generate an one to one mapping from \(A\) to \(B\) using a fixed \(q\):

$$l(r_1, r_2, \ldots, r_l-1, q, r_{i+1}, \ldots, r_N)$$

$$= \sum_{n=1}^{i-1} r_n s_n + q s_i + \sum_{n=1}^{N} r_n s_n$$

$$= \sum_{n=1}^{i-1} r_n s_n + q s_i + \sum_{n=i+1}^{N} r_n s_n$$

$$= r_0 = 0 \quad \text{and} \quad r_{N+1} = 0. \tag{17}$$

It is implicitly assumed that all the \(l\)'s generated via (17) will be in ascending order by the mechanism described in the Appendix. It is also obvious to see that the cardinality of \(A\) equals the cardinality of \(B\)

$$\text{Dim}(A) = \left( \prod_{n=1}^{i-1} R_i \right) \times 1 \times \left( \prod_{n=i+1}^{N} R_i \right)$$

$$= \left( \prod_{n=1}^{i-1} R_i \right) = \text{Dim}(B). \tag{18}$$

In the error gradient decent process, error will propagate along interconnection \(V_{ih}(i, q)(l = 1, \ldots, RW(i))\) from \(\mu_{ih}(i, q)(l = 1, \ldots, RW(i))\) to \(x_i\). The signal flow for the center of the first MF of \(x_i\), namely \(\hat{E}/\hat{m}_{10}\), is drawn in Fig. 6 (from Fig. 4) to demonstrate the approach of finding
dependent and repeated links between input layer and hidden layer in F-CONFIS. For instance, in order to find \( \partial E / \partial m_{10} \) of premise part in Fig. 5, we need to know that \( V_{10} = V_{12} = V_{14} = \ln(A_{10}(x_1)) \) are the links with the repeated weight of \( \ln(A_{10}(x_1)) \) from the first MF of \( x_1 \) to hidden neurons 0, 2, and 4.

Although (17) can indeed generate \( h_{j_1}(i, q)|\lambda = 1, \ldots, RW(i) \) by considering all the combinations of \( r_i \), it still not precise enough for the gradient descent training of F-CONFIS. The following Theorem 1 will show a precise formula of finding the ordinal indices of these links with repeated weights for the \( q \)th MF of the \( i \)th fuzzy input variable \( x_j \).

**Theorem 1:** The ordinal indices \( h_{j_2}(i, q)|\lambda = 1, \ldots, RW(i) \) for hidden neurons with links of repeated weights, which connect the \( q \)th MF of the \( i \)th fuzzy input variable \( x_j \), can be found from the following equation:

\[
h_{j_2}(i, q) = (\lambda - 1)q_{j_1} + q_{j_1} + [(\lambda - 1)/\lambda]R_i\lambda_j \quad (i = 1, \ldots, N; \ q = 0, \ldots, R_i - 1; \ \lambda = 1, \ldots, RW(i)) \tag{18}
\]

where \( A \% B \) represents the remainder of the division of \( A \) over \( B \), and \( A/B \) represents the quotient of the integer division of \( A \) over \( B \).

**Proof:** Please refer to the Appendix.

For illustration, the F-CONFIS in Fig. 4, \( RW(1) = 3 \) implies that there are three repeated links with the same weight, which connect three hidden neurons and \( x_1 \). Therefore, from Theorem 1, the ordinal indices \( h_{j_2}(i, q)|i = 1; \ q = 0; \ \lambda = 1, 2, 3 \) for hidden neurons with links of three repeated weights, which connect the 0th MF of the 1st fuzzy input variable \( x_1 \), is given as \( (s_1 = 1) \)

\[
egin{align*}
h_1(1, 0) &= (1 - 1)%1 + 0 * 1 + (0/1) * 2 * 1 = 0 \\
h_2(1, 0) &= (2 - 1)%1 + 0 * 1 + (1/1) * 2 * 1 = 2 \\
h_3(1, 0) &= (3 - 1)%1 + 0 * 1 + (2/1) * 2 * 1 = 4.
\end{align*}
\]

The ordinal indices \( h_{j_2}(i, q)|i = 1; \ q = 1; \ \lambda = 1, 2, 3 \) connect the first MF of the 1st fuzzy input variable can be found as \( (s_1 = 1) \)

\[
egin{align*}
h_1(1, 1) &= (1 - 1)%1 + 1 * 1 + (0/1) * 2 * 1 = 1 \\
h_2(1, 1) &= (2 - 1)%1 + 1 * 1 + (1/1) * 2 * 1 = 3 \\
h_3(1, 1) &= (3 - 1)%1 + 1 * 1 + (2/1) * 2 * 1 = 5.
\end{align*}
\]

From Fig. 4, links carry \( \ln(A_{10}(x_1)) \) are connected to \( \mu_0, \mu_2, \) and \( \mu_4 \) of hidden nodes, and links carry \( \ln(A_{11}(x_1)) \) are connected to \( \mu_1, \mu_3, \) and \( \mu_5 \) of hidden nodes.

Therefore, from the above illustration, it is important to find the updating law of the repeated links for F-CONFIS, especially in a large neural fuzzy inference network. The following Theorem 2 will show a precise updating law of F-CONFIS.

**Theorem 2:** Applying the gradient descent algorithm for F-CONFIS, the gradient components of the premise part should be updated \( RW(i) \) times, where \( RW(i) \) is the total number of repeated links. In the consequent part, the gradient component is updated only one time. The gradient components are updated as

\[
m_{iq}(t + 1) = m_{iq}(t) - \alpha \sum_{j=1}^{RW(i)} \frac{\partial E}{\partial \mu_{h_{j_2}(i, q)}} \frac{\partial \mu_{h_{j_2}(i, q)}}{\partial A_{iq}} \frac{\partial A_{iq}}{\partial m_{iq}}
\]

\[
s_{iq}(t + 1) = s_{iq}(t) - \alpha \sum_{j=1}^{RW(i)} \frac{\partial E}{\partial \lambda_{h_{j_2}(i, q)}} \frac{\partial \lambda_{h_{j_2}(i, q)}}{\partial A_{iq}} \frac{\partial A_{iq}}{\partial s_{iq}}
\]

and the gradient components of Gaussian-type learning algorithm are updated as

\[
m_{iq}(t + 1) = m_{iq}(t) - \beta \frac{\partial E}{\partial W_{jk}}
\]

\[
s_{iq}(t + 1) = s_{iq}(t) - \beta \frac{\partial E}{\partial W_{jk}}
\]

\[
w_{jk}(t + 1) = w_{jk}(t) - \beta \frac{\partial E}{\partial W_{jk}}
\]

\[
i = 1, \ldots, N, \ j = 0, \ldots, L - 1, k = 1, \ldots, M,
\]

\[
q = 0, \ldots, R_i - 1, \ \lambda = 1, \ldots, RW(i).
\]

**Proof:** By Proposition 1, we know that each variable \( x_i \) has \( RW(i) \) repeated links between the input layer and the hidden layer. Let \( A_{iq}(q \in [1, R_i - 1]) \) be the \( q \)th MF of the variable \( x_i \), \( (\ell = h_{j_2}(i, q) ; \ \lambda = 1, \ldots, RW(i)) \) in (14) are the links with repeated weight \( A_{iq} \) for the \( q \)th MF of
$x_i$, and the weights of the repeated links are $V_{ih_\ell(i,q)}(\ell = 1, \ldots, \text{RW}(i))$.

