ELSEVIER

Fuzzy Sets and Systems 75 (1995) 17–31

FUZZY sets and systems

Implementation of a fuzzy inference system using a normalized fuzzy neural network

Chun-Tang Chao, Ching-Cheng Teng*

National Chiao-Tung University, Institute of Control Engineering, Hsinchu, Taiwan

Received May 1994; revised August 1994

Abstract

In this paper, we present a normalized fuzzy neural network (NFNN) to implement fuzzy inference systems. The proposed NFNN architecture makes an effective rule combination technique possible and thus enables us to significantly reduce the number of rules in the NFNN. We also derive a sufficient condition for rule combination and provide an algorithm to perform rule combination. Simulation results show that when combined with a rule elimination method the proposed rule combination method can greatly reduce the number of rules in the NFNN.

Keywords: Fuzzy inference system; Fuzzy neural network; Rule combination

1. Introduction

The main goal of a fuzzy inference system is to model human decision making within the conceptual framework of fuzzy logic and approximate reasoning [8]. As is well known, a fuzzy inference system consists of four important parts: the fuzzification interface, knowledge base unit, decision making unit, and output defuzzification interface. A fuzzy inference system is a model having the format of a fuzzy controller, which is the most thoroughly developed area of the application of fuzzy set theory in engineering [10].

The benefits of combining fuzzy logic and neural networks have been explored extensively in the literature, e.g., the fuzzy neural network in [8, 12], the adaptive-network-based fuzzy inference system in [9], and the fuzzy logical system in [16, 17]. The common advantages of the above systems are that (1) they can automatically and simultaneously identify fuzzy logical rules and tune the membership functions, and (2) the parameters of these systems have clear physical meanings, which they do not have in general neural networks. Fuzzy systems utilizing the learning capability of neural networks can successfully construct the input–output mapping for many applications. However, no efficient process for reducing the complexity of a fuzzy neural network has been presented.

Lin and Lee's [12] neural network-based fuzzy logic control and decision system provided criteria for rule combination to reduce the number of rules in a fuzzy neural network. Grant and Wal [5] also applied this

* Corresponding author.

rule combination method to eliminate redundant rules in their fuzzy neural network. However, they could not prove the general validity of their criteria for rule combination to the structure of their fuzzy neural networks. Moreover, no searching algorithm is presented in their papers for finding rules that can be combined.

To combine the benefits of a fuzzy logic system and a neural network [6, 7], in this paper we present a normalized fuzzy neural network (NFNN), a special type of fuzzy neural network, for implementing fuzzy inference systems. The normalization layer in the proposed NFNN makes rule combination in the fuzzy neural networks more practical and logical. Several definitions and concepts concerning multilevel logic synthesis and multiple-valued minimization [1, 13] are applied to obtain a sufficient condition for rule combination and to formulate a searching algorithm for rule combination. When used with existing fuzzy tools, the NFNN simplifies the knowledge acquisition stage and it can be used to create a fuzzy controller as in [5] or to identify an unknown system.

This paper is organized as follows. The NFNN and its operation are introduced in detail in Section 2. Section 3 describes the procedure for minimizing the rule set. A rule combination theorem and a practical algorithm for rule combination will be proposed in this section. In Section 4, an example is given to illustrate the application of the rule combination technique to the NFNN. The final section concludes the paper.

2. Fuzzy inference system and the NFNN

A typical format for a fuzzy rule base consists of a collection of fuzzy IF–THEN rules in the following form:

$$j\text{th rule:}\quad \text{IF } x_1 \text{ is } A_1^j, \ldots, \text{ and } x_n \text{ is } A_n^j, \text{ THEN } y = \beta^j, \tag{1}$$

where A_i^j and β^j are fuzzy sets in $U_i \subset R$ and $V \subset R$, respectively, and $\underline{x} = (x_1, \ldots, x_n)^{\mathrm{T}} \in U_1 \times \cdots \times U_n$ and $y \in V$ are the input and output of the fuzzy inference system, respectively. The first task to make use of a fuzzy inference system is to derive the deterministic fuzzy input–output mapping by defining the fuzzy logical rules and, more specifically, the membership functions of the fuzzy input and output sets associated with each rule. The class of fuzzy inference systems under consideration is a simplified type which uses a singleton to represent the output fuzzy set of each fuzzy logical rule. Thus β^j is the consequence singleton of the jth rule.

