An Adaptive Neural Fuzzy Filter and Its Applications

Chin-Teng Lin, *Member, IEEE*, and Chia-Feng Juang

***Abstract*— A new kind of nonlinear adaptive filter, the adaptive neural fuzzy filter (ANFF), based upon a neural network's learning ability and fuzzy if-then rule structure, is proposed in this paper. The ANFF is inherently a feedforward multilayered connectionist network which can learn by itself according to numerical training data or expert knowledge represented by fuzzy if-then rules. The adaptation here includes the construction of fuzzy if-then rules (structure learning), and the tuning of the free parameters of membership functions (parameter learning). In the structure learning phase, fuzzy rules are found based on the matching of input–output clusters. In the parameter learning phase, a backpropagation-like adaptation algorithm is developed to minimize the output error. There are no hidden nodes (i.e., no membership functions and fuzzy rules) initially, and both the structure learning and parameter learning are performed concurrently as the adaptation proceeds. However, if some linguistic information about the design of the filter is available, such knowledge can be put into the ANFF to form an initial structure with hidden nodes. Two major advantages of the ANFF can thus be seen: 1) *a priori* knowledge can be incorporated into the ANFF which makes the fusion of numerical data and linguistic information in the filter possible; and 2) no predetermination, like the number of hidden nodes, must be given, since the ANFF can find its optimal structure and parameters automatically. Moreover, in contrast to traditional fuzzy systems where the input-output spaces are partitioned as grid type causing the combinatorial growing of fuzzy rules as the input–output dimensions increase, irregular partitions are done in the ANFF according to the distribution of training data so fewer fuzzy rules will be generated. To demonstrate the performance of the ANFF, two applications, the nonlinear channel equalization and the adaptive noise cancellation, are simulated. Efficiency and advantages of the ANFF are verified by these simulations and comparisons.**

I. Introduction

ADAPTIVE filtering has achieved widespread applications and success in many areas such as control, image processing, and communications [1]. Among the various adaptive filters, the adaptive linear filter is the most widely used one mainly due to its low hardware implementation cost and its properties, like the convergence, global minimum, misadjustment error and training algorithms, and can be easily analyzed and derived. The adaptive linear filtering has achieved a large amount of success in many situations. But for some situations where nonlinear phenomenon appear, the performance of linear filters have been poor [2]–[5], and the development of nonlinear filters is thus necessary.

Neural networks are composed of a large number of highly interconnected processing elements (nodes) which are connected through the weights. When looking into the structure and learning of neural networks, many common points to the methods used in adaptive signal processing can be found. For example, both of them have the adaptive linear combiner (ALC) properties in common [10]. Also, the backpropagation algorithm used to train the neural-network is in fact a generalized Widrow's least mean square (LMS) algorithm and can be contrasted to the LMS algorithm usually used in adaptive filtering. Characterized with these common points and the powerful learning and generalization ability, the neural network is now becoming an attractive candidate in adaptive signal processing. A problem encountered in the design of neural filters is that the internal layers of neural networks are always opaque to the user, so it is not easy to determine the structure and size of a network. To encode the input–output relationship into the neural network, repeated learning cycles must be performed and will take a lot of learning time. In a nonstationary environment, to adapt themselves to the statistical changes in the environment, the neural filters' adaptation will drag the weights away from their estimates of the previous environment, and knowledge forgetting then happens. The inconvenience of incorporating linguistic information expressed as fuzzy if-then rules in the design of neural filters is yet another shortcoming. To overcome the shortcomings encountered in neural filters, while still keeping their advantages, an adaptive neural fuzzy filter (ANFF) is developed in this paper.

The practical application of expert knowledge to solve real-world problems has received increasing attention. When we are constructing information processing systems, like the filters, the available information usually comes in two forms: numerical and linguistic. Most often, when we are designing filters, we use these two forms of information separately. We use the linguistic information for the choice of the most suitable kind of filter in application or the order of filters, etc. [26], while for the training of the filter we use numerical information only. The design of neural filters is such an example. Since filters can be considered as mapping functions, noisy inputs are mapped onto clean outputs. Very often, the available linguistic information is about this input–output relationship and is usually expressed as fuzzy if-then rules. For example, if we know that the high amplitude vicinity of the noisy input should be mapped by the filter onto the low amplitude vicinity of the output, we express it as "if input is high then output is low." For this reason, a good filter should be able to learn from such kind of information.

The ANFF is a feedforward multilayer network that integrates the basic elements and functions of a traditional fuzzy system into a connectionist structure. In this connectionist

Manuscript received November 9, 1995; revised June 14, 1996.

The authors are with the Department of Control Engineering, National Chiao-Tung University, Hsinchu 300, Taiwan, R.O.C. (e-mail: ctlin@fnn.cn.nctu.edu.tw).

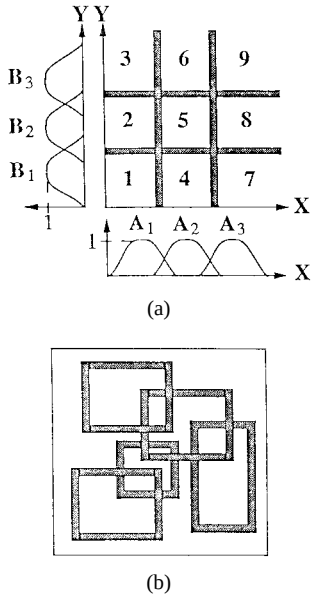
Publisher Item Identifier S 1083-4419(97)03881-8.

structure, the input and output nodes represent the corrupted signal process and desired signal process, respectively, and, in the hidden layers, there are nodes function as membership functions (activation functions) and fuzzy logic rules (connection types). An important feature of the proposed adaptive filter is that it can dynamically partition the input space and output space using irregular fuzzy hyperboxes according to training pattern distribution [26]. This irregular space-partitioning method is more flexible and can avoid combinatorial growth of partitioned grids in [26]. Fig. 1(a) and (b) show the grid-type partitions and the proposed partitioning method in the ANFF. The problem of space partitioning from numerical training data is basically a clustering problem. The proposed ANFF applies the Fuzzy Adaptive Resonance Theory (Fuzzy ART) proposed by Carpenter *et al.* [30], [31] to do fuzzy clustering in the input–output spaces and find proper fuzzy logic rules dynamically by associating input clusters with output clusters. For the adaptation of membership functions in the ANFF, the backpropagation algorithm is used to find the optimal parameters under the mean square error (MSE) criterion. Hence, in the ANFF, the Fuzzy ART is used for structure learning and the backpropagation algorithm for parameter learning. The ANFF can thus on-line partition the input–output spaces, tune membership functions, and find proper fuzzy logic rules dynamically on the fly. Users need not give the initial fuzzy partitions, membership functions, or fuzzy logic rules except for the case that expert knowledge is available and is used as the initial fuzzy rules. Hence, there are no hidden nodes in the beginning of learning; they are created and begin to grow as the training signal arrives. Since the structure of the ANFF is constructed from fuzzy if-then rules, once the input–output relationship is constructed, it will not be destroyed and, thus, no knowledge forgetting may happen. As the statistics of the environment change, the ANFF can automatically add new nodes to cope with the change and thus their estimates of the previous environment are still kept. These properties make the ANFF more suitable for on-line operation than the neural-network-type filters.

This paper is organized as follows. Section II discusses previous work about other adaptive filters, neural networks, fuzzy systems, and neural fuzzy networks. Section III describes the basic structure and functions of the ANFF. The on-line structure/parameter learning algorithm of the ANFF, which combines fuzzy ART and backpropagation learning algorithm under the MSE criterion is presented in Section IV. In Section V, the ANFF is applied to the nonlinear channel equalization problem and adaptive noise cancellation problem. Finally, conclusions are summarized in the last section.

II. Previous Work

Over the past two decades, many kinds of nonlinear filters designed using a nonadaptive approach have been proposed [6]. These include the class of filters based upon order statistics (e.g., L-filters, median filters, and α-trimmed mean filters), and the polynominal filters which are based on the Volterra series and Wiener series, etc. These filters perform well only when the statistics of signal and noise processes are known in advance or only for a special kind of noise. In real situations, the input data are usually nonstationary, and even for the stationary case, their statistics may not be available. Under these circumstances, the above filters perform poorly. Moreover, as the order of the filter increases, greater complexity in design occurs. Because of the two main drawbacks which appeared in the direct design approach, adaptive design method of nonlinear filters is required.



(a)

(b)

Fig. 1. (a) Grid-type fuzzy partitioning and (b) flexible hyperbox fuzzy partitioning.

Most of the proposed adaptive nonlinear filters are the adaptation versions of existing nonlinear filters. A major class is the adaptive stack filter [8] which provides an adaptive design method to existing generalized stack filters defined based upon threshold decomposition and Boolean operators, like the rank-order filters, morphological filters, stack filters and median filters, etc. [9]. The adaptive stack filter can solve both the problems of lacking statistics knowledge and the computation complexity in direct design; however, it is constrained to be applied to the situations when the threshold levels are small. Another example is the adaptive Volterra filter [7]. Volterra filters are linear combinations of order stochastics and adaptive Volterra filters enable them to tune the combination coefficients adaptively when the signal or noise statistics change. Since adaptive Volterra filters are inherently Volterra filters, they are also constrained to be applied to the class of nonlinear systems that can be represented by the Volterra series expansion. Basically, the structures of the above adaptive nonlinear filters are the same as those existing nonadaptive filters, and many assumptions are made in the derivation of these filters, which makes the suitability of their application limited. Since the performance of such kind of adaptive nonlinear filters is application oriented, it is difficult to say which one is dominantly better than others. The development of new adaptive nonlinear filters that can be applied under arbitrary situations is thus necessary. The universal approximation ability of neural and fuzzy networks makes them suitable for this requirement.

Many kinds of nonlinear filters designed using neural networks have been proposed. One of them is the neural filter, whose learning algorithm is shown to be more efficient than Lin's adaptive stack filtering algorithm [16]. This class of neural filters is based on the threshold decomposition and neural networks, and are divided into hard neural filters (whose activation functions are unit steps) and soft neural filters (whose activation functions are sigmoid functions). Another kind of neural filter is the recursive filter obtained by training a recurrent multilayer perception (RMLP) [17]. Other applications of neural networks in the adaptive filtering include nonlinear channel equalizers [13], [15] and the noisy speech recognition [5], [11], [12] where neural networks are used to map the noisy input features into clean output features for recognition. Common disadvantages of these neural filters are discussed in Section I.

A fuzzy system is composed of a bunch of fuzzy if-then rules. Conventionally, the selection of fuzzy if-then rules often relies on a substantial amount of heuristic observation to express proper strategy's knowledge. Obviously, it is difficult for human experts to examine all the input–output data from a complex system to find the suitable number of rules within the fuzzy systems. For this reason, a fuzzy system with neural network's learning ability is required.

To enable a neural network to learn from numerical data as well as expert knowledge expressed as fuzzy if-then rules, several approaches have been proposed [18]–[25]. The neural fuzzy network [21]–[24], which is inherently a fuzzy logic system embedded with neural network's learning ability, is one of them. Generally, two phases of learning, structure and parameter, are performed sequentially to construct neural fuzzy networks. First, the structure learning is employed to construct the rules, and then the parameter learning is performed to tune the free parameters of each rule. This sequential learning scheme makes these networks suitable only for off-line operation, not for on-line operation. Another type of network is the fuzzy neural network proposed in [25]. This network is inherently a neural network being able to learn from fuzzy input–output pairs. To encode a fuzzy if-then rule into the neural network, repeated training is required, which is time consuming and the learned fuzzy if-then relationship may be destroyed for on-line learning. The other type of neural network is the expert network [18]–[20] that combines neural network with symbolic method. The shortcoming of this method is that to encode an expert knowledge into the network, the number of nodes required is large, and the input–output relation expressed is crisp and not fuzzy.

The application of an adaptive fuzzy network as a filter can be found in [26]. Even though this filter can make use of both linguistic and numerical information in their natural form, some drawbacks of this structure can still be seen. For their recursive-least-squares (RLS)-type fuzzy adaptive filters, the input spaces are partitioned in the grid type as shown in Fig. 1(a). The RLS-type fuzzy adaptive filters do have high convergent speed, but input spaces partitioned as this type have a serious problem: the number of fuzzy subspaces increases exponentially as the dimension of input spaces increases and will cause a high computation load and high memory requirement. As to the least-mean-squares (LMS)-type fuzzy adaptive filters in [26], the number of fuzzy rules should be decided in advance and are initially assigned arbitrarily. In real situations, the proper number of fuzzy rules are not easy to decide and arbitrary assignment of the initial rules will restrain the learning speed. As shown in the following sections, these shortcomings are solved by the proposed ANFF, with the original advantages of traditional adaptive fuzzy systems kept.

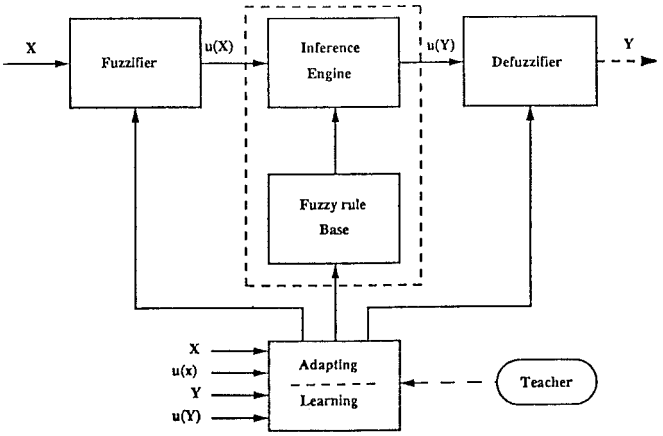


Fig. 2. Functional diagram of a fuzzy system.

III. The Structure of ANFF

The universal approximation ability of some fuzzy systems has been proven [27], [28]. Theoretically, any kind of filters, linear or nonlinear, can be approximated with these fuzzy systems. The structure and basic components of a conventional fuzzy system will be briefly introduced (for more details, please refer to [37] and [38]), and then the structure of the ANFF is proposed.

A. Basic Structure of a Fuzzy System

Fig. 2 shows the basic structure of a conventional fuzzy system with a learning/adapting component. Before proceeding, we must define some important terms. A *fuzzy set* F in a universe of discourse U is characterized by a membership function μ_F: $U \rightarrow [0,1]$. Thus, a fuzzy set F in U may be represented as a set of ordered pairs. Each pair consists of a generic element u and its grade of membership function $\mu_F(u)$, that is, $F = \{(u, \mu_F(u)) | u \in U\}$. u is called a *support value* if $\mu_F(u) > 0$. If U is a continuous universe and F is normal and convex (i.e., $\max_{u \in U} \mu_F(u) = 1$ and $\mu_F(\lambda u_1 + (1-\lambda)u_2) \geq \min(\mu_F(u_1), \mu_F(u_2)), u_1, u_2 \in U, \lambda \in [0,1]$), then F is a *fuzzy number*. A *linguistic variable* x in a universe of discourse U is characterized by

$$T(x) = \{T_x^1, T_x^2, \cdots, T_x^k\}$$

and

$$M(x) = \{M_x^1, M_x^2, \cdots, M_x^k\}$$

where $T(x)$ is the *term set* of x, that is, the set of names of linguistic values of x with each value T_x^i being a fuzzy number with membership function M_x^i defined on U. So $M(x)$ is a semantic rule for associating with each value its meaning.

For example, if x indicates speed, then $T(x)$ may be {*slow, medium, fast*}. Following the above definition, the input vector X which includes the input state linguistic variables x_i's, and the output state vector Y which includes the output state linguistic variables y_i's in Fig. 2 can be defined as

$$X = \{(x_i, U_i, \{T_{x_i}^1, T_{x_i}^2, \cdots, T_{x_i}^{k_i}\}, \{M_{x_i}^1, M_{x_i}^2, \cdots, M_{x_i}^{k_i}\})|_{i=1,\cdots,n}\} \quad (1)$$

$$Y = \{(y_i, U_i', \{T_{y_i}^1, T_{y_i}^2, \cdots, T_{y_i}^{l_i}\}, \{M_{y_i}^1, M_{y_i}^2, \cdots, M_{y_i}^{l_i}\})|_{i=1,\cdots,m}\}. \quad (2)$$

The fuzzifier in Fig. 2 is a mapping from an observed input space to fuzzy sets in certain input universe of discourse. So a specific value $x_i(t)$ at time t is mapped to the fuzzy set $T_{x_i}^1$ with degree $M_{x_i}^1(x_i(t))$ and to the fuzzy set $T_{x_i}^2$ with degree $M_{x_i}^2(x_i(t))$, and so on.