Let $m_{iq}$ and $\sigma_{iq}$ be the control parameters of $A_{iq}$. In the error gradient decent process, error will propagate along interconnection $V_{ih_\ell(i,q)}(\ell = 1, \ldots, \text{RW}(i))$ to $x_i$. To get error’s partial derivative with respect to $m_{iq}$ and $\sigma_{iq}$, from (1), (2), (4), and (8) by the chain rule, we can get these partial derivates as

$$
\begin{align*}
\frac{\partial E}{\partial m_{iq}} &= \frac{1}{P} \sum_{p=1}^{P} \sum_{k=1}^{M} \left( (y_p^k - d_p^k) (w_{jk} - y_p^k) \right) \mu_j \left( x_i^p - m_{iq} \right) \\
\frac{\partial E}{\partial \sigma_{iq}} &= \frac{1}{P} \sum_{p=1}^{P} \sum_{k=1}^{M} \left( (y_p^k - d_p^k)^2 (w_{jk} - y_p^k) \right) \mu_j \left( x_i^p - m_{iq} \right)
\end{align*}
$$

From (8), (9), (12), and (13), we have

$$
\begin{align*}
\frac{\partial E}{\partial m_{iq}} &= \frac{1}{P} \sum_{p=1}^{P} \sum_{k=1}^{M} \left( (y_p^k - d_p^k) (w_{jk} - y_p^k) \right) \mu_j \left( x_i^p - m_{iq} \right) \\
\frac{\partial E}{\partial \sigma_{iq}} &= \frac{1}{P} \sum_{p=1}^{P} \sum_{k=1}^{M} \left( (y_p^k - d_p^k)^2 (w_{jk} - y_p^k) \right) \mu_j \left( x_i^p - m_{iq} \right)
\end{align*}
$$

From (7) and (8), by chain rule, we have

$$
\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial y_p^k} \frac{\partial y_p^k}{\partial w_{jk}} = \frac{1}{P} \sum_{p=1}^{P} (y_p^k - d_p^k) \mu_j \left/ \sum_{l=0}^{L-1} \mu_l \right.
$$

$$
j = 0, \ldots, L - 1; k = 1, \ldots, M.
$$

After finding $\text{RW}(i)$ by Proposition 1, all these intermediate partial derivatives should be summed up to give the final update for $m_{iq}$ and $\sigma_{iq}$ as the following equations:

$$
\begin{align*}
\frac{\partial E}{\partial m_{iq}} &= \sum_{l=1}^{\text{RW}(i)} \frac{\partial E}{\partial m_{iq}} \frac{\partial m_{iq}}{\partial m_{iq}} \\
\frac{\partial E}{\partial \sigma_{iq}} &= \sum_{l=1}^{\text{RW}(i)} \frac{\partial E}{\partial \sigma_{iq}} \frac{\partial \sigma_{iq}}{\partial \sigma_{iq}}
\end{align*}
$$

In practical applications, given a hidden node number $l$, we can reversely find ordinal index of MFs for all fuzzy variable, namely $r_i(l)(i = 1, 2, \ldots, N)$ by (13). By (23), we can calculate one item in the right-hand side of (23) for every MF connected to hidden node. Traversing all the hidden nodes, we can find all the items in the right-hand side of (23) for every MFs. In consequent part, because there is no repeated links, the gradient component should be updated only one time. Therefore, the components of $W$ are finally updated by the following equations:

$$
\begin{align*}
m_{iq}(t + 1) &= m_{iq}(t) - \alpha \sum_{l=1}^{\text{RW}(i)} \frac{\partial E}{\partial m_{iq}} \frac{\partial m_{iq}}{\partial m_{iq}} \\
\sigma_{iq}(t + 1) &= \sigma_{iq}(t) - \alpha \sum_{l=1}^{\text{RW}(i)} \frac{\partial E}{\partial \sigma_{iq}} \frac{\partial \sigma_{iq}}{\partial \sigma_{iq}}
\end{align*}
$$

From (19), (21), and (22), we have

$$
\begin{align*}
m_{iq}(t + 1) &= m_{iq}(t) - \alpha \frac{1}{P} \sum_{p=1}^{P} \sum_{k=1}^{M} \left( (y_p^k - d_p^k) (w_{jk} - y_p^k) \mu_{jk} \mu_j (x_i^p - m_{iq}) \right) \\
\sigma_{iq}(t + 1) &= \sigma_{iq}(t) - \alpha \frac{1}{P} \sum_{p=1}^{P} \sum_{k=1}^{M} \left( (y_p^k - d_p^k)^2 (w_{jk} - y_p^k) \mu_{jk} \mu_j \left( x_i^p - m_{iq} \right) \right)
\end{align*}
$$

For example, the number of repeated links $\text{RW}(1)$ is 3 by Proposition 1. According to Theorem 2, the gradient components $m_{10}$ of the premise part should be updated $\text{RW}(1)$ times. Therefore, we have

$$
m_{10}(t + 1) = m_{10}(t) - \alpha \sum_{l=1}^{3} \frac{\partial E}{\partial m_{10}} \frac{\partial m_{10}}{\partial m_{10}}
$$

For notational convenience, let $m_{ij}(\alpha) = m_{ij}(t + 1)$, $\sigma_{ij}(\alpha) = \sigma_{ij}(t + 1)$, $W(\alpha, \beta) = W(t + 1)$, $[\Delta m_{ij}, \Delta \sigma_{ij}, \Delta w_{jk}] = [\partial E/\partial m_{ij}, \partial E/\partial \sigma_{ij}, \partial E/\partial w_{jk}]$. We can explicitly express weights and error function in terms of learning rates. From (12), we have

$$
W(\alpha, \beta) = W - [\alpha \Delta m_{ij}, \alpha \Delta \sigma_{ij}, \beta \Delta w_{jk}].
$$

After weights are updated, we can get new error with respect to new weights $E(\alpha, \beta) = E(W(\alpha, \beta))$. Gradient descent algorithm updates weights in negative gradient direction $-g = -[\Delta m_{ij}, \Delta \sigma_{ij}, \Delta w_{jk}]$. The weights update is $\Delta W(\alpha, \beta) = -[\alpha \Delta m_{ij}, \alpha \Delta \sigma_{ij}, \beta \Delta w_{jk}]$. To improve convergence rate, it is important to find the optimal learning rate. The following Theorem 3 will show the optimal learning rate of neuro-fuzzy systems.
Theorem 3: The optimal learning rate $\beta_{\text{opt}}$ of the consequent part in neuro-fuzzy systems can be obtained as

$$
\beta_{\text{opt}} = \frac{\sum_{p=1}^{P} \sum_{k=1}^{K} (y_k^p(\alpha) - d_k^p) \Delta y_k(\alpha)}{\sum_{p=1}^{P} \sum_{k=1}^{K} \Delta y_k(\alpha) \Delta y_k(\alpha)}.
$$

(25)

where $\Delta y_k(\alpha) = \sum_{j=1}^{L} \mu_j(\alpha) \Delta w_{jp}/s(\alpha)$, and the near optimal learning rate $\alpha_{\text{opt}}$ of the premise part in neuro-fuzzy systems can be obtained

$$
\alpha_{\text{opt}} = -\frac{\frac{\partial E(\alpha)}{\partial \alpha}}{\frac{\partial^2 E(\alpha)}{\partial \alpha^2}} |_{\alpha=0}.
$$

(26)

Proof: From (8), we have

$$
E(W) = \frac{1}{2P} \sum_{p=1}^{P} \sum_{k=1}^{M} (y_k^p - d_k^p)^2 \geq 0.
$$

In F-CONFIS, given the MFs are continuously differentiable and provided that learning rates are greater than zero, the error of the gradient-descent learning process will be decreased. Taking error gradient with respect to $(\alpha, \beta)$ is zero, and then the near optimal learning rates satisfy

$$
\frac{\partial E(\alpha, \beta)}{\partial \alpha} |_{\alpha=\alpha_{\text{opt}}} = 0, \quad \frac{\partial E(\alpha, \beta)}{\partial \beta} |_{\beta=\beta_{\text{opt}}} = 0.
$$

(27)

From (7), (8), and (22), we get

$$
\Delta w_{jk} = \frac{1}{PM} \sum_{p=1}^{P} \sum_{k=1}^{M} \frac{(y_k^p - d_k^p) \mu_j}{s}.
$$

(28)

Denote $\Delta y_k(\alpha) = \sum_{j=1}^{L} \mu_j(\alpha) \Delta w_{jk}/s(\alpha)$, from (7), we have

$$
y_k(\alpha, \beta) = \frac{\sum_{j=1}^{L} \mu_j(\alpha) w_{jk}(\beta)}{s(\alpha)} = \frac{\sum_{j=1}^{L} \mu_j(\alpha) (w_{jk} - \beta \Delta w_{jk})}{s(\alpha)} = y_k(\alpha) - \beta \Delta y_k(\alpha).
$$

(29)

Differentiating (8) with respect to $\beta$, we get

$$
\frac{\partial E(\alpha, \beta)}{\partial \beta} = \frac{1}{P} \sum_{p=1}^{P} \sum_{k=1}^{K} (y_k^p(\alpha, \beta) - d_k^p) \frac{\partial y_k^p(\alpha, \beta)}{\partial \beta}.
$$

(30)

From (29) and (30), we have

$$
\frac{\partial E(\alpha, \beta)}{\partial \beta} = -\frac{1}{P} \sum_{p=1}^{P} \sum_{k=1}^{K} (y_k^p(\alpha) - \beta \Delta y_p(\alpha) - d_k^p) \Delta y_k(\alpha).
$$

(31)

From (27) and (31), we get the optimal learning rate $\beta_{\text{opt}}$ of the consequent part

$$
\beta_{\text{opt}} = \frac{\sum_{p=1}^{P} \sum_{k=1}^{K} (y_k^p(\alpha) - d_k^p) \Delta y_k(\alpha)}{\sum_{p=1}^{P} \sum_{k=1}^{K} \Delta y_k(\alpha) \Delta y_k(\alpha)}.
$$

Although the optimal learning rate $\beta_{\text{opt}}$ of the consequent part can be obtained, unfortunately, the analytical expression of the premise part cannot be solved in a similar way because the premise part, $\frac{\partial E(\alpha, \beta)}{\partial \alpha} = 0$, is usually a transcendental equation. To obtain $\alpha_{\text{opt}}$, line search method [31] is feasible. The simplest way for performing a line search is to take a series of small steps along the chosen search direction until the error increases, and then go back one step. Though there are other better approaches for performing inexact line searches such as Armijo line search [32], [33], it is still time consuming. In this paper, we propose a method to get the near optimal learning rate for $\alpha$ by expanding $E(\alpha, \beta)$ with Taylor series. To avoid the complicity of multivariable Taylor series, we let the weights of consequent part $w_{jk}$ stay the same (that is to let $\beta = 0$) while deriving the learning rate of the premise part. Taylor series of $E(\alpha) = E(\alpha, \beta)$ to second order around point 0 is

$$
E(\alpha) = E(0) + \alpha \frac{\partial E(\alpha)}{\partial \alpha} |_{\alpha=0} + \frac{\alpha^2}{2} \frac{\partial^2 E(\alpha)}{\partial \alpha^2} |_{\alpha=0} + O(\alpha^3).
$$

(32)

If we neglect the higher order terms and differentiate both side of (32) with respect to $\alpha$, we have

$$
\frac{\partial E(\alpha)}{\partial \alpha} |_{\alpha=0} + \alpha \frac{\partial^2 E(\alpha)}{\partial \alpha^2} |_{\alpha=0} = 0.
$$

(33)

From (27) and (33), we get the near optimal learning rate $\alpha_{\text{opt}}$ of the consequent part

$$
\alpha_{\text{opt}} = -\frac{\frac{\partial E(\alpha)}{\partial \alpha} |_{\alpha=0}}{\frac{\partial^2 E(\alpha)}{\partial \alpha^2} |_{\alpha=0}}.
$$

Q.E.D.

Next, we need to derive $\frac{\partial E(\alpha)}{\partial \alpha}$ and $\frac{\partial^2 E(\alpha)}{\partial \alpha^2}$. Let

$$
e_k^p = \frac{1}{2} (y_k^p - d_k^p)^2.
$$

(34)

Then

$$
E = \frac{1}{P} \sum_{p=1}^{P} \sum_{k=1}^{M} e_k^p
$$

(35)

$$
\begin{align*}
\frac{\partial E(\alpha)}{\partial \alpha} &= \frac{1}{P} \sum_{p=1}^{P} \sum_{k=1}^{M} \frac{\partial e_k^p(\alpha)}{\partial \alpha} \\
\frac{\partial^2 E(\alpha)}{\partial \alpha^2} &= \frac{1}{P} \sum_{p=1}^{P} \sum_{k=1}^{M} \frac{\partial^2 e_k^p(\alpha)}{\partial \alpha^2}.
\end{align*}
$$

(36)
For conciseness, we use \( \partial e_k(\alpha)/\partial \alpha \) and \( \partial^2 e_k(\alpha)/\partial \alpha^2 \) to express \( \partial e_k^p(\alpha)/\partial \alpha \) and \( \partial^2 e_k^p(\alpha)/\partial \alpha^2 \) corresponding to any \( p \)th training pattern.

From (34), we get
\[
\frac{\partial e_k(\alpha)}{\partial \alpha} = (y_k(\alpha) - d_k) \frac{\partial y_k(\alpha)}{\partial \alpha}.
\] (37)

Differentiating (37) with respect to \( \alpha \), we get
\[
\frac{\partial^2 e_k(\alpha)}{\partial \alpha^2} = \left( \frac{\partial y_k(\alpha)}{\partial \alpha} \right)^2 + (y_k(\alpha) - d_k) \frac{\partial^2 y_k(\alpha)}{\partial \alpha^2}.
\] (38)

Differentiating (7) with respect to \( \alpha \), we get
\[
\frac{\partial y_k(\alpha)}{\partial \alpha} = \sum \frac{\partial (\mu_j(\alpha) w_{jk})}{\partial \alpha} - \sum \frac{\partial (\mu_j(\alpha))}{\partial \alpha} y_k(\alpha).
\] (39)

Differentiating (39) with respect to \( \alpha \), we have
\[
\frac{\partial^2 y_k(\alpha)}{\partial \alpha^2} = \sum w_{jk} \frac{\partial^2 \mu_j(\alpha)}{\partial \alpha^2} - y_k(\alpha) \sum \frac{\partial^2 (\mu_j(\alpha))}{\partial \alpha^2} - 2 \frac{\partial y_k(\alpha)}{\partial \alpha} \sum \frac{\partial (\mu_j(\alpha))}{\partial \alpha}.
\] (40)