The fuzzy rule base in Fig. 2 contains a set of fuzzy logic rules R. For a multi-input and multi-output (MIMO) system, we have

$$R = \{R_{\text{MIMO}}^1, R_{\text{MIMO}}^2, \cdots, R_{\text{MIMO}}^n\}$$

where the ith fuzzy logic rule is

$$R_{\text{MIMO}}^i\text{: IF}(x_1 \text{ is } T_{x_1} \text{ and} \cdots \text{and } x_p \text{ is } T_{x_p}) \text{ THEN}(y_1 \text{ is } T_{y_1} \text{ and} \cdots \text{and } y_q \text{ is } T_{y_q}).$$

The preconditions of R_{MIMO}^i form a fuzzy set $T_{x_1} \times \cdots \times T_{x_p}$ and the consequent of R_{MIMO}^i is the union of q independent outputs. So the rule can be represented by a fuzzy implication

$$R_{\text{MIMO}}^i\text{: } (T_{x_1} \times \cdots \times T_{x_p}) \rightarrow (T_{y_1} + \cdots + T_{y_q})$$

where "$+$" represents the union of independent variables. Since the outputs of MIMO rule are independent, the general rule structure of MIMO fuzzy system can be represented as a collection of multi-input and single-output (MISO) fuzzy systems by decomposing the above rule into q subrules with T_{y_i} as the single consequent of the ith subrule. In this subsection, for clarity, we will consider MISO system in the following analysis. A sample rule is

IF the speed is TOO SLOW and the acceleration
is DECREASING,
THEN INCREASE POWER STRONGLY.

The inference engine in Fig. 2 matches the rule preconditions in the fuzzy rule base with the input state linguistic terms and performs implication. For example, if there are two rules

R1: IF x_1 is $T_{x_1}^1$ and x_2 is $T_{x_2}^1$, THEN y is T_y^1
R2: IF x_1 is $T_{x_1}^2$ and x_2 is $T_{x_2}^2$, THEN y is T_y^2

then the firing strengths of rules R1 and R2 are defined as α_1 and α_2, respectively. Here α_i is defined as

$$\alpha_i = M_{x_1}^i(x_1) \wedge M_{x_2}^i(x_2) \quad (3)$$

where "$\wedge$" is the *fuzzy AND* operation. The most commonly used fuzzy AND operations are intersection and algebraic product [37], [38].

Rules R1 and R2 lead to the corresponding decision with the membership function, $\hat{M}_y^i(w), i = 1, 2$, which is defined as

$$\hat{M}_y^i(w) = \alpha_i \wedge M_y^i(w) \quad (4)$$

where w is the variable that represents the membership function support values. Combining these decisions, we obtain the output decision

$$\hat{M}_y(w) = \hat{M}_y^1(w) \vee \hat{M}_y^2(w) \quad (5)$$

where "$\vee$" is the *fuzzy OR* operation. The most commonly used fuzzy OR operations are union and bounded sum [37], [38].

Notice that the last result is a membership function curve. Before sending out the signal to the plant, we must defuzzify it to get a crisp decision, which is what defuzzifier block in Fig. 2 does. Among commonly used defuzzification strategies, the *center of area* method yields a superior result [37], [38]. Let w_j be the support value at which the membership function, $\hat{M}_y^j(w)$, reaches the maximum value $\hat{M}_y^j(w)|_{w=w_j}$. Then the defuzzification output is

$$y = \frac{\sum_j \hat{M}_y^j(w_j) w_j}{\sum_j \hat{M}_y^j(w_j)}. \quad (6)$$

The preceding describes the standard function operations in a conventional fuzzy system, although there are some alternatives for fuzzy OR, fuzzy AND, and reasoning operations [37], [38].

Enabling a fuzzy system to learn is an important issue. The learning/adapting block in Fig. 2 represents this function. This learning/adapting block finds suitable fuzzy logic rules and adapts the fuzzifier and the defuzzifier to find the proper shapes and membership function overlaps by learning the desired outputs. Traditionally, the structure or the number of rules in an adaptive fuzzy system are all predecided, and the adaptation includes only the parameters of the membership functions. The aim of this paper is to present an adaptive filter that can adapt itself to match the input–output pairs and construct fuzzy rules automatically. In the next subsection, the adaptive neural fuzzy filter (ANFF), a feedforward connectionist model, is proposed. This neural-network-based architecture eliminates the structure predescription process and distributively stores mapping knowledge in the connection types and link weights. More importantly, the connectionist architecture is a natural structure for performing neural learning [26].

B. Adaptive Neural Fuzzy Filters

In this section, we will describe the structure and functions of the proposed ANFF, a connectionist type of filter constructed from a set of fuzzy if-then rules. The ANFF (see Fig. 3) has five layers with node and link numbering defined by the brackets on the left-hand side of the figure. Layer-1 nodes are input nodes (*input linguistic nodes*) representing input linguistic variables. Layer-5 nodes are output nodes (*output linguistic nodes*) representing output linguistic variables. Layer-2 and layer-4 nodes are *term nodes* that act as

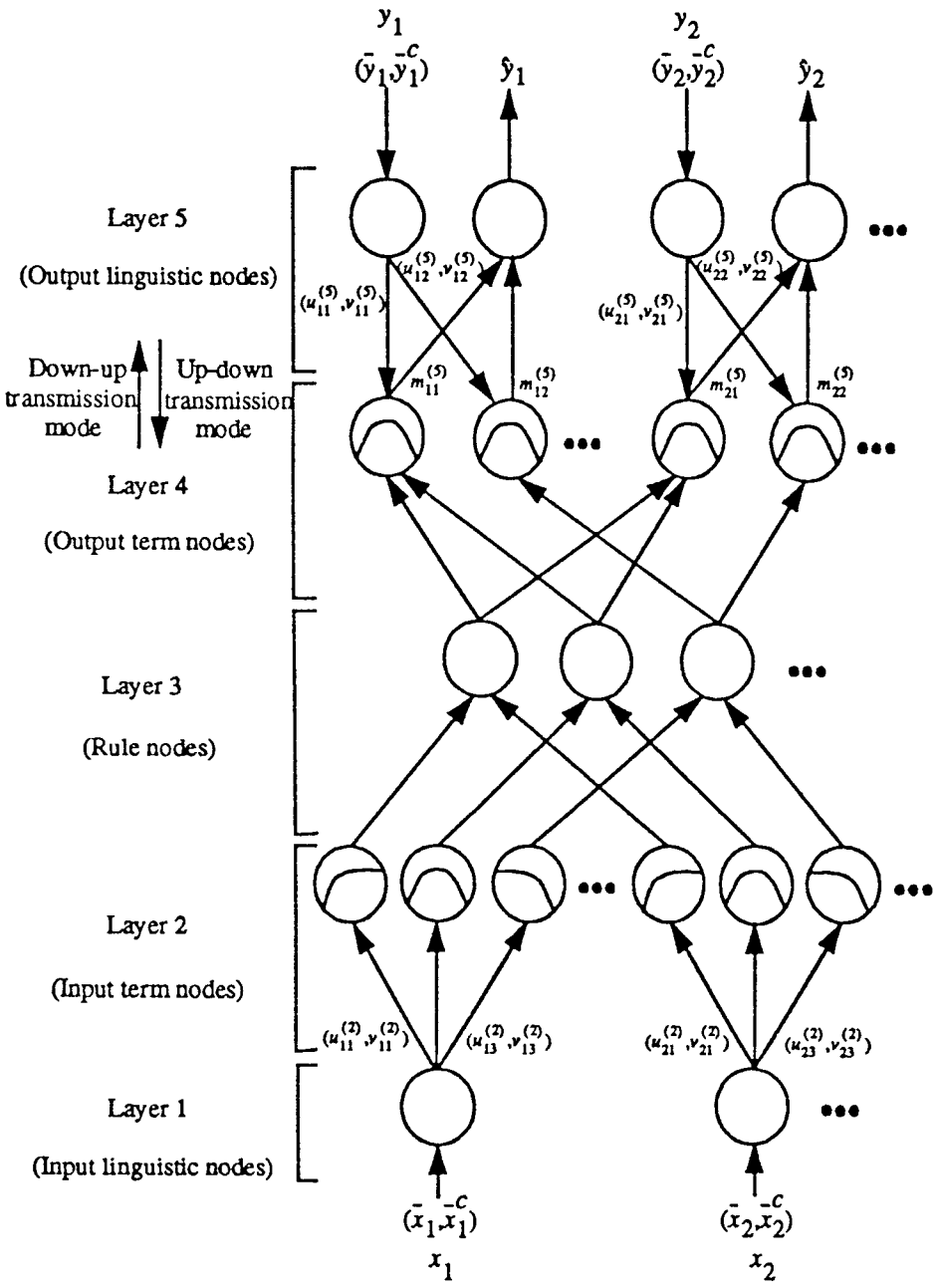


Fig. 3. Structure of the proposed ANFF.

membership functions representing the terms of the respective input and output linguistic variables. Each layer-3 node is a rule node representing one fuzzy logic rule. Thus, together all the layer-3 nodes form a fuzzy rule base. Links between layers 3 and 4 function as a *connectionist inference engine*. Layer-3 links define the preconditions of the rule nodes, and layer-4 links define the consequents of the rule nodes. Therefore, each rule node has at most one link to some term node of a linguistic node, and may have no such links. This is true both for precondition links (links in layer 3) and consequent links (links in layer 4). The links in layers 2 and 5 are fully connected between linguistic nodes and their corresponding term nodes. The arrows indicate the normal signal flow directions when the network is in operation (after it has been built and trained). We will later indicate the signal propagation, layer-by-layer, according to the arrow direction.

The ANFF uses the technique of *complement coding* from Fuzzy ART [30] to normalize the input–output training vectors. Complement coding is a normalization process that rescales an n-dimensional vector in $\Re^n, \boldsymbol{x} = (x_1, x_2, \cdots, x_n)$, to its $2n$-dimensional complement coding form in $[0,1]^{2n}, \boldsymbol{x}'$, such that

$$\begin{aligned}\boldsymbol{x}' &\equiv (\overline{x}_1, \overline{x}_1^c, \overline{x}_2, \overline{x}_2^c, \cdots, \overline{x}_n, \overline{x}_n^c) \\ &= (\overline{x}_1, 1-\overline{x}_1, \overline{x}_2, 1-\overline{x}_2, \cdots, \overline{x}_n, 1-\overline{x}_n) \end{aligned} \tag{7}$$

where $(\overline{x}_1, \overline{x}_2, \cdots, \overline{x}_n) = \overline{\boldsymbol{x}} = \boldsymbol{x}/\|\boldsymbol{x}\|$ and $\overline{x}_1^c$ is the complement of $\overline{x}_1$, i.e., $\overline{x}_1^c = 1 - \overline{x}_1$. As mentioned in [30], complement coding helps avoid the problem of category

proliferation when using fuzzy ART for fuzzy clustering. It also preserves training vector amplitude information. In applying the complement coding technique to the ANFF, all training vectors (either input state vectors or desired output vectors) are transformed to their complement coded forms in the preprocessing process, and the transformed vectors are then used for training.

A typical network consists of nodes with some finite number of fan-in connections from other nodes represented by weight values, and fan-out connections to other nodes. Associated with the fan-in of a node is an integration function f which combines information, activation, or evidence from other nodes, and provides the net input, i.e.,

$$\text{net-input} = f(z_1^{(k)}, z_2^{(k)}, \cdots, z_p^{(k)}; w_1^{(k)}, w_2^{(k)}, \cdots, w_p^{(k)}) \tag{8}$$

where $z_i^{(k)}$ is the ith input to a node in layer k, and $w_i^{(k)}$ is the weight of the associated link. The superscript in the above equation indicates the layer number. This notation will be also used in the following equations. Each node also outputs an activation value as a function of its net-input

$$\text{output} = a(f) \tag{9}$$

where $a(\cdot)$ denotes the activation function. We will next describe the functions of the nodes in each of the five layers of the ANFF. Assume that the dimension of the input space is n, and that of the output space is m.

Layer 1: Each node in this layer is called an input linguistic node and corresponds to one input linguistic variable. Layer-1 nodes just transmit input signals to the next layer directly. That is,

$$f(\overline{x}_i, \overline{x}_i^c) = (\overline{x}_i, \overline{x}_i^c) = (\overline{x}_i, 1 - \overline{x}_i) \quad \text{and} \quad a(f) = f. \tag{10}$$

From the above equation, the link weight in layer 1 $(w_i^{(1)})$ is unity. Notice that due to the complement coding process, for each input node i, there are two output values, $\overline{x}_i$ and $\overline{x}_i^c = 1 - \overline{x}_i$.

Layer 2: Nodes in this layer are called input term nodes and each represents a term of an input linguistic variable, and acts as a one-dimensional membership function. The following trapezoidal membership function [39] is used

$$f(z_{ij}^{(2)}) = \frac{1}{n}\left[1 - g(z_{ij}^{(2)} - v_{ij}^{(2)}, \gamma) - g(u_{ij}^{(2)} - z_{ij}^{(2)}, \gamma)\right]$$

and

$$a(f) = f \tag{11}$$

where $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$ are, respectively, the left-flat and right-flat points of the trapezoidal membership function of the jth input term node of the ith input linguistic node [see Fig. 4(a)]; $z_{ij}^{(2)}$ is the input to the jth input term node from the ith input linguistic node (i.e., $z_{ij}^{(2)} = \overline{x}_i$); and

$$g(s, \gamma) = \begin{cases} 1, & \text{if } s\gamma > 1 \\ s\gamma, & \text{if } 0 \le s\gamma \le 1 \\ 0, & \text{if } s\gamma < 0. \end{cases} \tag{12}$$

The parameter γ is the sensitivity parameter that regulates the fuzziness of the trapezoidal membership function. A large γ means a more crisp fuzzy set, and a smaller γ makes the fuzzy set less crisp. A set of n input term nodes (one for each input linguistic node) is connected to a rule node in layer 3 where its outputs are combined. This defines an n-dimensional membership function in the input space, with each dimension specified by one input term node in the set. Hence, each input linguistic node has the same number of term nodes. That is, each input linguistic variable has the same number of terms in the ANFF. This is also true for output linguistic nodes. A layer-2 link connects an input linguistic node to one of its term nodes. There are two weights on each layer-2 link. We denote the two weights on the link from input node i (corresponding to the input linguistic variable x_i) to its jth term node as $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$ (see Fig. 3). These two weights define the membership function in (11). The two weights, $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$, correspond, respectively, to the two inputs, $\overline{x}_i$ and $\overline{x}_i^c$ from the input linguistic node i. More precisely, $\overline{x}_i$ and $\overline{x}_i^c$, the two inputs to the input term node j, will be used during the fuzzy-ART clustering process in ANFF's structure-learning step to decide $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$, respectively. In ANFF's parameter-learning step and normal operating, only $\overline{x}_i$ is used

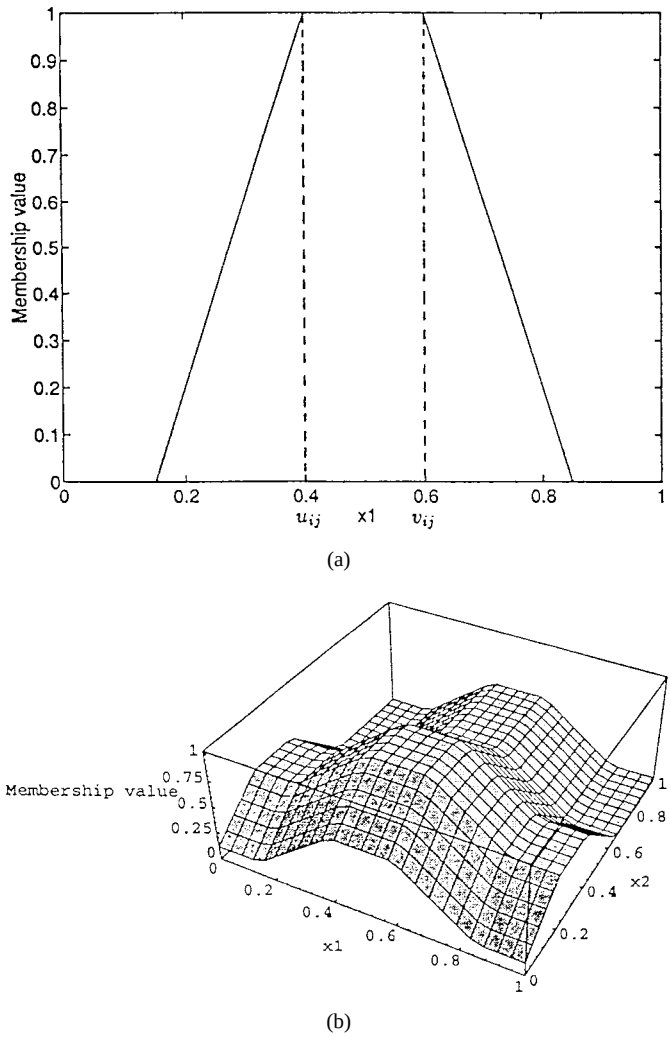


Fig. 4. (a) One-dimensional and (b) two-dimensional trapezoidal membership function.

in the forward reasoning process [i.e., $z_{ij}^{(2)} = \overline{x}_i$ in (11)]. We detail the ANFF learning scheme in Section IV.