Form (1) and (9), we have
\[
\mu_j(\alpha) = \prod_{i=1}^{n} A_{ij}(\alpha) = \exp \left( - \sum_{i=1}^{n} \left( \frac{(x_i - m_{ij} + a \Delta m_{ij})^2}{2(\sigma_{ij} - a \sigma_{ij})^2} \right) \right).
\] (41)

Differentiating (41) with respect to \( \alpha \), we have
\[
\frac{\partial \mu_j(\alpha)}{\partial \alpha} = -\mu_j(\alpha) \sum_{i=1}^{n} \left( \frac{(x_i - m_{ij}(\alpha)) \Delta m_{ij}}{(\sigma_{ij}(\alpha))^2} + \frac{(x_i - m_{ij}(\alpha))^2 \Delta \sigma_{ij}}{(\sigma_{ij}(\alpha))^3} \right).
\] (42)

Differentiating (42) with respect to \( \alpha \), we have
\[
\frac{\partial^2 \mu_j(\alpha)}{\partial \alpha^2} = -\frac{\partial \mu_j(\alpha)}{\partial \alpha} \sum_{i=1}^{n} \left( \frac{(x_i - m_{ij}(\alpha))(\sigma_{ij}(\alpha) \Delta m_{ij} + (x_i - m_{ij}(\alpha)) \Delta \sigma_{ij})}{(\sigma_{ij}(\alpha))^3} - \mu_j(\alpha) h(\alpha) \right)
\] (43)

where \( h(\alpha) \) is
\[
\frac{\partial}{\partial \alpha} \sum_{i=1}^{n} \left( \frac{(x_i - m_{ij}(\alpha))(\sigma_{ij}(\alpha) \Delta m_{ij} + (x_i - m_{ij}(\alpha)) \Delta \sigma_{ij})}{(\sigma_{ij}(\alpha))^3} - \mu_j(\alpha) h(\alpha) \right)
\]
\[
= \sum_{i=1}^{n} \left( \frac{(\Delta m_{ij})^2}{(\sigma_{ij}(\alpha))^2} + \frac{4(x_i - m_{ij}(\alpha))(\Delta m_{ij} \Delta \sigma_{ij})}{(\sigma_{ij}(\alpha))^3} + \frac{3(x_i - m_{ij}(\alpha))^2(\Delta \sigma_{ij})^2}{(\sigma_{ij}(\alpha))^4} \right).
\]

By (42) and (43), we can calculate \( \partial \mu_j(\alpha)/\partial \alpha \) and \( \partial^2 \mu_j(\alpha)/\partial \alpha^2 \). With \( \partial \mu_j(\alpha)/\partial \alpha \) and \( \partial^2 \mu_j(\alpha)/\partial \alpha^2 \), by (39) and (40), we can calculate \( \partial y_k(\alpha)/\partial \alpha \) and \( \partial^2 y_k(\alpha)/\partial \alpha^2 \).

With \( \partial y_k(\alpha)/\partial \alpha \) and \( \partial^2 y_k(\alpha)/\partial \alpha^2 \), by (37), we can calculate the gradient components of the repeated weights.

For each iteration, during the gradient descent process, the near optimal learning rate of the premise part \( \alpha_{opt} \) is first calculated via (26). With the \( \alpha_{opt} \), the control parameters of MFs are updated. Then, the optimal learning rate of the consequent part is calculated by (25). Therefore, according to Theorem 2, the components of \( W \) are finally updated by following equations:

\[
\begin{align*}
\lambda_{iq}(t+1) &= \lambda_{iq}(t) - \alpha_{opt} \\
&\quad \times \left( \sum_{i=1}^{\text{RW}(i)} \frac{\partial E}{\partial \lambda_{i}(\alpha)} \frac{\partial \mu_{i}(\alpha)}{\partial \lambda_{i}(\alpha)} \frac{\partial A_{iq}}{\partial \lambda_{i}(\alpha)} \right)
\end{align*}
\]

and \( \alpha_{opt} \), we can calculate \( \partial E/\partial \alpha \) and \( \partial^2 E/\partial \alpha^2 \). With \( \partial E/\partial \alpha \) and \( \partial^2 E/\partial \alpha^2 \), by (26), we can calculate \( \frac{\partial^2 E}{\partial \alpha^2} \). With \( \partial E/\partial \alpha \) and \( \partial^2 E/\partial \alpha^2 \), by (26), we can calculate \( \alpha_{opt} \). For each iteration, during the gradient descent process, the near optimal learning rate of the premise part \( \alpha_{opt} \) is first calculated via (26). With the \( \alpha_{opt} \), the control parameters of MFs are updated. Then, the optimal learning rate of the consequent part is calculated by (25). Therefore, according to Theorem 2, the components of \( W \) are finally updated by following equations:

\[
\begin{align*}
w_{jk}(t+1) &= w_{jk}(t) - \beta_{opt} \frac{\partial E}{\partial w_{jk}}
\end{align*}
\]
Algorithm 1 Optimal Learning of Neuro-Fuzzy System via F-CONFIS

Step 1: Initializing the weight vector, learning rates. Let \( t \) be iteration count, \( t = 1 \).

Step 2: Calculate the number of fuzzy rule \( L \) by (6), truth value \( \mu_j \) by (1), output \( y_k \) by (7), error \( E \) in (8), the repeated link weights \( RW(i) \) by (3), and the ordinal indices \( h_1(i, q) \) by Theorem 1.

Step 3: Calculates the update changes of the control parameters in the premise part i.e., the overall update changes of the centers and spreads of the Gaussian MFs.

1) Let hidden node number \( l = 0 \).
2) For training input data \( i = 1, \ldots, N \)
   a) By (13), get the ordinal index \( r_i(l) \) of MFs for the \( i \)th fuzzy variable.
   b) Find the gradient components \( \partial E/\partial m_{rij}(l) \) and \( \partial E/\partial \sigma_{rij}(l) \) in premise part via (21).
   c) Since every same weight \( A_{iq} \) will be repeated for \( RW(i) \) times, we have to sum the update changes with the same subscripts for \( i \)th fuzzy variable \( x_i \) for \( RW(i) \) times by (23).
3) If \( l \) less than to the total number of fuzzy rules \( L \), then \( l = l + 1 \), go to 2), else go to next step 4.

Step 4: Calculate the gradient components of consequent part by (22).

Step 5: Calculate the learning rate \( \alpha_{opt} \) of the premise part via (26) according to Theorem 3.

Step 6: Update \( m_{ij} \) and \( \sigma_{ij} \) by (45) according to Theorem 2.

Step 7: Calculate the learning rate \( \beta_{opt} \) of the consequent part by (25) according to Theorem 3.

Step 8: Update \( w_{jk} \) by (45) according to Theorem 2.

Step 9: Calculate error \( E \) in (8).

Step 10: If error \( E \) is less than error tolerance or iteration \( t \) reaches maximum iterations, go to step 11, else iteration \( t = t + 1 \); go to step 2.

Step 11: Stop.

The optimal learning rates of the premise part and the consequent part are given by Theorem 3. Next, an optimal learning algorithm for F-CONFIS will be described.

Fig. 7 describes the process of the proposed learning.

IV. DISCUSSION OF TIME COMPLEXITY AND DERIVATIVE-BASED METHOD

Assume there are \( n \) adjustable parameters, \( L \) fuzzy rules. The training procedure of FIS can be summarized as the following pseudo-code.

```
while(epochs < max_epochs) {
    For(i=0; i < L; i++) // forward pass of training algorithm
    Calculate_firing_strength_of_each_rule();
    Calculate_the_output_of_FIS();
    }
    if(error < tolerance) break;
    // backward pass of training algorithm
    Calculate_derivative();
    Update_the_parameters();
}
```

In one epoch, the time complexity for forward pass is \( O(L) \).

There are three cases for backward pass.

1) If only first-order derivative is used, then the time complexity is \( O(n) \).
2) If second-order derivative is used, then the time complexity is \( O(n^2) \).
3) If there are operation of inverse matrix [like Levenberg–Marquardt (LM) method or Gauss–Newton (QN) method], then the time complexity is \( O(n^3) \).

So, the overall time complexity for algorithm may be \( O[epochs*(L+n)] \), \( O[epochs*(L + n^2)] \), or \( O[epochs*(L+n^3)] \). ANFIS and our proposed method used only the first-order derivative, the time complexity is \( O(epochs*(L + n)) \). For a real application, the \( n \) and \( L \) are fixed, whereas epochs is variable. It usually holds that \( L < n < \text{epochs} \), so the dominated factor is epochs. The proposed approach applies dynamic optimal learning algorithm in premise and consequent parts of FIS, so it can obtain less iteration times than that of ANFIS, which leading to reduce of epochs and time complexity. Zhao–Li–Irwin’s method apply LM method \[34\], LM method contains matrix inverse operation, so its time complexity is \( O(epochs*(L+n^3)) \). Theoretically, LM method can achieve less iteration times than gradient-descent method, but \( n^3 \) item in time complexity counteract this advantage.

The derivative-based optimization can be written as

\[
W_{k+1} = W_k - \eta_k B_k \nabla E_k.
\]  

\[
\begin{align*}
    m_{iq}(t + 1) &= m_{iq}(t) - \alpha_{opt} \left( \frac{1}{p} \sum_{j=1}^{L} \sum_{k=1}^{M} \frac{\sum_{l=1}^{L} (y_{pl} - d_{pl}) (w_{jk} - y_{pl}) \mu_{h_1(i, q)} (x_{i} - m_{iq})}{\sum_{j=0}^{L} \mu_{j}} \right) \\
    \sigma_{iq}(t + 1) &= \sigma_{iq}(t) - \alpha_{opt} \left( \frac{1}{p} \sum_{j=1}^{L} \sum_{k=1}^{M} \frac{\sum_{l=0}^{L-1} (y_{pl} - d_{pl}) (w_{jk} - y_{pl}) \mu_{h_1(i, q)} (x_{i} - m_{iq})^2}{\sum_{j=0}^{L} \mu_{j}} \right) \\
    w_{jk}(t + 1) &= w_{jk}(t) - \beta_{opt} \left( \frac{1}{p} \sum_{j=0}^{L} \mu_{j} \frac{1}{\sum_{l=0}^{L-1} \mu_{j}} \right) \\
    i &= 1, \ldots, N, \quad j = 0, \ldots, L - 1, \quad k = 1, \ldots, M, \quad q = 0, \ldots, R_l - 1, \quad \lambda = 1, \ldots, RW(i).
\end{align*}
\]  

(45)
Fig. 8. Conventional method of genfis1.

Fig. 9. Conventional method of genfis2.

Fig. 10. Conventional method of genfis3.

When $B_k$ is identity matrix ($B_k = I_k$), it is gradient descent method. When $B_k$ is the inverse of Hessian matrix ($B_k = (\nabla^2 w_k)^{-1}$), it is Newton’s method. Since the inverse of Hessian matrix may not exist and the computation cost is high.

The quasi-Newton method is a practical alternative, in which $B_k$ is an approximation of the true inverse Hessian matrix. BFGS and DFP are two kinds of quasi-Newton method. Theoretically, if the object function is quadratic, the convergence rate of quasi-Newton method is superlinear. If we take the BFGS or DFP method, then we cannot get the optimal learning rate in closed form, and must resort to line search method. If the line search is inexact or not based on the Wolfe conditions, the $B_k$ will becomes a poor approximation to the true inverse Hessian and produce bad results. The possible exploration in dynamic learning rate of approximate second-order derivative methods like BFGS or DFP for a neuro-fuzzy system should be discouraged and can be regarded as a possible direction of future work.

V. ILLUSTRATED EXAMPLES

In practical application, choosing the proper learning rate is a time consuming process. Manually selected learning rate needs many experiments and the result may lack generality. The proposed optimal learning algorithm can adjust the learning rate dynamically. In this section, several examples are conducted to verify the effectiveness of the proposed training algorithms for the F-CONFIS, and compared with the results from some well-known methods. In our proposed approach, the values of premise parameters are initialized using fuzzy C-means clustering method, and the values of consequent parameters are initialized by least-squares method.

Example 1 (Comparison of the Performance of Well-Known Fuzzy Systems With Conventional Method and Our Proposed Method): Consider the approximation of the following 1-D nonlinear function [35]

$$y(x) = \frac{1}{1 + e^{-25(x-0.5)}} \quad 0 \leq x \leq 1$$

(47)

where $x$ is input and $y$ is output. In all, 400 data samples were generated, every sample is $(x(t), y(t))$. The first 200 data samples is used for the training process and remaining 200 for testing process. $x(t)$ uniformly distributed in the interval [0, 1] and $y(t)$ was given by

$$y(t) = f(x(t)) + \varepsilon(t) \quad 0 \leq x \leq 1$$

(48)

where $\varepsilon(t) \sim N(0, 0.04^2)$.

Example 1.1 (Performance Comparison Between ANFIS and F-CONFIS): For comparison purpose, fuzzy models were developed under the same condition using several well-known methods, including genfis1, genfis2, and genfis3 of ANFIS and F-CONFIS. The fuzzy models were optimized by the proposed method and the conventional algorithms of ANFIS, which are available in MATLAB toolbox.

After iteration reaches 100, Figs. 8–10 show the outputs of genfis1, genfis2, and genfis3 trained by conventional method of ANFIS. Fig. 11 shows the output of F-CONFIS trained by the proposed method. In each figure, top curve is the output (dashed line) of neuro-fuzzy systems and training data (solid line) and bottom curve is the prediction error between the output and original training data. It is obvious that proposed method achieves better accuracy and faster convenience speed.

Table II shows the performance (MSE) comparison of conventional methods and proposed method. Both conventional and the new optimal learning approaches were utilized in the example, then 10, 100, 200, 500, and 1000 iterations were conducted for the methods. It can be seen that when iteration reaches 10, the MSE of proposed method is 0.010832, the
error reduction is much better than that of other methods at iteration of 1000.

**Example 1.2 (Comparison of Conventional Algorithms and Our Proposed Algorithm):** For comparison purpose, the experiments were conducted under the same condition. ANFIS and F-CONFIS are trained using the conventional gradient descent algorithm with fixed learning rate and the proposed method respectively.

Table III shows the performance comparison of the proposed method and traditional gradient descent with fixed learning rate algorithm (Case a: $\alpha = 0.2, \beta = 0.1$; Case b: $\alpha = 0.5, \beta = 0.8$; Case c: $\alpha = 0, \beta = 0.6$; Case d: $\alpha = 0.4, \beta = 0.4$). From Table III, we can see that when iteration reaches 10, the proposed method almost converge to zero. The error reduction is much better than that of others method at iteration of 1000. The improvement of convergence rate is considerable and impressive.