Layer 3: Nodes in this layer are called rule nodes and each represents one fuzzy logic rule. Each layer-3 node has n input term nodes fed into it, one for each input linguistic node. Hence, there are as many rule nodes in the ANFF as there are term nodes of an input linguistic node (i.e., the number of rules equals the number of terms of an input linguistic variable). Notice that each input linguistic variable has the same number of terms in the ANFF as mentioned in the above. The links in layer 3 are used to perform precondition matching of fuzzy logic rules. Hence the rule nodes perform the operation

$$f(z_i^{(3)}) = \sum_{i=1}^{n} z_i^{(3)} \quad \text{and} \quad a(f) = f \tag{13}$$

where $z_i^{(3)}$ is the ith input to a node in layer 3 and the summation is over the inputs of this node. The link weight in layer 3 ($w_i^{(3)}$) is then unity. The summation in the above equation is equivalent to defining a multidimensional (n-dimensional) membership function, which is the summation of the trapezoid functions in (11) over i. This forms a multidimensional trapezoidal membership function called the *hyperbox membership function* [39], since it is defined on a hyperbox in the input space. The corners of the hyperbox are decided by the layer-2 weights, $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$, for all i's. More clearly, the interval $[u_{ij}^{(2)}, v_{ij}^{(2)}]$ defines the edge of the hyperbox in the ith dimension. Hence, the weight vector $[(u_{1j}^{(2)}, v_{1j}^{(2)}), \cdots, (u_{ij}^{(2)}, v_{ij}^{(2)}), \cdots, (u_{nj}^{(2)}, v_{nj}^{(2)})]$, defines a hyperbox in the input space. An illustration of a two-dimensional hyperbox membership function is shown in Fig. 4(b). The rule nodes output are connected to sets of m output term nodes in layer 4, one for each output linguistic variable. This set of output term nodes defines an m-dimensional trapezoidal (hyperbox) membership function in the output space that specifies the consequent of the rule node. Different rule nodes may be connected to the same output hyperbox (i.e., they may have the same consequent) as shown in Fig. 3.

Layer 4: The nodes in this layer are called output term nodes; each has two operating modes: *down-up* transmission and *up-down* transmission (see Fig. 3). In down-up transmission mode, the links in layer 4 perform the fuzzy OR operation on fired (activated) rule nodes that have the same consequent

$$f(z_i^{(4)}) = \max(z_1^{(4)}, z_2^{(4)}, \cdots, z_p^{(4)}) \quad \text{and} \quad a(f) = f \tag{14}$$

where $z_i^{(4)}$ is the ith input to a node in layer 4 and p is the number of inputs to this node from the rule nodes in layer 3. Hence the link weight is $w_i^{(4)} = 1$. In up-down transmission mode, the nodes in this layer and the up-down transmission links in layer 5 function exactly the same as those in layer 2: each layer-4 node represents a term of an output linguistic variable and acts as a one-dimensional membership function. A set of m output term nodes, one for each output linguistic node, defines an m-dimensional hyperbox (membership function) in the output space, and there are also two weights, $u_{ij}^{(5)}$ and $v_{ij}^{(5)}$, on each of the up-down transmission links in layer 5 (see Fig. 3). The weights define hyperboxes (and thus the associated hyperbox membership functions) in the output space. More clearly, the weight vector, $[(u_{1j}^{(5)}, v_{1j}^{(5)}), \cdots, (u_{ij}^{(5)}, v_{ij}^{(5)}), \cdots, (u_{mj}^{(5)}, v_{mj}^{(5)})]$, defines a hyperbox in the output space.

Layer 5: Each node in this layer is called a output linguistic node and corresponds to one output linguistic variable. There are two kinds of nodes in layer 5. The first kind of node performs up-down transmission for training data (desired outputs) to feed into the network, acting exactly like the input linguistic nodes. For this kind of node, we have

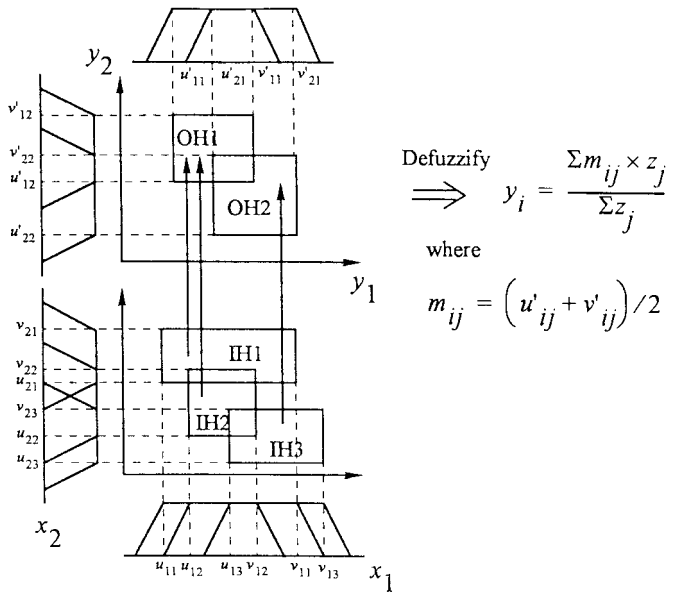
$$f(\overline{y}_i, \overline{y}_i^c) = (\overline{y}_i, \overline{y}_i^c) = (\overline{y}_i, 1 - \overline{y}_i) \quad \text{and} \quad a(f) = f \tag{15}$$

where $\overline{y}_i$ is the ith element of the normalized desired output vector. Notice that complement coding is also performed on the desired output vectors. Thus, as mentioned above, there are two weights on each of the up-down transmission links in layer 5 (the $u_{ij}^{(5)}$ and $v_{ij}^{(5)}$ shown in Fig. 3). The weights define hyperboxes and the associated hyperbox membership functions in the output space. The second kind of node performs down-up transmission for decision signal output. These nodes and the layer-5 down-up transmission links attached to them act as a defuzzifier. If $u_{ij}^{(5)}$ and $v_{ij}^{(5)}$ are the corners of the hyperbox of the jth term of the ith output linguistic variable y_i, then the following functions can be used to simulate the *center of area* defuzzification method:

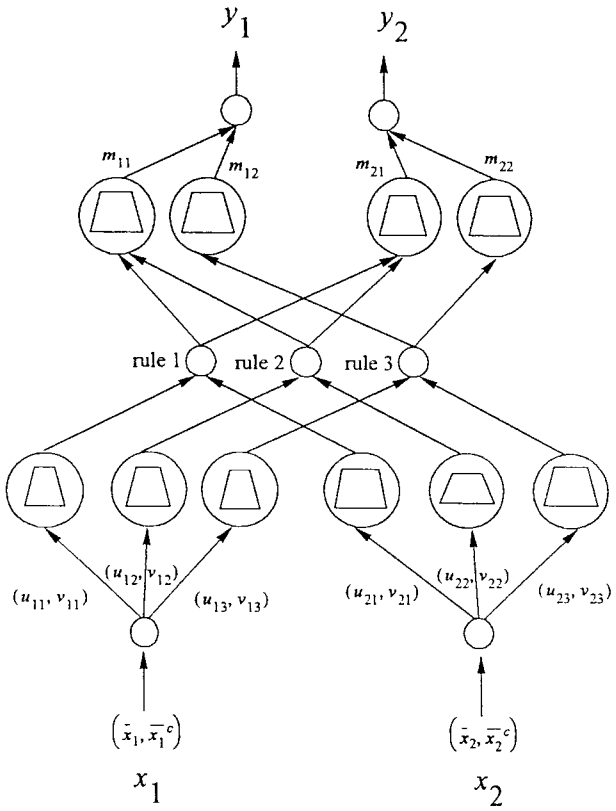
$$f(z_j^{(5)}) = \sum_j w_{ij}^{(5)} z_j^{(5)} = \sum_j m_{ij}^{(5)} z_j^{(5)} \quad \text{and}$$
$$a(f) = \frac{f}{\sum_j z_j^{(5)}} \tag{16}$$

where $z_j^{(5)}$ is the input to the ith output linguistic node from its jth term node, and $m_{ij}^{(5)} = (u_{ij}^{(5)} + v_{ij}^{(5)})/2$ denotes the center value of the output membership function of the jth term of the ith output linguistic variable. The center of a fuzzy region is defined as the point with the smallest absolute value among all the other points in the region at which the value of membership function is equal to one. Here the weight, $w_{ij}^{(5)}$, on a down-up transmission link in layer 5 is defined by $w_{ij}^{(5)} \equiv m_{ij}^{(5)} = (u_{ij}^{(5)} + v_{ij}^{(5)})/2$, where $u_{ij}^{(5)}$ and $v_{ij}^{(5)}$ are the weights on the corresponding up-down transmission link in layer 5.

The fuzzy reasoning process in the ANFF is illustrated in Fig. 5(a), which shows a graphic interpretation of the center of area defuzzification method. Fig. 5(b) shows the corresponding structure of the ANFF. Here, we consider a two-input and two-output case. As shown in the figure, three hyperboxes (IH1, IH2, and IH3) are formed in the input space and two hyperboxes (OH1, OH2) are formed in the output space. These hyperboxes are defined by the weights u_{ij}, v_{ij}, u'_{ij}, and v'_{ij}. The three fuzzy rules indicated in the figure are "IF $\boldsymbol{x}$ is IH1 THEN $\boldsymbol{y}$ is OH1 (rule 1)," "IF $\boldsymbol{x}$ is IH2 THEN $\boldsymbol{y}$ is OH1 (rule 2)," and "IF $\boldsymbol{x}$ is IH3 THEN $\boldsymbol{y}$ is OH2 (rule 3)," where $\boldsymbol{x} = (x_1, x_2)$ and $\boldsymbol{y} = (y_1, y_2)$. If an input pattern is located inside a hyperbox, the membership value is equal to one [see (12)]. In this figure, according to (14),



(a)



(b)

Fig. 5. (a) The fuzzy reasoning process in the ANFF model. (b) The corresponding ANFF structure of (a).

z_1 is obtained by performing fuzzy OR (maximum) operation on the inferred results of rules 1 and 2, which have the same consequent, OH1. Also according to (14), z_2 is directly the inferred result of rule 3. z_1 and z_2 are then defuzzified to get the final output according to (16).

Note that with the proposed learning algorithms developed in Section IV, no input–output term nodes and no rule node exist when learning begins. They are created dynamically as on-line teaching signals are received and learning proceeds. But for the cases where some expert knowledges, which are expressed as fuzzy if-then rules, are known in advance, these rules can be presented as initial rules in the ANFF. As training proceeds, if the expert knowledge is not able to handle the desired mapping, new input and output term nodes and rule nodes are added during the learning process. This enables the combination of expert knowledge and numerical training data into a filter, a major advantage of the ANFF.

IV. Learning Algorithm for the ANFF

In this section, we develop an on-line learning algorithm to find the optimal fuzzy filter under the MSE criterion. The learning algorithm combines structure learning and parameter learning to determine the proper corners of the hyperbox (u_{ij}'s and v_{ij}'s) for each term node in layers 2 and 4. It also learns fuzzy logic rules and link connection types in layers 3 and 4, that is, the precondition and consequent links of the rule nodes.

A. Problem Formulation

The problem of the design or adaptation of an optimal fuzzy filter can be phrased as follows. Given a process $\boldsymbol{x}(m)$ specified in a finite interval length, $m_1 < m < m_2$, we are to design a nonlinear filter in such a way that the estimated value, $\hat{s}(k)$, based upon $\boldsymbol{x}(m)$ is as close as possible to the desired process $s(k)$. Written in mathematical form, we have

$$\hat{s}(k) = F(\boldsymbol{x}(m)) \tag{17}$$

where k is in the interval $[m_1, m_2]$, and $F(\cdot)$ represents the function of the desired nonlinear filter. The process $\boldsymbol{x}(m)$ is usually a nonlinear version of $s(k)$ corrupted with noise $n(m)$. The objective is to find the optimal filter, $F(\cdot)$, so as to minimize the MSE

$$E[(s(k) - \hat{s}(k))^2] = E[(s(k) - F(\boldsymbol{x}(m)))^2]. \tag{18}$$

Two steps, the structure learning step and the parameter learning step as shown in Fig. 6, are used concurrently to achieve this goal, and are introduced in the following two subsections.

B. The Structure-Learning Step

The structure-learning task can be stated as following. Given input training data at time $k, x_i(k), i = 1, \cdots, n$ and desired output value $s(k)$, find proper fuzzy partitions, membership functions, and fuzzy logic rules. At this step, the network works in a two-sided manner, that is, the nodes and links in layer 4 are in the up-down transmission mode so training input and output data are fed in the ANFF from both sides.

The structure-learning step consists of three learning processes: input fuzzy clustering process, output fuzzy clustering process, and mapping process. The first two processes are performed simultaneously on both sides of the network, and are described below.

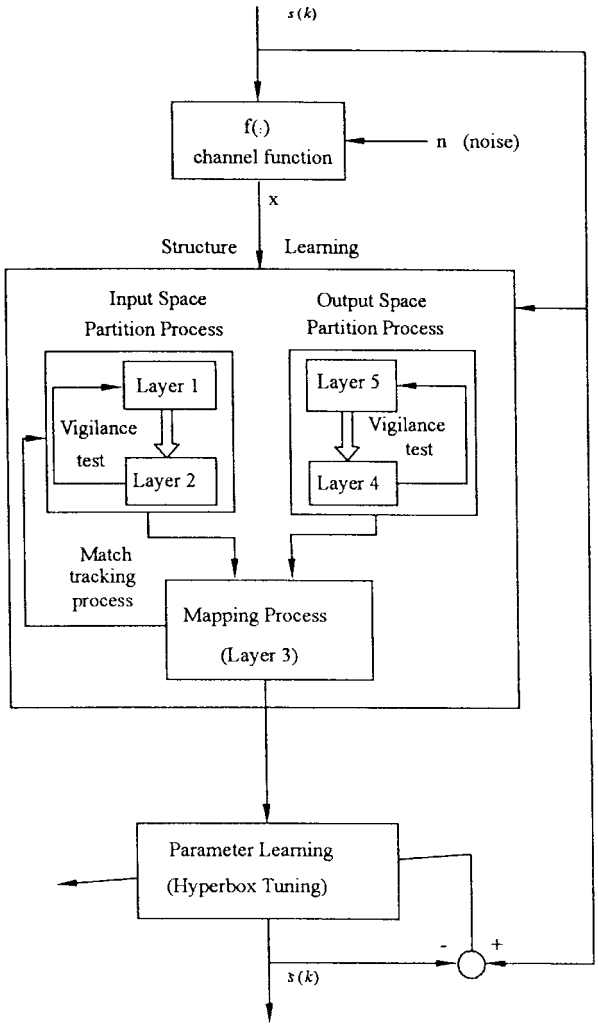


Fig. 6. Flowchart of the learning algorithm for the ANFF.