**Example 2 (Nonlinear System Identification):** The second example is to identify a nonlinear system, which is described...
by the second-order difference equation [36]
\[ y(k + 1) = g[y(k), y(k - 1)] + u(k) \]  \hspace{1cm} (49)

where
\[ g[y(k), y(k - 1)] = \frac{y(k)y(k+1)[y(k) + 2.5]}{1 + y^2(k) + y^2(k-1)} \]  \hspace{1cm} (50)

A series-parallel neuro-fuzzy system identifier [18], [36] is given by
\[ \hat{y}(k + 1) = \hat{f}[y(k), y(k - 1)] + u(k) \]  \hspace{1cm} (51)

where \( \hat{f}[y(k), y(k - 1)] \) is the form of (7) with two fuzzy variables \( y(k) \) and \( y(k - 1) \). The 500 training samples are generated by the plant model using a random input signal \( u(k) \), which is distributed in \([-2, 2]\) and every fuzzy variable has five MFs. The MFs are the Gaussian functions. After the training process is finished, the model is tested by applying a sinusoidal input signal \( u(k) = \sin(2\pi k/25) \). The proposed optimal learning algorithm was applied to the F-CONFIS.

**Example 2.1 (Comparison of Fixed Learning Rates and Dynamic Learning Rate in F-CONFIS):** For comparison purpose, the experiments are conducted in F-CONFIS with fixed learning rates and dynamic learning rate.

After 500 iterations, the trajectories of error of first 10 iterations are shown in Fig. 14. It also shows the performance comparison with the Cases b–e, where the learning rates are fixed (Case a: proposed method; Case b: \( \alpha = 0.5, \beta = 0.8 \); Case c: \( \alpha = 0, \beta = 0.6 \); Case d: \( \alpha = 0.1, \beta = 0.2 \)). Apparently, the error of dynamic approach converges to zero faster than other cases.

Table V shows the comparison performance with fixed learning rate and the proposed method, where 10, 200, and 500 iterations \( (t) \) were conducted. It can be seen that the total squared error \( J \) of proposed method is 0.00074829 when iteration reaches 10, whereas the total square error \( J \) of other cases is close to this value till iteration reaches 200. Obviously, the proposed tuning algorithm for F-CONFIS is much better than other cases.

**Example 2.2 (Comparison of Conventional Optimal Learning Methods of ANFIS and the Proposed Algorithm of F-CONFIS):** To improve convergence rate, some kinds of dynamic learning rate methods [12], [16], [18]–[22] can be applied to the neuro-fuzzy systems. The learning rate of premise part was obtained by genetic search techniques in [18].

This example conducts the performance comparison between the conventional dynamic optimal learning method in [18] and our proposed method under the same condition. Table VI shows the performance comparison between conventional optimal learning methods of the fuzzy NN in [18] with the proposed method.

After five iterations and 120 iterations, the total squared error \( J \) of the proposed method is 0.003286 and 0.0004019 compared with 0.00719 and 0.0023 of method in [18]. It can be seen that the convergence speed of the proposed algorithm is dozens times faster than those of others. To show the accuracy of the proposed method, the output of F-CONFIS trained by proposed method is shown in Fig. 15, where the sign + denotes the F-CONFIS output.
Table VII

MSE COMPARISON OF DIFFERENT ALGORITHM

<table>
<thead>
<tr>
<th>Method</th>
<th>Epochs</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conventional gradient decent algorithm</td>
<td>1000</td>
<td>0.087277</td>
</tr>
<tr>
<td>Proposed optimal learning algorithm</td>
<td>200</td>
<td>0.077028</td>
</tr>
</tbody>
</table>

Example 3 (Gas Furnace Data With the New Algorithm):
The gas furnace data set is a commonly used benchmark. This data set has been used extensively as a benchmark example in neuro-fuzzy field [37], [38]. There are 296 input–output measurements. The input $u(t)$ is the flow rate of the methane gas and the output measurement $y(t)$ is the concentration of carbon dioxide in the gas mixture flowing out of the furnace under a steady air supply. In our experiment, the inputs are $u(t-4)$ and $y(t-1)$, with the output variable is $y(t)$. The F-CONFIS shown in Fig. 17 is equivalent to the original neuro-fuzzy system shown in Fig. 16, it has two input nodes and one output node, each input variable has five items, so there are 25 fuzzy rules. The learning rates for the conventional algorithm are set as $\alpha = 0.01$ and $\beta = 0.2$. The MFs are the Gaussian functions.

Table VII shows MSE comparison of between conventional gradient decent algorithm and the proposed method, the MSE of the conventional gradient decent algorithm reaches 0.087277 in 1000 epochs, but with the proposed dynamic optimal learning rates the MSE reaches 0.077028 in 200 epochs. The proposed dynamic learning rate approach converges much faster than the conventional gradient decent algorithm. It can be seen that the proposed tuning algorithm for F-CONFIS is very effective.

After 500 iterations, the trajectories of error of first five iterations are shown in Fig. 18. It also shows the performance comparison with other Cases b–e in which the learning rates are fixed. Table VIII shows the comparison performance of fixed learning rates and the proposed method in 50, 100, 200, and 500 iterations ($t$). Apparently, the convergence speed of proposed algorithm is much faster than those of other cases. Fig. 19 shows the output of F-CONFIS trained by the dynamic learning rates, where the solid line is the original data and the dot denotes the F-CONFIS output with the dynamic learning rate algorithm. It is shown that the proposed approach achieves excellent performance.
VI. CONCLUSION

The conventional four-layer neuro-fuzzy system is transformed into an equivalent fully connected three-layer feed-forward NN, or F-CONFIS. Because F-CONFIS is a variation of multilayer NNs and has dependent and repeated links, the derivation of the learning algorithm is carefully designed to cope with weights in these repeated links. In addition, the dynamic optimal learning of F-CONFIS is derived not only in the premise part, but also in the consequent part. The simulation results indicate the proposed dynamic learning improves the accuracy considerably and converges much fast.

APPENDIX

The following Lemma 1 shows the increment rule of ordinal indices for hidden neurons with links of repeated weights, which is needed to prove Theorem 1.

Lemma 1: Let \( h(i, q) \) be the ordinal indices for hidden neurons with links of repeated weights, which connect the \( q \)th MF of the \( i \)th fuzzy input variable \( x_i \). Then, we have

\[
\begin{align*}
    h_{i+1}(i, q) &= h_i(i, q) + R_is_i - s_i + 1, \quad \text{if } h_i(i, q) \% s_i = s_i - 1 \\
    h_{i+1}(i, q) &= h_i(i, q) + 1, \quad \text{otherwise}
\end{align*}
\]

where \( s_i \) is defined in (15).

Proof: We know that \( h_i(i, q) \) is a specific fuzzy rule number \( l \) in (17) for which we should have \( r_i(l) = q \), or

\[
h_i(i, q) = l(r_1(\lambda), r_2(\lambda), \ldots, r_{i-1}(\lambda), q, r_{i+1}(\lambda), \ldots, r_N(\lambda))
\]

\[
= \left( \sum_{n=1}^{i-1} r_n(\lambda)s_n + qs_i + \sum_{n=i+1}^N r_n(\lambda)s_n \right)
\]

\[
= \sum_{n=1}^{i-1} r_n(\lambda)s_n + qs_i
\]

\[
+ R_is_i (r_{i+1}(\lambda) + r_{i+2}(\lambda)R_{i+1} + \cdots + r_N(\lambda) \prod_{n=i+1}^{N-1} R_n).
\]

Let

\[
\begin{align*}
    hv(\lambda) &= hv(r_1(\lambda), r_2(\lambda), \ldots, r_{i-1}(\lambda)) = \sum_{n=1}^{i-1} r_n(\lambda)s_n \\
    hv(\lambda) &= hv(r_{i+1}(\lambda), r_{i+2}(\lambda), \ldots, r_N(\lambda)) = \frac{1}{R_is_i} \sum_{n=i+1}^N r_n(\lambda)s_n
\end{align*}
\]

\( hv(\lambda) = 0, \text{ if } i = 1; \; hv(\lambda) = 0, \text{ if } i = N \).

Then, (A.1) can be rewritten as

\[
h_{i+1}(i, q) = hv(\lambda) + qsi + R_is_i hv(\lambda).
\]  

(A.2)

The (A.1) and (A.2) are actually found from (17) so that all the \( h_i(i, q) \) in \( H = \{h_i(i, q)|\lambda = 1, \ldots, RW(i)\} \) are arranged in ascending order by the following mechanism.

1) Let \( (r_1, r_2, \ldots, r_{i-1}, r_{i+1}, \ldots, r_N) = (0, 0, \ldots, 0) \).

2) Increment the \( h_i(i, q) \) by increasing the \( (r_1, r_2, \ldots, r_{i-1}, r_{i+1}, \ldots, r_N) \) using the following sequence: \( (0, 0, \ldots, 0, 0, 0, \ldots, 0), (1, 0, \ldots, 0, 0, 0, 0, \ldots, 0), \ldots, (R_i - 1, 0, \ldots, 0, 0, 0, 0, \ldots, 0), \ldots, (R_i - 1, 0, \ldots, 0, 0, 0, 0, \ldots, 0), \ldots, (R_i - 1, 0, \ldots, 0, 0, 0, 0, \ldots, 0), \ldots, (R_i - 1, 0, \ldots, 0, 0, 0, 0, \ldots, 0), \ldots, (R_i - 1, 0, \ldots, 0, 0, 0, 0, \ldots, 0) \).

The reason for the above arrangement is quite obvious due to the fact that \( s_{i+1} > s_i \) in (A.1), and the only way to let \( h_{i+1}(i, q) \) in \( H = \{h_i(i, q)|k = 1, \ldots, RW(i)\} \) in ascending order is to increase the indices of \( r_i \) quicker than \( r_{i+1} \). Denote

\[
\begin{align*}
    lv(r_1, r_2, \ldots, r_{i-1}) &= \sum_{n=1}^{i-1} r_n s_n, \quad (hv(r_1, r_2, \ldots, r_{i-1}) = 0, \text{ if } i = 1) \\
    hv(r_{i+1}, \ldots, r_N) &= \frac{1}{s_{i+1}} \sum_{n=i+1}^N r_n s_n, \quad (hv(r_1, r_2, \ldots, r_{i-1}) = 0, \text{ if } i = n)
\end{align*}
\]

It is obvious that the minimum value of \( hv(r_1, r_2, \ldots, r_{i-1}) \) is

\[
lv_{\text{min}} = hv(0, 0, \ldots, 0) = \sum_{n=1}^{i-1} 0s_n = 0.
\]

The maximum value can be similarly found as

\[
\begin{align*}
    lv_{\text{max}} &= hv(R_1 - 1, R_2 - 1, \ldots, R_{i-1} - 1) \\
    &= \sum_{n=1}^{i-1} (R_n - 1)s_n = \sum_{n=1}^{i-1} (s_{n+1} - s_n) = s_i - s_1 = s_i - 1.
\end{align*}
\]

Then, we have

\[
0 \leq hv(\lambda) \leq s_i - 1. \quad \text{(A.3)}
\]

So, \( hv(\lambda) \) may take any natural number between 0 and \( s_i - 1 \). The minimum value of \( hv(r_{i+1}, \ldots, r_N) \) is

\[
\begin{align*}
    hv_{\text{min}} &= hv(0, 0, \ldots, 0) = 0 + 0R_{i+1} + \cdots + 0 \prod_{n=i+1}^{N-1} R_n = 0.
\end{align*}
\]
Its maximum value can be found as
\[
h_{\text{max}} = hv(R_{i+1}, R_{i+2}, \ldots, R_N) = \frac{1}{s_{i+1}} \sum_{n=i+1}^{N} (R_n - 1)s_n
\]
\[
= \frac{1}{s_{i+1}} \sum_{n=i+1}^{N} (s_{N+1} - s_n) = \frac{s_{N+1} - s_{i+1}}{s_{i+1}} = \frac{s_{N+1}}{s_{i+1}} - 1.
\]

Then, we have
\[
0 \leq hv(\lambda) \leq (s_{N+1}/s_{i+1}) - 1. \quad \text{(A.4)}
\]

So, \(hv(\lambda)\) may take any natural number between 0 and \((s_{N+1}/s_{i+1}) - 1\). From (A.2)–(A.4), for any natural number \(x\) in \([0, s_i - 1]\) and \(y\) in \([0, (s_{N+1}/s_{i+1}) - 1]\), we have
\[
(x + q_{s_i} + R_is_i y) \in H = [h_2(i, q)|\lambda = 1, \ldots, \text{RW}(i)].
\]

(A.5)

It is also important to say that we have to adopt the above ascending order mechanism for \([x y] = [hv(r_1, r_2, \ldots, r_{i-1})hv(r_{i+1}, \ldots, r_N)]\) so that the \(h_2(i, q)\) in \(H = [h_2(i, q)|\lambda = 1, \ldots, \text{RW}(i)]\) will be in ascending order. From (A.2) with the fact that \(0 \leq hv(\lambda) \leq s_i - 1\), we have

\[
h_2(i, q)q_{s_i} = hv(\lambda)s_i + 0 + 0 = hv(\lambda).
\]

The above (A.6) will be used to check if the maximum value of \(hv(\lambda)\) has been reached or not. From (A.2), we have
\[
h_{j+1}(i, q) - h_j(i, q) = hv(\lambda + 1) - hv(\lambda)
\]
\[
+ R_is_i[y - hv(\lambda)]
\]
\[
= x - lv(\lambda) + R_is_i[y - hv(\lambda)]. \quad \text{(A.7)}
\]

Since we have assumed that all the \(h_k(i, q)\) in \(H = [h_2(i, q)|\lambda = 1, \ldots, \text{RW}(i)]\) are arranged in ascending order in a manner explained in the above ascending order mechanism, therefore the \(x = hv(\lambda + 1)\) in (A.7) will be incremented by 1 from \(hv(\lambda)\) until it reaches its maximum, i.e., \(s_i - 1\). However, \(y\) will not be incremented in this stage, i.e., we let \(yhv(\lambda) = 0\). Therefore, we have the following Case 1, i.e., if \(h_2(i, q)q_{s_i} = hv(\lambda) < s_i - 1\).

Case 1: If \(h_2(i, q)q_{s_i} = hv(\lambda) < s_i - 1\), then
\[
h_{j+1}(i, q) - h_j(i, q) = x - lv(\lambda) + R_is_i[y - hv(\lambda)]
\]
\[
= 1 + 0 = 1.
\]

Then, if \(h_2(i, q)q_{s_i} = hv(\lambda) = s_i - 1\), then \(x = hv(\lambda + 1)\) must be reset to zero, so the \(y = hv(\lambda + 1)\) will be incremented from \(hv(\lambda)\) by 1 until it reaches its maximum value \((s_{N+1}/s_{i+1}) - 1\). So, we have the following Case 2, i.e., if \(h_2(i, q)q_{s_i} = hv(\lambda) = s_i - 1\).