1) Input Fuzzy Clustering Process: We use the fuzzy ART fast learning algorithm [30], [31] to find the input membership function parameters, $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$. This is equivalent to finding proper input space's fuzzy clustering or, more precisely, to forming proper fuzzy hyperboxes in the input space. Compared with other fuzzy clustering techniques, the major advantage of this one is on the ability of on-line generation of a new cluster when necessary. Hence, no preassignment of the number of rules is required. Initially, for each complement coded input vector $\boldsymbol{x}'$ [see (7)], the values of choice functions, T_j, are computed by

$$T_j(\boldsymbol{x}') = \frac{|\boldsymbol{x}' \wedge \boldsymbol{w}_j|}{\alpha + |\boldsymbol{w}_j|} \qquad j = 1, 2, \cdots, N \tag{19}$$

where "$\wedge$" is the minimum operator performed for the pairwise elements of two vectors, $\alpha \geq 0$ is a constant, N is the current number of rule nodes, and $\boldsymbol{w}_j$ is the *complement weight vector*, which is defined by

$$\boldsymbol{w}_j \equiv [(u_{1j}^{(2)}, 1-v_{1j}^{(2)}), \cdots, (u_{ij}^{(2)}, 1-v_{ij}^{(2)}), \cdots, (u_{nj}^{(2)}, 1-v_{nj}^{(2)})].$$

Notice that $[(u_{1j}^{(2)}, v_{1j}^{(2)}), \cdots, (u_{ij}^{(2)}, v_{ij}^{(2)}), \cdots, (u_{nj}^{(2)}, v_{nj}^{(2)})]$ is the weight vector of layer-2 links associated with rule node j. The choice function value indicates the similarity between the input vector $\boldsymbol{x}'$ and the complement weight vector $\boldsymbol{w}_j$. We then need to find the complement weight vector closest to $\boldsymbol{x}'$. This is equivalent to finding a hyperbox (category) that $\boldsymbol{x}'$ could belong to. The chosen category is indexed by J, where

$$T_J = \max\{T_j : j = 1, \cdots, N\}. \tag{20}$$

Resonance occurs when the match value of the chosen category meets the vigilance criterion

$$\frac{|\boldsymbol{x}' \wedge \boldsymbol{w}_J|}{|\boldsymbol{x}'|} \geq \rho \tag{21}$$

where $\rho \in [0, 1]$ is a vigilance parameter. If the vigilance criterion is not met, we say *mismatch reset* occurs. In this case, the choice function value T_J is set to zero for the duration of the input presentation to prevent persistent selection of the same category during search (we call this action "disabling J"). A new index J is then chosen using (20). The search process continues until the chosen J satisfies (21). This search process is indicated by the feedback arrow marked with "vigilance test" in Fig. 6. If no such J is found, then a new input hyperbox is created by adding a set of n new input term nodes, one for each input linguistic variable, and setting up links between the newly added input term nodes and the input linguistic nodes. The complement weight vectors on these new layer-2 links are simply given as the current input vector, $\boldsymbol{x}'$. These newly added input term nodes and links define a new hyperbox, and thus a new category, in the input space. We denote this newly added hyperbox as J.

2) Output Fuzzy Clustering Process: The output fuzzy clustering process is exactly the same as the input fuzzy clustering process except that it is performed between layers 4 and 5 which are working in the up-down transmission mode. Of course, the training pattern used now is the desired output vector after complement coding, $\boldsymbol{s}' = (\overline{s}, \overline{s}^c) = (\overline{s}, 1 - \overline{s})$. We denote the chosen or newly added output hyperbox by K. This hyperbox is defined by the complement weight vector in layer 5,

$$\boldsymbol{w}_K = [(u_{1j}^{(5)}, 1-v_{1j}^{(5)}), \cdots, (u_{ij}^{(5)}, 1-v_{ij}^{(5)}), \cdots, (u_{mj}^{(5)}, 1-v_{mj}^{(5)})].$$

The above two fuzzy clustering processes produce a chosen input hyperbox indexed as J and a chosen output hyperbox indexed as K, where the input hyperbox J is defined by $\boldsymbol{w}_J$ and the output hyperbox K by $\boldsymbol{w}_K$. If the chosen input hyperbox J is not newly added, then there is a rule node, J, that corresponds to it. If the input hyperbox J is a newly added one, then a new rule node (indexed as J) in layer 3 is added, and connected to the input term nodes that constitute it.

3) Mapping Process: After the two hyperboxes in the input and output spaces are chosen in the input and output fuzzy clustering processes, the next step is to perform the mapping process which decides the connections between layer-3 and layer-4 nodes. This is equivalent to deciding the consequents of fuzzy logic rules. This mapping process is described by the following algorithm, wherein connecting rule node J to output hyperbox K we means connecting the rule node J to the output term nodes that constitutes the hyperbox K in the output space.

Step 1: IF rule node J is a newly added node

THEN connect rule node J to output hyperbox K.

Step 2: ELSE IF rule node J is not connected to output hyperbox K originally

THEN disable J and perform Input Fuzzy Clustering Process to find the next qualified J [i.e., the next rule node that satisfies (20) and (21)].

Go to Step 1.

Step 3: ELSE no structure change is necessary.

In the mapping process, hyperboxes J and K are resized according to the *fast learning rule* [30] by updating weights, $\boldsymbol{w}_J$ and $\boldsymbol{w}_K$, as

$$\boldsymbol{w}_J^{(\text{new})} = \boldsymbol{x}' \wedge \boldsymbol{w}_J^{(\text{old})}, \quad \boldsymbol{w}_K^{(\text{new})} = \boldsymbol{s}' \wedge \boldsymbol{w}_K^{(\text{old})}. \tag{22}$$

Note that once the consequent of a rule node has been decided in the mapping process, it will not be changed thereafter. To show how the structure learning of the ANFF works, a simple example is given below.

4) Simple Example: Fig. 5(b) shows the structure of an ANFF, which is being constructed during the learning process. Learning is performed continuously for succeeding incoming training data. Fig. 5(a) shows the generated two-dimensional hyperboxes in the input–output spaces, and the projected membership functions for each variable. Fig. 5(b) shows the corresponding structure of the ANFF. For a given training datum, the input fuzzy clustering process and the output fuzzy clustering process find or form proper clusters (hyperboxes) in the input and output spaces, respectively. Assume that the input and output hyperbox pair found (or formed) are (J, K). The mapping process then tries to relate these two hyperboxes by setting up links between them. This is equivalent to finding a fuzzy logic rule that defines the association between an input hyperbox and an output hyperbox. The following cases may happen during the mapping process. Case 1: If the input hyperbox J and output hyperbox K as well as their association (J, K) exist already [e.g., $(J, K) =$ (IH1, OH1), (IH2, OH1), or (IH3, OH2) in Fig. 5(a)], then only Step 3 in the mapping process is satisfied and thus no structural change is necessary. Case 2: If input hyperbox J is newly formed (i.e., $J =$ IH4), and thus not connected to any output hyperbox, then Step 1 in the mapping process is satisfied and input hyperbox J will be connected to output hyperbox K directly, where K could be OH1, OH2, or a newly formed hyperbox, OH3. Case 3: If input hyperbox J is associated with an output hyperbox different from K originally (i.e., assume $(J, K) =$ (IH2, OH2), but the original mapping is (IH2, OH1)), then Step 2 in the mapping process is satisfied and a new input hyperbox close to J will be found or formed by performing the input fuzzy clustering process again. This search, called "match tracking" (see Fig. 6), continues until an input hyperbox, J', that can be associated with output hyperbox K is found [e.g., $(J', K) =$ (IH3, OH2)].

The vigilance parameter, ρ, is an important structure-learning parameter that determines learning cluster density. High (approaching 1) ρ values tend to produce increasingly finer learning clusters, until at 1, each training datum is assigned to its own cluster in the input (output) space. Low (approaching 0) ρ values tend to produce increasingly coarser learning clusters, until at 0, all training data are assigned to a single cluster in the input (output) space.

Clearly, a constantly high or low ρ value will result in formation of excessively high numbers of clusters on the one hand, or very low output accuracy (and thus, low network representation power) on the other hand. For these reasons, we chose an adaptive vigilance strategy in which the ρ parameter is initially set high to allow fast ANFF structure growth, and then monotonically decreased to slow cluster formation and stabilize learning. Empirical studies have shown this approach to be efficient and stable in the learning speeds and numbers of clusters it produces.

C. The Parameter-Learning Step

After the network structure has been adjusted according to the current training pattern in the structure-learning step, it is then necessary to fine tune the network parameters using the same training pattern. This fine tuning process is necessary to assure the desired output accuracy of a network. Using the terminology of fuzzy logic: once our adaptive neural fuzzy filter has found its fuzzy logic rules, its membership functions must be tuned to make its output meet the desired output as closely as possible. Notice that the following parameter learning is performed on the whole network after the structure learning step, no matter whether the nodes (links) are newly added or are existent originally. The parameter-learning task can be stated as: Given the training input data $x_i(k), i = 1, \cdots, n$, the desired output value $s(k)$, and the network structure (specified by input and output hyperboxes and fuzzy logic rules), we need to adjust the network parameters to make the network output match the desired output values as closely as possible. Thus, the network works in a feedforward manner; that is, the nodes and links in layer 4 are in the down-up transmission mode. Basically, the backpropagation algorithm is used to find node output errors, which are then analyzed to guide parameter adjustment.

As mentioned above, the goal of training the ANFF is to minimize the error function [see (18)]

$$E = \tfrac{1}{2}(s(k) - \hat{s}(k))^2 \tag{23}$$

where $s(k)$ is the desired signal, and $\hat{s}(k)$ is the filtered signal. Based upon this MSE criterion and in analogy to the backpropagation algorithm, we can derive the following general parameter-learning rule:

$$w(k+1) = w(k) + \Delta w(k) = w(k) + \eta\left(-\frac{\partial E}{\partial w}\right) \tag{24}$$

$$-\frac{\partial E}{\partial w} = -\frac{\partial E}{\partial f}\,\frac{\partial f}{\partial w} = -\frac{\partial E}{\partial a}\,\frac{\partial a}{\partial f}\,\frac{\partial f}{\partial w} \tag{25}$$

where w is the adjustable parameter in the filter (i.e., u_{ij} or v_{ij}). To show the parameter-learning rules, we derive the rules layer-by-layer using the hyperbox membership functions with corners u_{ij}'s and v_{ij}'s as the adjustable parameters for these computations. In the following derivation, we consider only one output linguistic variable for notational clarity. Hence, the adjustable parameters in layer 5 are denoted by $u_j^{(5)}, v_j^{(5)}$, and $m_j^{(5)} = (u_j^{(5)} + v_j^{(5)})/2$, for the jth term node.

Layer 5: Using (16), (24), and (25), the updating rule for the corners of the hyperbox membership function $v_j^{(5)}$ is

$$\frac{\partial E}{\partial v_j^{(5)}} = \frac{\partial E}{\partial a^{(5)}} \frac{\partial a^{(5)}}{\partial v_j^{(5)}} = -(s(k) - \hat{s}(k)) \frac{z_j^{(5)}}{2\sum z_j^{(5)}}. \quad (26)$$

And the corner parameter is updated by

$$v_j^{(5)}(k+1) = v_j^{(5)}(k) + \eta(s(k) - \hat{s}(k)) \frac{z_j^{(5)}}{2\sum z_j^{(5)}}. \quad (27)$$

Similarly, using (16), (24), and (25), the updating rule for the other corner parameter $u_j^{(5)}$ is

$$\frac{\partial E}{\partial u_j^{(5)}} = \frac{\partial E}{\partial a^{(5)}} \frac{\partial a^{(5)}}{\partial u_j^{(5)}} = -(s(k) - \hat{s}(k)) \frac{z_j^{(5)}}{2\sum z_j^{(5)}}. \quad (28)$$

And this corner parameter is updated by

$$u_j^{(5)}(k+1) = u_j^{(5)}(t) + \eta(s(k) - \hat{s}(k)) \frac{z_j^{(5)}}{2\sum z_j^{(5)}}. \quad (29)$$

Notice that since the update values for $v_j^{(5)}$ and $u_j^{(5)}$ are the same, the constraint $v_j^{(5)} \geq u_j^{(5)}$ associated with the trapezoidal membership function in the output space is preserved after tuning.

The error propagated to the preceding layer is

$$\delta^{(5)} = -\frac{\partial E}{\partial a^{(5)}} = s(k) - \hat{s}(k). \quad (30)$$

Layer 4: There is no parameter to be adjusted in this layer. Only the error signal $(\delta_i^{(4)})$ needs to be computed and propagated. According to (16), the error signal $\delta_i^{(4)}$ is derived by

$$\delta_i^{(4)} = -\frac{\partial E}{\partial a^{(4)}} = -\frac{\partial E}{\partial a^{(5)}} \frac{\partial a^{(5)}}{\partial a^{(4)}} \quad (31)$$

where

$$\frac{\partial E}{\partial a^{(4)}} = -\delta^{(5)} \quad (32)$$

$$\frac{\partial a^{(5)}}{\partial a^{(4)}} = \frac{m_i^{(5)} \sum z_i^{(5)} - \sum m_i^{(5)} z_i^{(5)}}{\left(\sum z_i^{(5)}\right)^2}. \quad (33)$$

Hence, the error signal is

$$\delta_i^{(4)} = \delta^{(5)} \frac{m_i^{(5)} \sum z_i^{(5)} - \sum m_i^{(5)} z_i^{(5)}}{\left(\sum z_i^{(5)}\right)^2}. \quad (34)$$

In the multi-output case, the computations in layers 5 and 4 are exactly the same as the above and proceed independently for each output linguistic variable.

Layer 3: As in layer 4, only the error signals need to be computed in this layer. According to (14), this error signal can be derived by

$$\delta_i^{(3)} = -\frac{\partial E}{\partial a^{(3)}} = -\frac{\partial E}{\partial a^{(4)}} \frac{\partial a^{(4)}}{\partial f^{(4)}} \frac{\partial f^{(4)}}{\partial a^{(3)}} \quad (35)$$

where

$$\frac{\partial E}{\partial a^{(4)}} = -\delta_i^{(4)} \quad (36)$$

$$\frac{\partial a^{(4)}}{\partial f^{(4)}} = 1 \quad (37)$$

$$\frac{\partial f^{(4)}}{\partial a^{(3)}} = \frac{\partial f^{(4)}}{\partial z_i^{(4)}} = \frac{z_i^{(4)}}{z_{\max}} \quad (38)$$

where $z_{\max} = \max(\text{inputs of output terms node } j)$. The term, $z_i^{(4)}/z_{\max}$, normalizes the error to be propagated for fired rules with the same consequent. Hence the error signal is

$$\delta_i^{(3)} = \delta_i^{(4)} \frac{z_i^{(4)}}{z_{\max}}. \quad (39)$$

If there are multiple outputs, then the error signal becomes

$$\delta_i^{(3)} = \sum_k \frac{z_k^{(4)}}{z_{\max\, t-1}} \delta_k^{(4)}$$

where the summation is performed over the consequents of a rule node; that is, the error of a rule node is the summation of the errors of its consequents.

Layer 2: Using (11), (24), and (25), the updating rule of $v_{ij}^{(2)}$ is derived as in

$$-\frac{\partial E}{\partial v_{ij}^{(2)}} = -\frac{\partial E}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial v_{ij}^{(2)}} \quad (40)$$

where

$$\frac{\partial a^{(3)}}{\partial a^{(2)}} = 1 \quad (41)$$

$$\frac{\partial a^{(2)}}{\partial v_{ij}^{(2)}} = \begin{cases} \frac{\gamma}{n}, & \text{if } 0 \leq (\overline{x}_i - v_{ij}^{(2)})\gamma \leq 1 \\ 0, & \text{otherwise.} \end{cases} \quad (42)$$

So the updating rule of $v_{ij}^{(2)}$ is

$$v_{ij}^{(2)}(t+1) = v_{ij}^{(2)}(t) + \eta \delta_i^{(3)} \frac{\partial a^{(2)}}{\partial v_{ij}^{(2)}}. \quad (43)$$

Similarly, using (11), (24), and (25), the updating rule of $u_{ij}^{(2)}$ is derived as

$$-\frac{\partial E}{\partial u_{ij}^{(2)}} = -\frac{\partial E}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial u_{ij}^{(2)}} \quad (44)$$

where

$$\frac{\partial a^{(3)}}{\partial a^{(2)}} = 1 \quad (45)$$

$$\frac{\partial a^{(2)}}{\partial u_{ij}^{(2)}} = \begin{cases} -\frac{\gamma}{n}, & \text{if } 0 \leq (u_{ij}^{(2)} - \overline{x}_i)\gamma \leq 1 \\ 0, & \text{otherwise.} \end{cases} \quad (46)$$

Hence, the updating rule of u_{ij} becomes

$$u_{ij}^{(2)}(k+1) = u_{ij}^{(2)}(k) + \eta\delta_i^{(3)} \frac{\partial a^{(2)}}{\partial u_{ij}^{(2)}}. \tag{47}$$

Notice that after tuning, the constraint $v_{ij}^2 \geq u_{ij}^2$ associated with the trapezoidal membership function in the input space is kept by setting $u_{ij}^{(2)}(k+1) = v_{ij}^{(2)}(k+1)$, if the violation condition $u_{ij}^{(2)}(k+1) > v_{ij}^{(2)}(k+1)$ is encountered.

V. Applications and Simulations

A. Application to Nonlinear Channel Equalization

The proposed ANFF is used as a nonlinear channel equalizer in this application. In [26], the first use of fuzzy filter to equalize nonlinear channels is proposed. Nonlinear channel equalization is a technique used to combat some imperfect phenomenon (mainly refers to intersymbol interference in the presence of noise) in high-speed data transmission over channels, like the high-speed modems [1]. The structure of the system is shown in Fig. 7. The transmitted input signal, $s(k)$, is a sequence of statistically independent random binary symbols which takes values of 1 or -1 with equal probability. The signal is sent through the channel. In real communications, the channel (like the telephone channel and radio channel) is in fact dispersive and the dispersion will cause interference between successive samples (intersymbol interference) which greatly complicates reliable transmission and reception. If $\hat{x}(k)$ denotes the output of the channel, then the channel function can be described as

$$\hat{x}(k) = f(s(k), s(k-1), \cdots, s(k-m)). \tag{48}$$

In general, f is a nonlinear function of the past transmitted signals, and the channels change slowly but significantly over time, so a nonlinear channel equalizer with adaptation ability is needed. At the receiving end, the observed signal $x(k)$ is the channel output $\hat{x}(k)$ corrupted by additive noise $e(k)$, that is

$$x(k) = \hat{x}(k) + e(k). \tag{49}$$

The task of the equalizer is to reconstruct the transmitted signal, $s(k-d)$, from the observed information sequence $x(k), x(k-1), \cdots, x(k-n+1)$ (where d and n denote the lag and order, respectively) such that greater speed and reality can be achieved. Following the functions defined in [13], [15], the geometric formulation of the equalizers can be described as follows. In mathematical form, the function of the equalizer is

$$\hat{s}(k-d) = h(x(k), x(k-1), \cdots, x(k-n+1)), \quad \text{where} \quad h\colon \Re^n \to \{-1, 1\}. \tag{50}$$

Let the possible channel noise-free output vectors

$$\hat{\boldsymbol{x}}(k) = [\hat{x}(k), \hat{x}(k-1), \cdots, \hat{x}(k-n+1)]^T \tag{51}$$

that are produced by sequences of channel inputs containing $s(k-d) = 1$ and $s(k-d) = -1$, be denoted by the set $P_{n,d}(1)$ and $P_{n,d}(-1)$, respectively, i.e.,

$$P_{n,d}(1) = \{\hat{\boldsymbol{x}}(k) \in \Re^n | s(k-d) = 1\} \tag{52}$$

$$P_{n,d}(-1) = \{\hat{\boldsymbol{x}}(k) \in \Re^n | s(k-d) = -1\}. \tag{53}$$

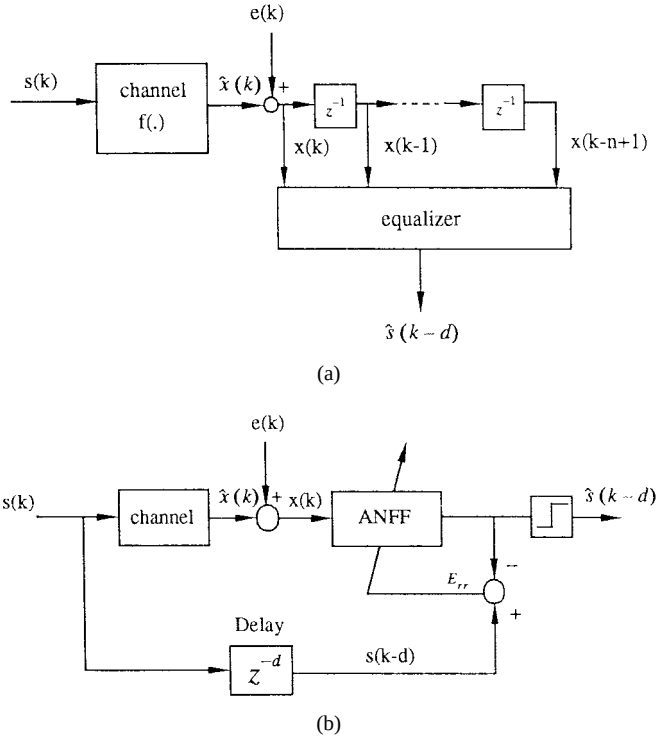


Fig. 7. (a) Schematic of a data transmission system. (b) ANFF as an adaptive equalizer.

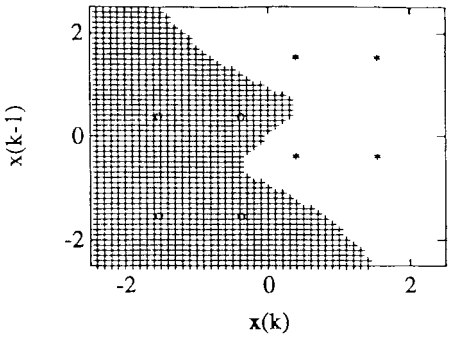


Fig. 8. Channel output points and optimal decision region in Example 1.

As shown in [15], the optimal equalizer, h_{opt}, which achieves the minimum bit error rate for a given order n and lag d is

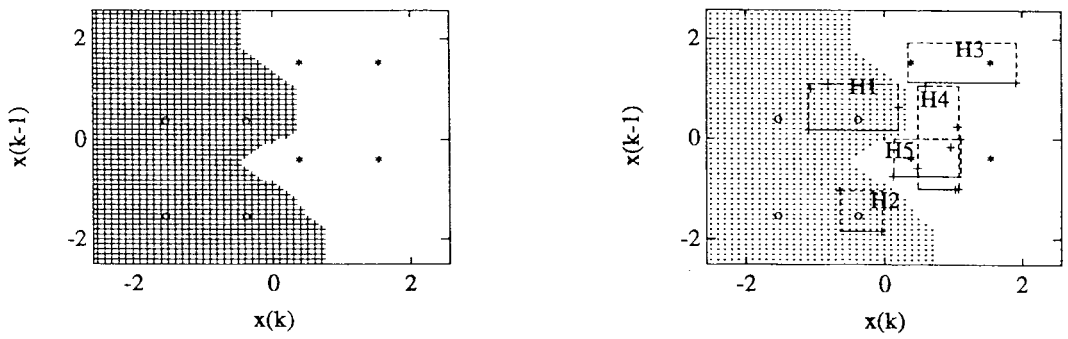
$$\hat{s}(k-d) = \begin{cases} 1, & \text{if } h_{\text{opt}}(\boldsymbol{x}(k)) \\ & = \text{sgn}[p_1(\boldsymbol{x}(k)) - p_{-1}(\boldsymbol{x}(k))] \\ & = 1 \\ -1, & \text{if } h_{\text{opt}}(\boldsymbol{x}(k)) \\ & = \text{sgn}[p_1(\boldsymbol{x}(k)) - p_{-1}(\boldsymbol{x}(k))] \\ & = -1 \end{cases} \tag{54}$$

where $p_1(\boldsymbol{x})$ and $p_{-1}(\boldsymbol{x})$ denote the conditional density functions of observing output $\boldsymbol{x}$ given $\hat{\boldsymbol{x}} \in P_{n,d}(1)$ and $\hat{\boldsymbol{x}} \in P_{n,d}(-1)$, respectively. The optimal decision boundary is thus the set of points that satisfy

$$h_{\text{opt}}(\boldsymbol{x}(k)) = 0, \quad \text{with} \quad \boldsymbol{x}(k) \in \Re^n. \tag{55}$$

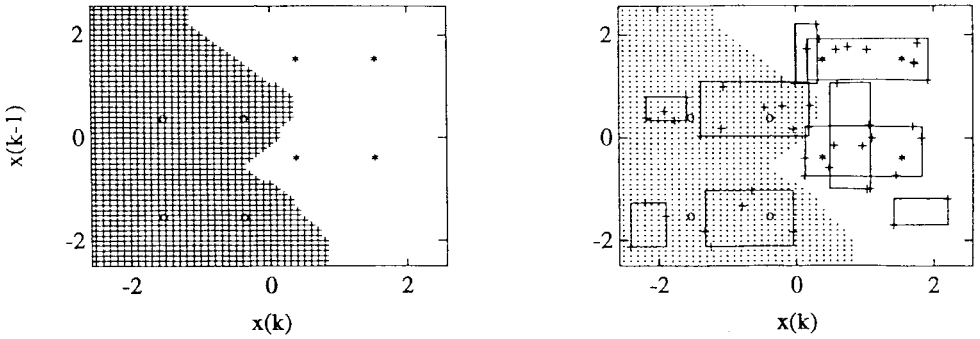
For the case that the noise $e(k)$ is Gaussian distributed with zero mean and covariance matrix

$$Q = E[(e(k), \cdots, e(k-n+1))(e(k), \cdots, e(k-n+1))^T] \tag{56}$$



(a) (b)

Fig. 9. (a) Decision region of the ANFF in Example 1 when the adaptation is stopped at $k = 17$, where five rules are generated. (b) The generated hyperboxes and the training data (denoted as "+").



(a) (b)

Fig. 10. (a) Decision region of the ANFF in Example 1 when the adaptation is stopped at $k = 50$, where nine rules are generated. (b) The generated hyperboxes and the training data (denoted as "+").

the optimal equalizer, h_{opt}, is

$$\text{sgn}\left[\sum_{\hat{\boldsymbol{x}}_+ \in P_{n,d}(1)} \exp\left[-\frac{1}{2}(\boldsymbol{x}(k)-\hat{\boldsymbol{x}}_+)^T Q^{-1}(\boldsymbol{x}(k)-\hat{\boldsymbol{x}}_+)\right] - \sum_{\hat{\boldsymbol{x}}_- \in P_{n,d}(-1)} \exp\left[-\frac{1}{2}(\boldsymbol{x}(k)-\hat{\boldsymbol{x}}_-)^T Q^{-1}(\boldsymbol{x}(k)-\hat{\boldsymbol{x}}_-)\right]\right]. \tag{57}$$

To demonstrate the performance of the ANFF's used as equalizers, different situations are illustrated in the following examples.

Example 1: Suppose the nonlinear channel function is

$$\hat{x}(k) = \hat{o}(k) - 0.9\hat{o}^3(k) \tag{58}$$

where $\hat{o}(k) = s(k) + 0.5s(k-1)$, and the noise $e(k)$ is white Gaussian distributed with $\sigma_e^2 = E[e^2(k)] = 0.2$. Then the covariance matrix Q used in (56) is

$$Q = \begin{bmatrix} 0.2 & 0 \\ 0 & 0.2 \end{bmatrix}.$$

If $n = 2$ and $d = 1$, then by (54) the optimal boundary can be derived and is shown in Fig. 8. In Fig. 8, the shaded region is the region where the transmitted signal is classified as 1. Also shown in the figure are the symbols "$\circ$" and "$*$" which denote the elements of the sets of $P_{2,0}(1)$ and $P_{2,0}(-1)$, respectively.

We now use the ANFF as an equalizer to solve the above problem. For training the ANFF, all input–output data should be normalized to be between 0 and 1. For the output, the two desired values, 1 and -1, are normalized as 0.75 and 0.25, respectively. Of course, other normalized values are allowed if they are within 0 and 1. Since the desired output value is either 0.75 or 0.25, one output node with two clusters centering at 0.75 and 0.25 is used. Since only two clusters are generated at the output node, we can simply set the output vigilance ρ_{out} as 1. The decided threshold at the output is set as 0.5 so that for the output whose value is larger than 0.5, the transmitted signal is classified as 1, otherwise it is classified as -1. The input vigilance ρ_{in} is set as 0.87 initially and is kept decreasing as training proceeds. The sensitivity parameter, γ, and learning constant, η, are chosen as $\gamma = 4$ and $\eta = 0.001$. There are no rules initially and they are generated during the training process. The simulation results (the decision boundaries) after the on-line training stopped at $k = 17$ and $k = 50$ are shown in Fig. 9 and Fig. 10 with the generated rule numbers being 5 and 9, respectively, where k denotes the number of time steps (sampling points). In Figs. 9(b) and 10(b), the training data, denoted as "+," and their corresponding generated hyperboxes which denote the number of rules are shown.

To explain the meaning of the above results, let us consider Fig. 9(b). In Fig. 9(b), five hyperboxes (or rules) marked as H_1, H_2, H_3, H_4, H_5 are shown. The five fuzzy rules that the

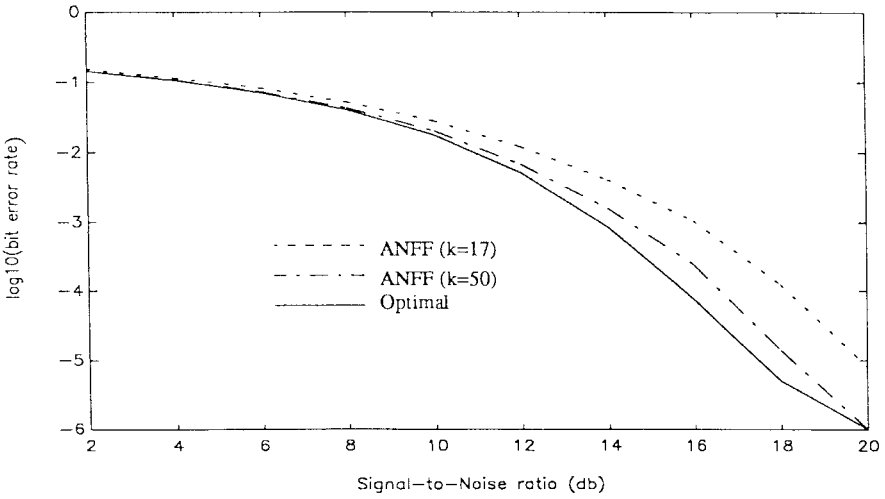


Fig. 11. Comparison of bit-error-rate curves for the optimal equalizer and the ANFF trained with $k = 17$ and $k = 50$ in Example 1.

hyperboxes stand for are

Rule 1: IF $\boldsymbol{x}$ is H_1, THEN y is O_1
Rule 2: IF $\boldsymbol{x}$ is H_2, THEN y is O_1
Rule 3: IF $\boldsymbol{x}$ is H_3, THEN y is O_2
Rule 4: IF $\boldsymbol{x}$ is H_4, THEN y is O_2
Rule 5: IF $\boldsymbol{x}$ is H_5, THEN y is O_2

where $\boldsymbol{x} = (x(k), x(k-1))^T$, and O_1 and O_2 are the two output clusters meaning the decided result being 1 or -1, respectively. The function of these rules is in fact to classify whether the received samples contains 1 or -1. From this point of view, the equalizer can be viewed as a classifier, and the problem can be considered as a classification problem [36]. In this perspective, the ANFF uses ART to do clustering in the input space and then maps the clusters to category "1" or "-1," a technique similar to fuzzy ARTMAP [31]. In contrast to fuzzy ARTMAP, where a cluster is merely a rectangle, a cluster in the ANFF is in fact a multidimensional membership function with membership value decaying outside the hyperboxes as shown in Fig. 4(b). The degree (or membership) a sample point belongs to each cluster is calculated and these degrees are then integrated via the defuzzification process to produce a final classification result. Moreover, the hyperboxes (clusters) are tuned optimally in the ANFF as mentioned in Section IV-C.