Case 2: If \(h_2(i, q)q_{s_i} = hv(\lambda) = s_i - 1\), then we reset \(x\) to zero and let \(y = hv(\lambda) + 1\) to have
\[
h_{j+1}(i, q) - h_j(i, q) = x - lv(\lambda) + R_is_i[y - hv(\lambda)]
\]
\[
= 0 - lv(\lambda) + R_is_i = R_is_i - s_i + 1.
\]

After combining the above Cases 1 and 2, we have the following conclusion:
\[
\begin{align*}
h_{j+1}(i, q) &= h_j(i, q) + R_is_i - s_i + 1, \quad \text{if } h_2(i, q)q_{s_i} = s_i - 1 \\
h_{j+1}(i, q) &= h_j(i, q) + 1, \quad \text{otherwise.}
\end{align*}
\]

Q.E.D.

**Theorem 1**: The ordinal indices \(\{h_\lambda(i, q)|\lambda = 1, \ldots, \text{RW}(i)\}\) for hidden neurons with links of repeated weights, which connect the \(q\)th MF of the \(i\)th fuzzy input variable \(x_i\), can be found from the following equation:
\[
h_\lambda(i, q) = (\lambda - 1)q_{si} + q_{si} + [\lambda - 1]/s_iR_is_i
\]
\[
\lambda = 1, \ldots, \text{RW}(i).
\]

(A.8)

where \(A \% B\) is the remainder of the division of \(A\) over \(B\), and \(A/B\) is the quotient of the integer division of \(A\) over \(B\).

Proof: For the \(i\)th fuzzy input variable \(x_i\) by (3), we have
\[
\text{RW}(i) = \frac{\prod_{m=1}^{N} R_m}{R_i} = \frac{\prod_{m=1}^{t-1} R_m R_i \left( \prod_{m=i+1}^{N} R_m \right)}{R_i} = s_i \prod_{m=i+1}^{N} R_m.
\]

Therefore, we have \(1 \leq \lambda \leq s_i \prod_{m=i+1}^{N} R_m\), which will fall into one of the ranges \([[(t - 1)s_i + 1, s_i] | t = 1, \ldots, \prod_{m=i+1}^{N} R_m\) \((i = N, \text{then } t = 1 \text{ and } k \text{ ranges from 1 to } s_i)\). This can be illustrated by Fig. 20.

If we can prove (A.8) holds for all \(t\), then it accordingly holds for all \(\lambda\). We will prove this by mathematical induction.

**Base**: When \(t = 1\), \(\lambda\) will fall into the range of \([1, s_i]\). For \(\lambda = 1\), from (A.2) with the fact that \([r_1(1)r_2(1)\ldots r_{i-1}(1)r_{i+1}(1)\ldots r_N(1)] = [00000000]\)
\[
h_{\lambda=1}(i, q) = 0 + q_{si} + 0 = 0 + q_{si} + (\lambda - 1).
\]

This will imply \(h_1(i, q)q_{s_i} = 0\). Therefore, according to Lemma 1, \(h_2(i, q) = h_1(i, q) + 1 = q_{si} + 1\). This process can be continued as follows:
\[
h_1(i, q)q_{s_i} = 0 \Rightarrow h_2(i, q) = h_1(i, q) + 1 = q_{si} + 1;
\]
\[
h_2(i, q)q_{s_i} = 1 \Rightarrow h_3(i, q) = h_2(i, q) + 1 = q_{si} + 2;
\]
\[
\ldots
\]
\[
h_{s_i-2}(i, q)q_{s_i} = 1 \Rightarrow h_{s_i-1}(i, q) = q_{si} + s_i - 2;
\]
\[
h_{s_i-1}(i, q)q_{s_i} = 1 \Rightarrow h_{s_i-1}(i, q) = q_{si} + s_i - 1.
\]

So that we have \(h_j(i, q)q_{s_i} = \lambda - 1, \text{if } 1 \leq \lambda \leq s_i\)
\[
h_\lambda(i, q) = q_{si} + \lambda - 1, \quad 1 \leq \lambda \leq s_i.
\]

(A.9)
For $1 \leq \lambda \leq s_i$, we have $(\lambda - 1)/s_i = 0$ and $(\lambda - 1) = (\lambda - 1)/s_i$. From (A.9), we have
\[
h_{\lambda}(i, q) = g_{s_i} + \lambda - 1 = [(\lambda - 1)/s_i]R_{s_i} + q_{s_i} + (\lambda - 1)/s_i, \quad 1 \leq \lambda \leq s_i.
\]
So (A.8) holds for $t = 1$.

**Induction Hypothesis:** Assume (A.8) holds for some specified $t_0$, then we have
\[
h_{\lambda}(i, q) = [(\lambda - 1)/s_i]R_{s_i} + q_{s_i} + (\lambda - 1)/s_i = (t_0 - 1)s_i + 1 \leq \lambda \leq t_0 s_i.
\]

**Inductive Step:** The last value of $\{h_{\lambda}(i, q)\}$ $(t_0 - 1)s_i + 1 \leq \lambda \leq t_0 s_i$ is
\[
h_{\lambda}(i, q) = [(t_0 s_i - 1)/s_i]R_{s_i} + q_{s_i} + (t_0 s_i - 1)/s_i
\]
\[= [(t_0 - 1)s_i + s_i - 1)/s_i]R_{s_i} + q_{s_i} + [(t_0 - 1)s_i + s_i - 1]/s_i
\]
\[= (t_0 - 1)s_i + q_{s_i} + s_i - 1.
\]
From (A.11), we have $h_{\lambda}(t_0 s_i + 1, i, q) = s_i - 1$. According to the Lemma 1, we have
\[
h_{\lambda}(i, q) = h_{\lambda}(t_0 s_i + 1, i, q) + R_{s_i} - s_i + 1
\]
\[= (t_0 - 1)s_i + q_{s_i} + s_i - 1 + R_{s_i} - s_i + 1
\]
\[= t_0 R_{s_i} + s_i.
\]
For $1 \leq \lambda \leq s_i + 1$, we have $(\lambda - 1)/s_i = t_0$ and $(\lambda - 1)/s_i = 0$. From (A.12), we have
\[
h_{\lambda}(i, q) = t_0 R_{s_i} + q_{s_i}
\]
\[= [(\lambda - 1)/s_i]R_{s_i} + q_{s_i} + (\lambda - 1)/s_i
\]
\[= t_0 s_i + 1 \leq \lambda \leq (t_0 + 1)s_i.
\]
From (A.13), $h_{\lambda}(t_0 s_i + 1, i, q)/s_i = 0$. Therefore, the next $s_i - 1$ items of $\{h_{\lambda}(i, q)\}$ will have the property of $h_{\lambda}(i, q) < s_i - 1/s_i$. Therefore, we have
\[
h_{\lambda}(i, q) = t_0 R_{s_i} + q_{s_i} + (\lambda - 1)/s_i
\]
\[= t_0 s_i + 1 \leq \lambda \leq (t_0 + 1)s_i.
\]
For $t_0 s_i + 1 \leq \lambda \leq (t_0 + 1)s_i$, we have $(\lambda - 1)/s_i = t_0$. Therefore, we can combine (A.13) and (A.14) to have
\[
h_{\lambda}(i, q) = [(\lambda - 1)/s_i]R_{s_i} + q_{s_i} + (\lambda - 1)/s_i
\]
\[t_0 s_i + 1 \leq \lambda \leq (t_0 + 1)s_i.
\]
Thereby, (A.8) holds for $t_0 + 1$.

Q.E.D.