The results in Figs. 9 and 10 show that as training proceeds, the decision boundary has the tendency of converging to the optimal one. To see the actual bit-error-rate, a realization of 10^6 points of the sequences $s(k)$ and $e(k)$ are used to test the bit-error-rate of the equalizers. We stopped the training of the ANFF at $k = 17$ and 50. From Fig. 11, we can see that the curve of the bit-error-rate caused by the trained ANFF at $k = 50$ is very close to the optimal one. At higher signal-to-noise ratio (SNR), these two curves show a little more deviation. To explain this phenomenon, we take the case with SNR $= 18$ for example. At SNR $= 18$, the number of misclassified bits for the optimal filter are 2, while that for the ANFF trained after $k = 50$ are 14, for a total of 10^6 testing bits. Hence the deviation of the two bit-error-rate curves at high SNR in Fig. 11 is in fact caused by a small difference in the numbers of misclassified bits. Better performance can be achieved if more sampling points (k) are used for training.

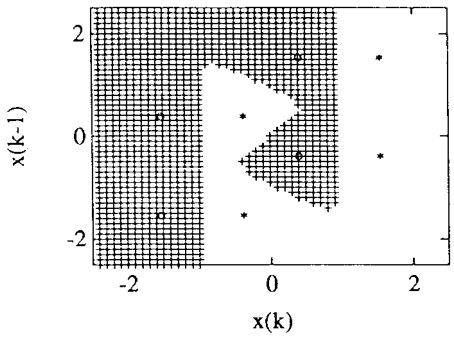
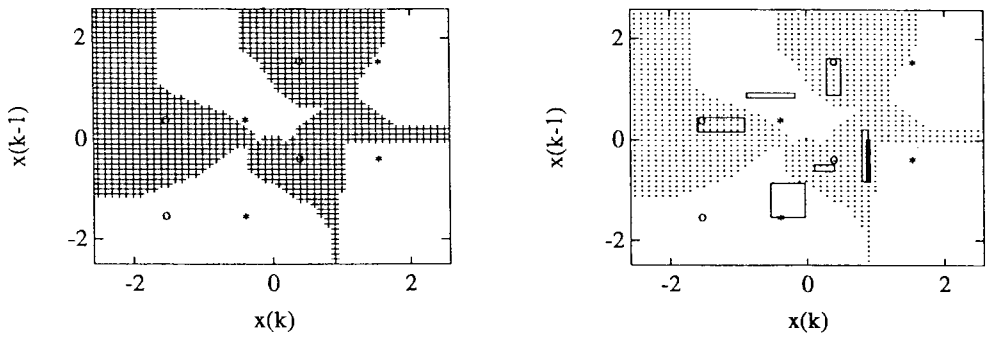


Fig. 12. Channel output points and optimal decision region in Example 2.

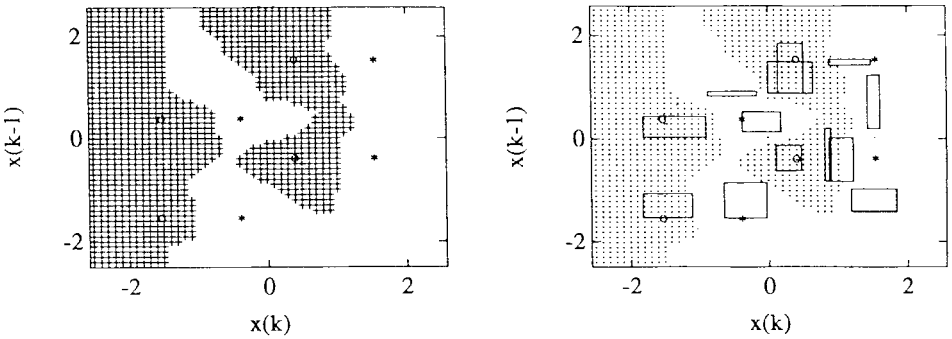
Example 2: The convergence of the ANFF to the optimal equalizer will be verified for a more complex case in this example. First, consider the same channel function as that in Example 1, except that $d = 0$ is used. The optimal decision boundary is shown in Fig. 12. From Fig. 12, we see that the decision boundary is highly nonlinear and the linear equalizer will not be able to handle it. By choosing input vigilance $\rho_{\text{in}} = 0.92$ and doing the same training job as in Example 1, we trained the ANFF and stopped the training at $k = 20$, 50, and 130, respectively. The corresponding decision boundaries and generated hyperboxes are shown in Figs. 13–15 with generated rule numbers being 10, 17, and 24, respectively. It should be noted that at the beginning of training, the hyperbox may be only a single point and is not clear in these figures. The tendency of converging to the optimal equalizer is verified in Figs. 13–15. Also from Figs. 13–15, we find that the ANFF takes only $k = 130$ for the vicinity of the sets, $P_{2,1}(1)$ and $P_{2,1}(-1)$, to be properly decided. The bit-error-rate curves of the ANFF trained for $k = 50$ and $k = 130$ are shown in Fig. 17.

For comparison, the LMS-type adaptive fuzzy filter [26] with 24 randomly chosen rules is used. By using the same training data and after $k = 500$ time steps of training, the



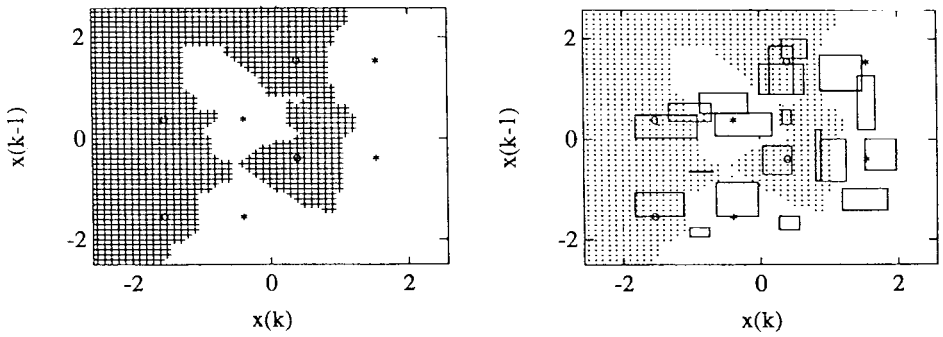
(a) (b)

Fig. 13. (a) Decision region of the ANFF in Example 2 when the adaptation is stopped at $k = 20$, where 10 rules are generated. (b) The generated hyperboxes.



(a) (b)

Fig. 14. (a) Decision region of the ANFF in Example 2 when the adaptation is stopped at $k = 50$, where 17 rules are generated. (b) The generated hyperboxes.



(a) (b)

Fig. 15. (a) Decision region of the ANFF in Example 2 when the adaptation is stopped at $k = 130$, where 24 rules are generated. (b) The generated hyperboxes.

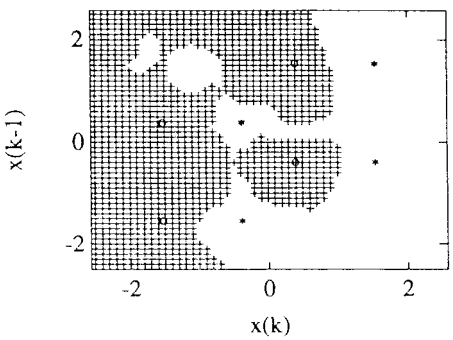


Fig. 16. Decision region of the LMS-type adaptive fuzzy filter with 24 rules in Example 2 when the adaptation is stopped at $k = 500$.

decision boundary and the bit-error-rate curve are shown in Fig. 16 and 17, respectively. Since the initial rules of this type of filter are randomly assigned, the training results may be different for different initial conditions. The result shown here is a better one of them. More training data (larger time steps k) are required for the LMS-type adaptive fuzzy filter to achieve the same performance of the ANFF, which required only 130 training time steps. This is due to the fact that the generation of fuzzy rules in the ANFF is based on the distribution of input data rather than on arbitrary assignment. The actual computation time of the ANFF trained for 130 time steps is 0.33 s, while for the LMS-type adaptive fuzzy filter trained

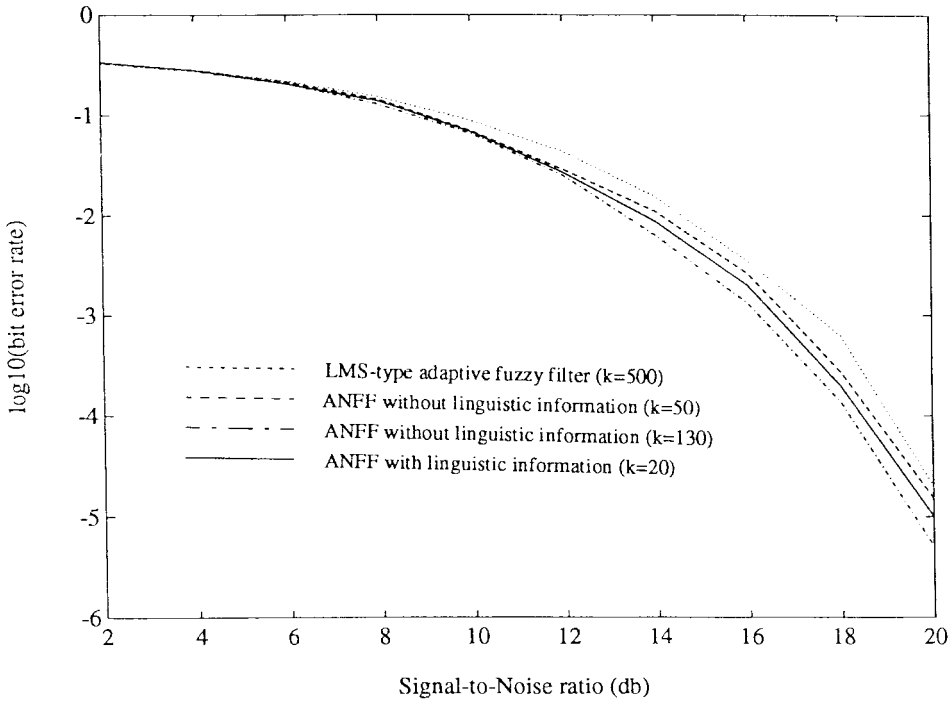


Fig. 17. Comparison of bit-error-rate curves for the ANFF trained with $k = 50$ and $k = 130$, where no linguistic information is incorporated, the LMS-type adaptive fuzzy filter trained with $k = 500$, and the ANFF trained with $k = 20$, where 10 initial fuzzy rules are incorporated.

for 500 time steps, the total computation time is 5.04 s. In average, for one step of training, the computation time of the LMS-type adaptive filter is three times of that of the ANFF. Moreover, for on-line training with a constant sampling rate, if the computation time of one step of training is less than the sampling time, then a less number of time steps k means a shorter training time is required, that is, a fast learning is achieved. The time steps required for the training of the ANFF is comparable with that of the RLS-type adaptive fuzzy filters in [26]; however, the RLS-type filter needs 81 fuzzy rules to achieve the same performance of the ANFF, that uses only 24 fuzzy rules in the example.

Example 3: As mentioned earlier, the ANFF can combine the training of numerical data and linguistic fuzzy if-then rules together. In Examples 1 and 2, we have demonstrated the training of the ANFF by numerical data. In this example, we will show the case that some expert knowledge is known in advance and we can incorporate such knowledge into the ANFF as is done in [26], where the combination of numerical data and linguistic fuzzy if-then rules was first proposed for equalizer. Looking at the results in Examples 1 and 2, we find that the function of the equalizer is in fact to find the regions in the input space that are corresponding to the input sequences containing 1 or -1. If in some specific situations, there are experts who know roughly the decision boundary and can assign degrees to some input regions to reflect their belief that the regions should belong to the "1" or "-1" category, then we can incorporate such knowledge into the ANFF to improve its learning speed. Consider, for example, the case in Fig. 12. If there are experts who are familiar with this case and can draw fuzzy regions, we can then assign fuzzy rules to these fuzzy regions.

In constructing the ANFF, the assignment of fuzzy rules is merely to draw the input and output hyperboxes in the input and output spaces according to the preconditions and consequents of the given fuzzy rules, and then connect the input–output hyperboxes according to the given if-then relationship. For the uncertain regions no assignment is required, because as numerical training proceeds new rules will be generated if necessary. Thus, unlike the scheme used in [26], the initial number of assigned rules is not required to be equal to the final total number of rules used in the filter, so more flexibility is allowed in our ANFF. The assigned hyperboxes are shown in Fig. 18(a). Since the regions are very rough, the mean values (center points) are assigned as 0.4 and -0.4 for category 1 and -1, respectively, to reflect the confidence. Without numerical training data, the decision boundary decided by the given 10 fuzzy rules are shown in Fig. 18(b). From Fig. 18(b), we see that the decision boundary is correct for the rightmost or the leftmost parts, but not good for the middle part. This result reflects that the expert knowledge is rough and can only provide partial information. To achieve more precise decision, training via numerical data is still necessary. Using the same training data as in Fig. 13, and stopping the training at $k = 20$, the decision region of the trained ANFF is shown in Fig. 19. The result shows that only 20 time steps $(k = 20)$ are required to properly classify the vicinities of the sets, $P_{2,0}(1)$ and $P_{2,0}(-1)$, and the bit-error-rate curve is shown in Fig. 17. The improvement of learning speed by incorporating expert fuzzy rules is verified from these curves.

B. Application to Adaptive Noise Cancellation

The ANFF is applied for adaptive noise cancellation in this subsection [32]. Adaptive noise cancellation is concerned with

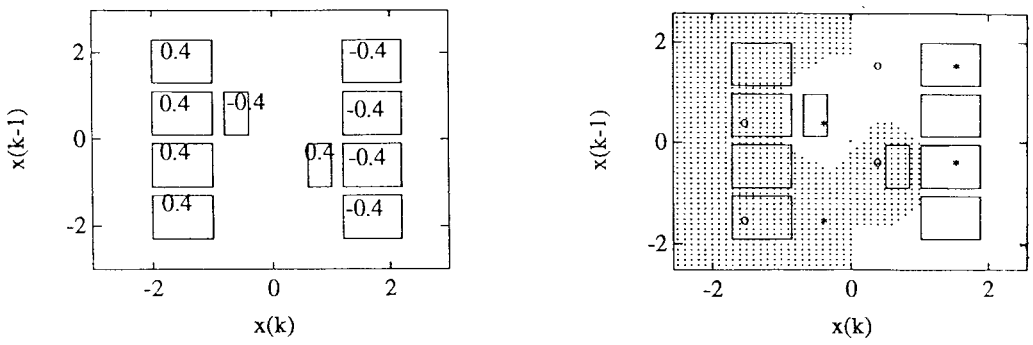


Fig. 18. (a) The assigned hyperboxes and their degree of confidence in Example 3. (b) The decision region generated by the rules from expert knowledge.

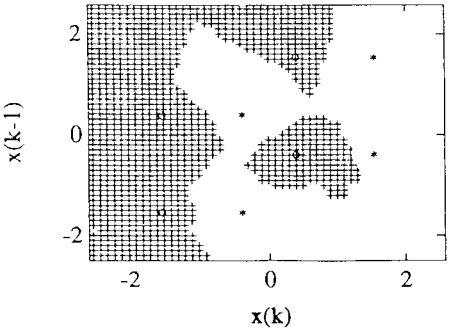


Fig. 19. Decision region of the ANFF in Example 3 with 10 initial fuzzy rules when the adaptation is stopped at $k = 20$.

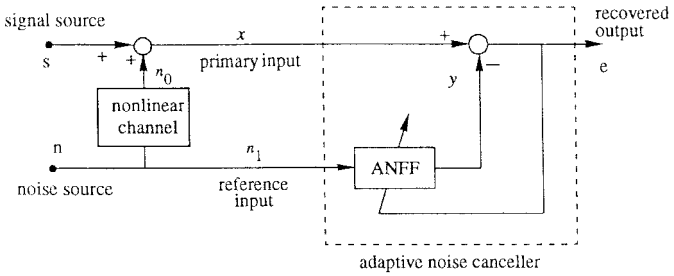


Fig. 20. Flow diagram of using ANFF for solving the adaptive noise cancelling problem.

the enhancement of noise corrupted signals and is based upon the availability of a primary input source and an auxiliary (reference) input source located at the noise field which contains no or little signal as shown in Fig. 20. In Fig. 20, the primary input source contains the desired signal s, which is corrupted by noise n_0 generated from the noise source n. The received signal is thus

$$x(k) = s(k) + n_0(k). \tag{59}$$

The secondary or auxiliary (reference) input source receives the noise n_1, which is correlated with the corrupting noise, n_0. The principle of the adaptive noise cancellation techniques is to adaptively process (by adjusting the filter's weights) the reference noise n_1 to generate a replica of n_0, and then subtract the replica of n_0 from the primary input x to recover the desired signal, s. We denote the replica of n_0, i.e., the adaptive filter output, as process y. To show how the system works, we will follow what is derived in [32]. In [32], the assumptions that s, n_0, and n_1 are stationary zero-mean processes, s is uncorrelated with n_0 and n_1, and n_0 and n_1 are correlated, are made. Also, the reference input source is situated in such a position that it detects only the noise not the signal s. Here, another constraint that process y is uncorrelated with process s is added due to the use of nonlinear adaptive filters. From Fig. 20, we have

$$e(k) = s(k) + n_0(k) - y(k). \tag{60}$$

By squaring and taking expectation of both sides, we can obtain

$$E[e^2(k)] = E[s^2(k)] + E[(n_0(k) - y(k))^2]. \tag{61}$$

Our objective is to minimize $E[(n_0(k) - y(k))^2]$. Observing (61), we can see that this objective is equivalent to minimizing $E[e^2(k)]$, and when

$$E[(n_0(k) - y(k))^2] = E[(n_0(k) - F(n_1(k))^2]$$

approaches zero, the remaining error $e(k)$ is in fact the desired signal $s(k)$, where $F(\cdot)$ represents the function of the nonlinear adaptive filter.

Traditionally, the design of the adaptive filters for the aforementioned noise cancelling problem is based upon a linear filter adapted by the LMS or RLS algorithm. In real situations, the environment between n and n_0 or n and n_1 is so complex that n_0 or n_1 is in fact a nonlinear function of n [33]–[35]. Higher performance of noise cancellation by using a nonlinear filter can thus be expected. To verify this and show the advantage of using the ANFF for adaptive noise cancellation, the following examples are given.

Example 4: Consider the case where the primary input source $s(k) = \sin(0.06k)\cos(0.01k)$ as shown in Fig. 21(a), and the noise is generated by a white noise n. Assume that the relation between the noise source n and the corrupting noise n_0 is a nonlinear function as

$$n_0(k) = 0.6(n(k))^3 \tag{62}$$

and the reference input is placed just in front of the noise source so that we have $n_1(k) = n(k)$. The performance of the

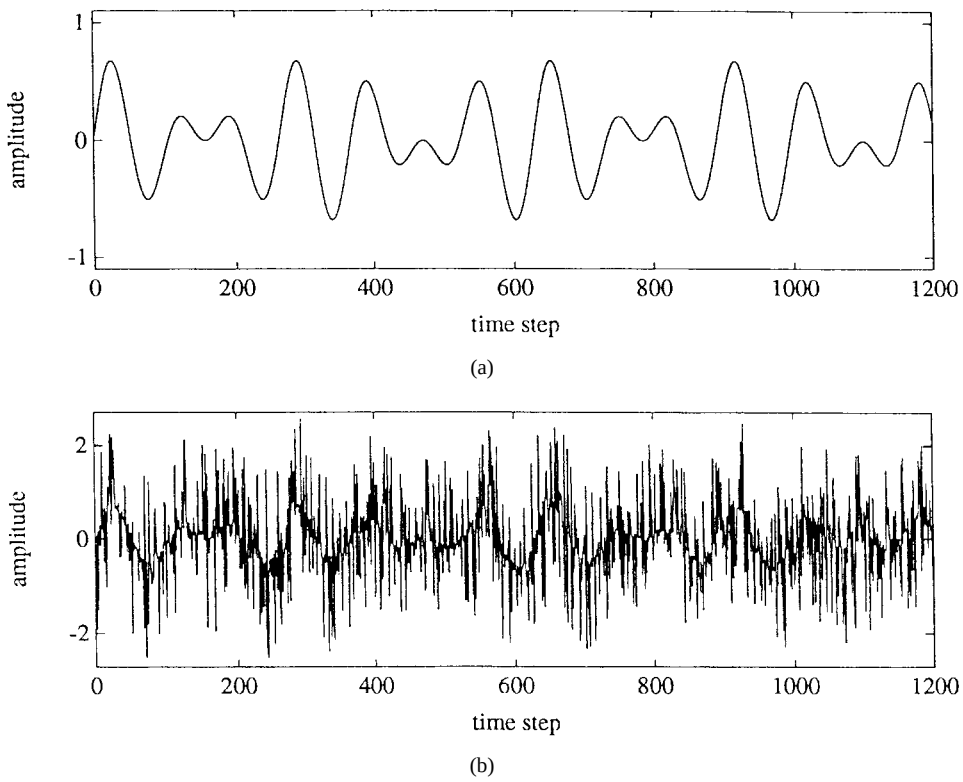


Fig. 21. (a) The signal source s in Example 4. (b) The corrupted signal x.

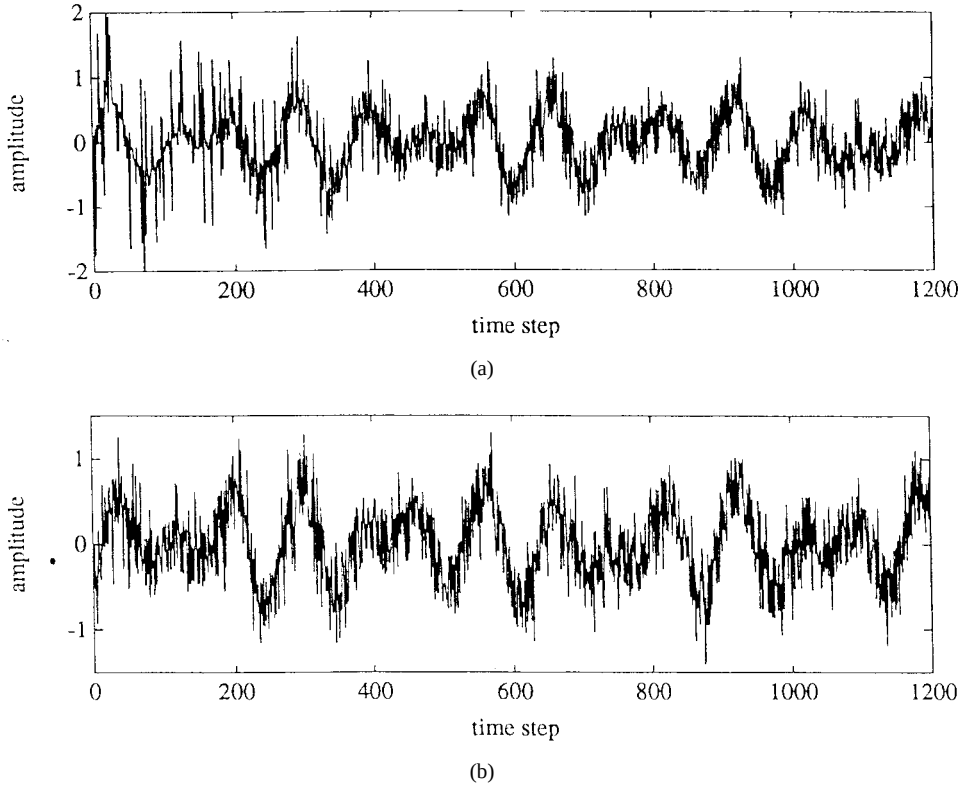


Fig. 22. (a) The recovered signal using the linear filter with order 0 during the first epoch of training. (b) The recovered signal using the linear filter with order 0 after convergence (see Example 4).

linear filter with transfer function

$$H(z) = \sum_{i=0}^{n} h_i z^{-i} \tag{63}$$

where n is the order of the filter, is investigated first. Linear filters with order 0, 1, and 2 are tested by setting all initial weights h_i equal to 0.1. By using the LMS algorithm with learning constant $\eta = 0.005$, we find that for order one and

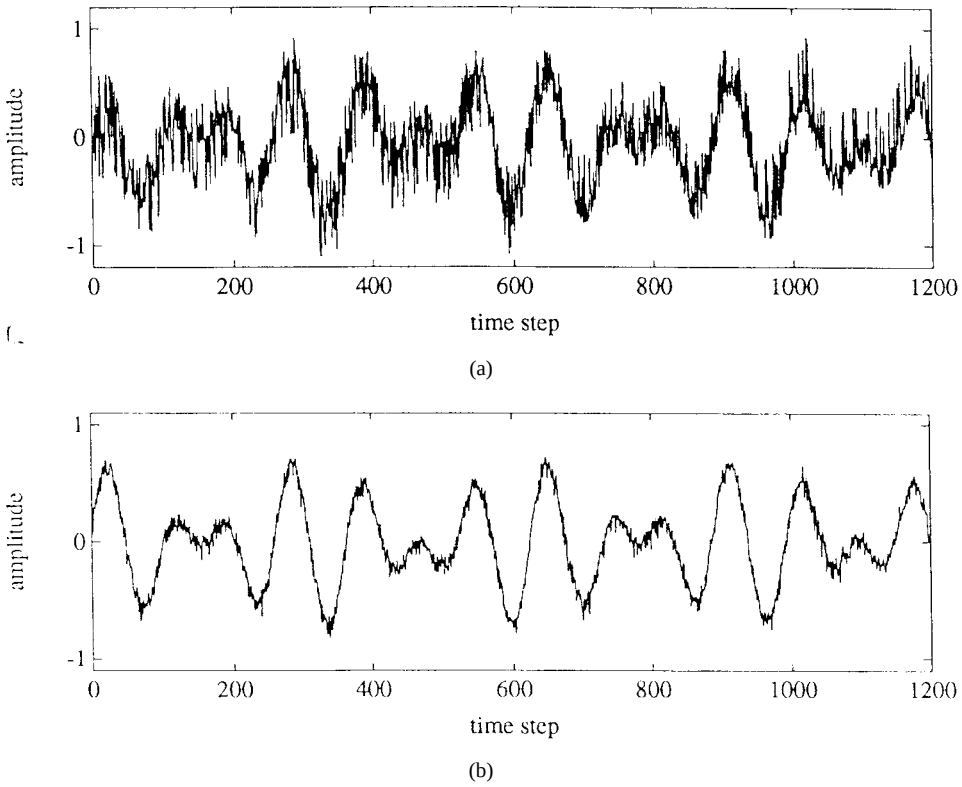


Fig. 23. (a) The recovered signal using the ANFF without linguistic information during the first epoch of training. (b) The recovered signal using the ANFF without linguistic information after convergence, where seven rules are generated (see Example 4).

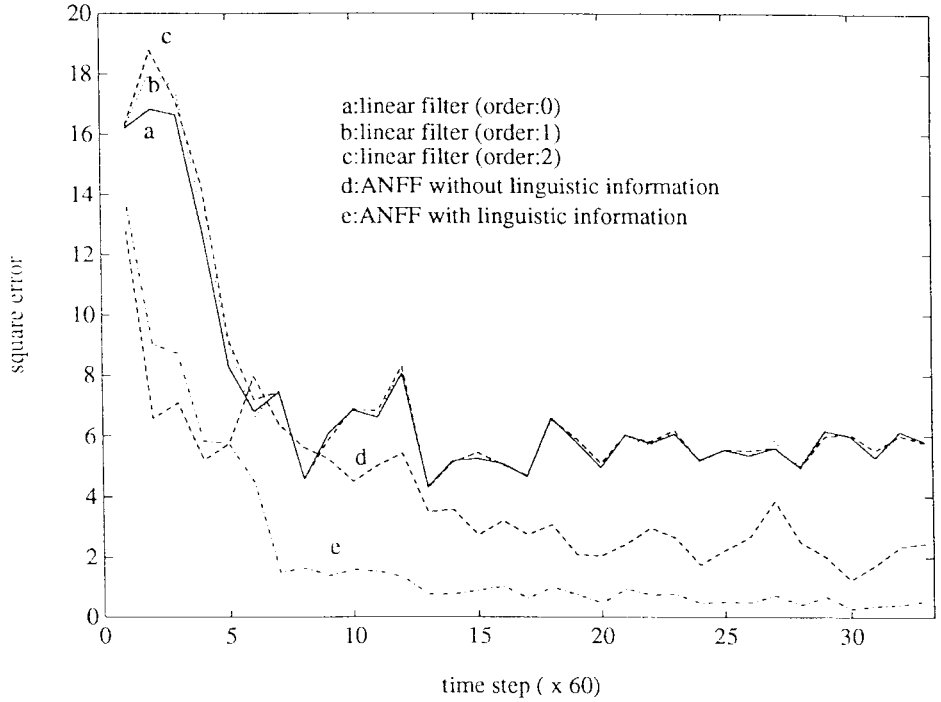


Fig. 24. Comparison of square error between the recovered signal and original signal for different orders of linear filters (a), (b), and (c), ANFF without linguistic information (d), and ANFF with three initial fuzzy rules (e) for the channel function in (63), with each plotted error value being the sum of errors over 60 adjacent time steps from the start of adaptation (see Example 4).

two, the weights h_1, h_2 all decay to zero, and the weight h_0 converges to around 0.8 for all the three linear filters. This shows that a linear filter with order zero is enough. The recovered signals during the first epoch of training and after convergence by using the linear filter with order zero are shown in Fig. 22(a) and (b), respectively. The results indicate that the performance of the linear filters is poor, and the development of nonlinear filters is thus necessary.

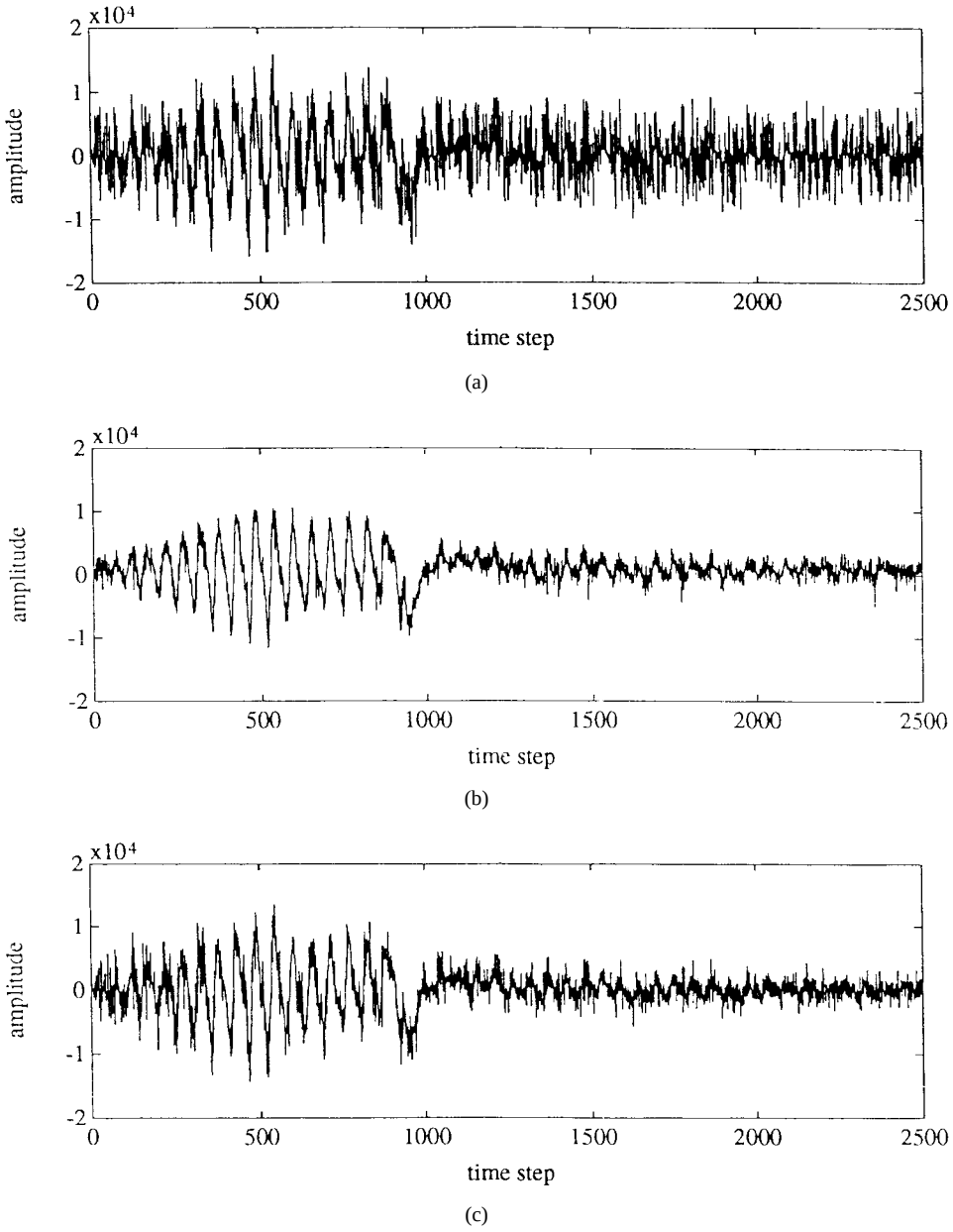


Fig. 25. Illustrations of Example 5. (a) Noise corrupted speech signal. (b) Recovered speech signal using the ANFF. (c) Recovered speech signal using linear filter.

We next apply the ANFF to this example. Before applying the ANFF for the adaptive noise cancellation, one thing should be noted. As stated in Section III, the input and output of the ANFF should be normalized to be between 0 and 1. This normalization will violate the assumption that processes s, n_0, and n_1 should be of zero mean, and thus the derivations of the above equations will not be valid anymore. To keep the assumption true, we will perform the normalization only in the input–output space partition process. When the partition process is finished, we then denormalize the corners of the hyperboxes as well as the input–output training data. Thus, in the training process, the zero-mean assumption holds.

The adaptation of the ANFF is performed after choosing the initial vigilances as $\rho_{\text{in}} = 0.4, \rho_{\text{out}} = 0.7$, and learning parameters as $\eta = 0.01, \gamma = 4$. A total of 1200 training data are used. The recovered signals during the first epoch of training and after 100 epochs of training are shown in Fig. 23(a) and (b), respectively. After 100 epochs of training, seven fuzzy rules are generated in the ANFF. From Fig. 23, we find that during the first epoch of training the initial error (the recovered signal) is not so large as the one obtained from linear filter. This phenomenon is due to the fact that in the early stage of the ANFF training, structure learning has learned the matching of input–output clusters quite well, so the error is small. Once the structure or rule base is constructed, fine tuning is performed continuously to minimize the error of the ANFF. As compared to the performance of the linear filters, the recovered signal through the use of the trained ANFF is much closer to the original signal.

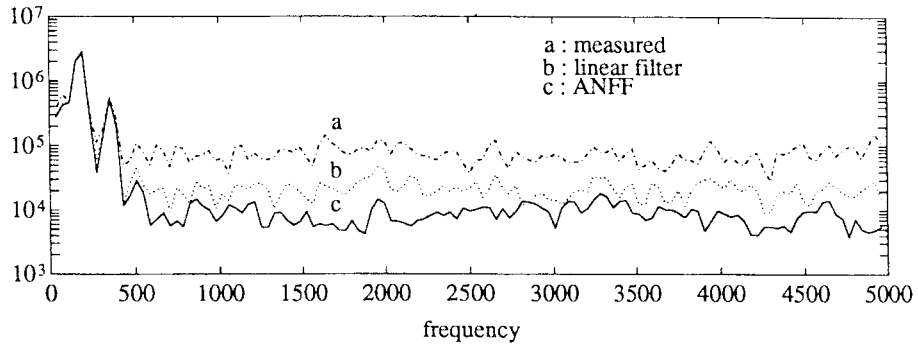


Fig. 26. Power spectrum of the noisy speech signal and recovered speech signal using linear filter and ANFF.

To make the performance comparison more precise, we calculate the square error between the recovered signal and the original signal for various filters. Fig. 24 shows the error curves for three different orders of linear filters [curves (a), (b), and (c)], and for the ANFF trained in the above—the ANFF without initial fuzzy rules [curve (d)]. In the figure, the error is calculated from the start of adaptation, and each plotted error value is the sum of square errors over 60 adjacent time steps. Also shown in the figure is the error curve for the ANFF with initial fuzzy rules, which will be explained in the following. The error curves showed in Fig. 24 indicate that the error of the ANFF is much smaller than that of the linear filters.

Observing the above results, we see that the adaptation of the filter is in fact to find a nonlinear function between n_0 and n_1. If we know the characteristics of the channels, we can design the filter directly. However, in real situations, the precise channel function is unknown and will change with time, so the direct design approach is nearly infeasible. Even though, if we know approximately the characteristic of the channel function, we can add the *a priori* fuzzy knowledge to the ANFF to improve its learning speed. Consider the noise cancellation problem in this example again [see (62)]. If we know roughly that the corrupting noise n_0 is proportional to the reference measured noise n_1. (Under some situations, this is a reasonable relationship, since if the amplitude of the noise source, n, is large, the measured noises n_0 and n_1 are usually both large. On the contrary, if the amplitude of the noise source, n, is small, the measured noises n_0 and n_1 are usually both small.) We can then add the following fuzzy rules in the ANFF

IF n_1 is High,	THEN	n_0 is High,
IF n_1 is Middle,	THEN	n_0 is Middle,
IF n_1 is Low,	THEN	n_0 is Low.

In the normalized domain, the two corners of the trapezoidal membership functions corresponding to the fuzzy terms "High," "Middle," and "Low" are (0.05, 0.3), (0.35, 0.65), and (0.6, 0.9), respectively. These three fuzzy rules are constructed in the ANFF initially. Then, after doing the same training job as in the above, the error curve is shown in Fig. 24 [curve (e)], where a total of six rules are generated. Fig. 24 shows that higher learning speed is achieved if *a priori* knowledge is incorporated into the ANFF.

Example 5: In previous examples, the signal used is a sinusoidal signal. In real applications, the signal is usually very complex such as a human's speech signal. To demonstrate the performance of the ANFF in a real-world case, consider the channel function used in (62) with the signal s being a voice utterance "e" sampled at 10 kHz and is corrupted with white noise. The measured signal, $s + n_0$, is shown in Fig. 25(a), where SNR $= -1.1$ dB. After five epochs of training, the recovered signal, $\hat{s}$, by using the ANFF (where eight rules are generated) is shown in Fig. 25(b). Of course, a better result can be achieved if more epochs of training are performed. Fig. 25(c) shows the recovered signal by using the linear filter trained until convergence. The aim of the adaptive filtering can be regarded as to minimize the power spectrum of the measured signal. The output power spectrums of the corrupted and recovered signals using the ANFF and linear filter is shown in Fig. 26. Since the corrupting noise is white, minimization over almost all frequency range is seen. From the comparison of power spectrums, high performance and learning speed are indeed achieved by using the ANFF.

VI. Conclusion

An ANFF is developed in this paper. The ANFF can be trained by numerical data and linguistic information expressed by fuzzy if-then rules. This feature makes the incorporation of *a priori* knowledge into the design of filters possible. Another key feature of the ANFF is that, without any given initial structure, the ANFF can construct itself automatically from numerical training data. Especially, the irregular-type partitioning on the input–output space can avoid the combinatorial growing problem existing in the conventional grid-type partitioning approach of fuzzy systems. Good performance is achieved by applying the ANFF to the nonlinear channel equalization and adaptive noise cancellation problems. More applications, like the noisy speech recognition and noisy image filtering, will be investigated.

References

[1] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1985.

[2] S. Benedetto and E. Biglieri, "Nonlinear equalization of digital satellite channels," presented at the *9th AIAA Conf. Comm. Satellite Syst.*, San Diego, CA, Mar. 1982.

[3] D. D. Falconer, "Adaptive equalization of channel nonlinearities in QAM data transmission systems," *Bell Syst. Tech. J.*, vol. 57, pp. 2589–2611, Sept. 1978.
[4] O. Agazzi, D. G. Messerschmitt, and D. A. Hodges, "Nonlinear echo cancellation of data signals," *IEEE Trans. Commun.*, vol. COMM-30, pp. 2421–2433, Nov. 1982.
[5] S. Moon and T. N. Hwang, "Coordinated training of noise removing networks," *Proc. Int. Conf. Acoust., Speech, Signal Processing*, 1993, pp. 573–576.
[6] I. Pitas and A. N. Venetsanopoulos, *Nonlinear Digital Filters*. Boston, MA: Kluwer, 1989.
[7] M. Bellanger, *Adaptive Digital Filters and Signal Analysis*. New York: Marcel Dekker, 1987.
[8] J. H. Lin, T. M. Sellke, and E. J. Coyle, "Adaptive stack filtering under the mean absolute error criterion," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 38, pp. 934–954, June 1990.
[9] J. H. Lin and E. J. Coyle, "Minimum mean absolute error estimation over the class of generalized stack filters," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 38, pp. 663–678, Apr. 1990.
[10] B. Widrow and R. Winter, "Neural nets for adaptive filtering and adaptive pattern recognition," *Computer*, pp. 25–39, Mar. 1988.
[11] S. Tamura and A. Waibel, "Noise reduction using connectionist models," in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, New York, Apr. 1988, pp. 553–556.
[12] H. B. D. Sorensen, "A cepstral noise reduction multilayer neural network," in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, 1991, pp. 933–936.
[13] G. J. Gibson, S. Siu, and C. F. N. Cowan, "The application of nonlinear structures to the reconstruction of binary signals," *IEEE Trans. Signal Processing*, vol. 39, pp. 1877–1884, Aug. 1991.
[14] K. A. Al-Mashouq and I. S. Reed, "The use of neural nets to combine equalization with decoding for severe intersymbol interference channels," *IEEE Trans. Neural Networks*, vol. 5, pp. 982–988, Nov. 1994.
[15] S. Chen, G. J. Gibson, C. F. N. Cowan, and P. M. Grant, "Reconstruction of binary signals using an adaptive radial-basis-function equalizer," *Signal Process.*, vol. 22, pp. 77–93, 1991.
[16] L. Yin, J. Astola, and Y. Neuvo, "A new class of nonlinear filters—neural filters," *IEEE Trans. Signal Processing*, vol. 41, pp. 1201–1222, Mar. 1993.
[17] T. H. J. Lo, "Synthetic approach to optimal filtering," *IEEE Trans. Neural Networks*, vol. 5, pp. 803–811, Sept. 1994.
[18] G. G. Towell, J. W. Shavlik, and M. O. Noordewier, "Refinement of approximate domain theories by knowledge-based neural networks," in *Proc. AAAI*, 1990, pp. 861–866.
[19] Q. Yang and V. K. Bhargava, "Building expert systems by a modified perceptron network with rule-transfer algorithms," in *Proc. IJCNN*, San Diego, CA, 1990, vol. 2, pp. 77–82.
[20] R. C. Lacher, S. I. Hruska, and D. C. Kuncicky, "Back-propagation learning in expert networks," *IEEE Trans. Neural Networks*, vol. 3, pp. 62–72, Jan. 1992.
[21] S. Horikawa, T. Furuhashi, and Y. Uchikawa, "On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm," *IEEE. Trans. Neural Networks*, vol. 3, pp. 801–806, Sept. 1992.
[22] C. T. Lin and C. S. G. Lee, "Neural-network-based fuzzy logic control and and decision system," *IEEE Trans. Comput.*, vol. 40, no. 12, pp. 1320–1336, 1991.
[23] J. S. Jang, "ANFIS: Adaptive-network-based fuzzy inference system," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, pp. 665–685, May 1993.
[24] L. X. Wang and J. M. Mendel, "Back propagation fuzzy systems as nonlinear dynamic system identifiers," in *Proc. IEEE Int. Conf. Fuzzy Systems*, San Diego, CA, 1992, pp. 1409–1418.
[25] H. Ishibuchi, R. Fujioka, and H. Tanaka, "Neural networks that learn from fuzzy if-then rules," *IEEE Trans. Fuzzy Syst.*, vol. 1, pp. 85–87, May 1993.
[26] L. X. Wang and J. M. Mendel, "Fuzzy adaptive filters with application to nonlinear channel equalization," *IEEE Trans. Fuzzy Syst.*, vol. 1, pp. 161–170, Aug. 1993.
[27] B. Kosko, "Fuzzy systems as universal approximators," in *Proc. IEEE Int. Conf. Fuzzy Systems*, San Diego, CA, Mar. 8-12, 1992, pp. 1153–1162.
[28] L. X. Wang, "Fuzzy systems are universal approximators," in *Proc. IEEE Int. Conf. Fuzzy Systems*, San Diego, CA, Mar. 8–12, 1992, pp. 1163–1170.
[29] C. J. Lin and C. T. Lin, "An ART-based fuzzy adaptive learning control network," in *Proc. IEEE Int. Conf. Fuzzy Systems*, Orlando, FL, 1994, pp. 1–6.
[30] G. A. Carpenter, S. Grossberg, and D. B. Rosen, "Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system," *Neural Networks*, vol. 4, pp. 759–771, 1991.
[31] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen, "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps," *IEEE Trans. Neural Networks*, vol. 3, pp. 698–712, Sept. 1992.
[32] B. Widrow *et al.*, "Adaptive noise cancellation: Principles and applications," *Proc. IEEE*, vol. 63, pp. 1691–1717, 1975.
[33] M. J. Coker and D. J. Simkins, "A nonlinear adaptive noise canceller," *IEEE Int. Conf. Acoust., Speech, Signal Processing*, pp. 470–473, 1980.
[34] J. C. Stapleton and G. Ramponi, "Adaptive noise cancellation for a class of nonlinear dynamic reference channels," in *Int. Symp. Circuits Systems*, 1984, pp. 268–271.
[35] S. A. Billings and F. A. Alturki, "Performance monitoring in nonlinear adaptive noise cancellation" *J. Sound Vibration*, vol. 157, no. 1, pp. 161–175, 1992.
[36] S. Theodoridis, C. M. S. See, and C. F. N. Cowan, "Nonlinear channel equalization using clustering techniques," in *IEEE Int. Conf. Commun.*, Chicago, IL, 1992, vol. 3, pp. 1277–1279.
[37] C. C. Lee, "Fuzzy logic in control systems: fuzzy logic controller—Parts I and II," *IEEE Trans. Syst., Man, Cybern.*, vol. 20, no. 2, pp. 404–435, 1990.
[38] B. Kosko, *Neural Networks and Fuzzy Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1992.
[39] P. K. Simpson, "Fuzzy min-max neural networks—Part II: Clustering," *IEEE Trans. Fuzzy Syst.*, vol. 1, no. 1, pp. 32–45, 1993.

Chin-Teng Lin (S'88–M'91) received the B.S. degree in control engineering from National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., in 1986, and the M.S.E.E. and Ph.D. degrees in electrical engineering from Purdue University, West Lafayette, IN, in 1989 and 1992, respectively.

Since August 1992, he has been with the College of Electrical Engineering and Computer Science, National Chiao-Tung University where he is currently an Associate Professor of Control Engineering. His current research interests are fuzzy systems, neural networks, intelligent control, human-machine interface, and video and audio processing. He is the co-author of *Neural Fuzzy Systems—A Neuro-Fuzzy Synergism to Intelligent Systems*, (Englewood Cliffs, NJ: Prentice-Hall), and the author of *Neural Fuzzy Control Systems with Structure and Parameter Learning*, (Singapore: World Scientific).

Dr. Lin is a member of Tau Beta Pi and Eta Kappa Nu. He is also a member of the IEEE Computer Society, the IEEE Robotics and Automation Society, and the IEEE Systems, Man, and Cybernetics Society.

Chia-Feng Juang received the B.S. degree in control engineering from National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., in 1993. He is currently pursuing the Ph.D. degree there.

His current research interests are neural networks, learning systems, fuzzy control, noisy speech recognition, and signal processing.