Let m be the number of fuzzy IF–THEN rules, that is, $j = 1, 2, \ldots, m$ in (1). The numerical output of the fuzzy inference system with *center average defuzzifier*, *product inference rule*, and *singleton fuzzifier* is of the following form:

$$y = \frac{\sum_{j=1}^{m} \beta^j (\prod_{i=1}^{n} \mu_{A_i^j}(x_i))}{\sum_{j=1}^{m} \prod_{i=1}^{n} \mu_{A_i^j}(x_i)}, \tag{2}$$

where $\mu_{A_i^j}$ denotes the membership function of fuzzy set A_i^j. This simplified fuzzy inference system has been shown to be a universal approximator [3] which is capable of approximating any real continuous function to any desired degree of accuracy, provided sufficiently many fuzzy logical rules are available [10].

2.1. The NFNN structure

In this subsection, we will construct a four-layer NFNN structure to implement the fuzzy inference system stated in (2). We first denote by A_{ij} the membership function of the jth term node of input variable x_i and assume that x_i has n_i term nodes for fuzzy partition. An NFNN structure with three input variables, two term nodes for each input variable, two output nodes, and eight rule nodes is illustrated in Fig. 1.

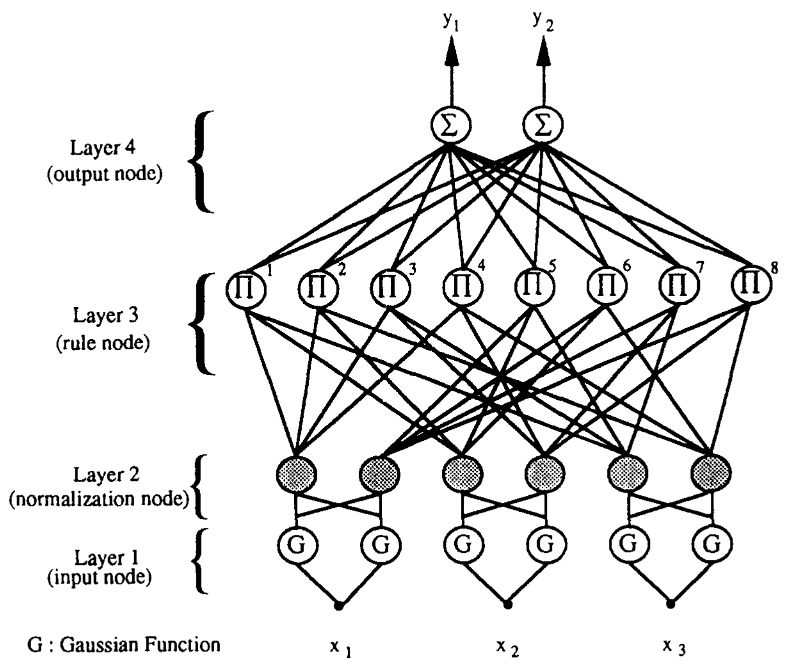

Fig. 1. The structure of the NFNN.

Layer 1: linguistic term layer

This layer uses a Gaussian function as a membership function, so the output of the jth term node associated with x_i is

$$\mu_{A_{ij}}(x_i) = \exp\left(-\left(\frac{x_i - m_{ij}}{\sigma_{ij}}\right)^2\right), \tag{3}$$

where m_{ij} and σ_{ij} denote the mean (center) and variance (width) of A_{ij}, respectively.

Layer 2: normalization layer

This layer performs a normalization procedure for the output of layer 1. Notice that no weight is adjusted here and that normalization has been done, i.e.,

$$\mu'_{A_{ij}}(x_i) = \frac{\mu_{A_{ij}}(x_i)}{\sum_{k=1}^{n_i} \mu_{A_{ik}}(x_i)}, \tag{4}$$

where $\mu'_{A_{ij}}(x_i)$ denotes the normalized output of $\mu_{A_{ij}}(x_i)$. The normalization procedure can also be represented in another form,

$$\mu'_{A_i^j}(x_i) = \frac{\mu_{A_i^j}(x_i)}{\sum_{k=1}^{n_i} \mu_{A_{ik}}(x_i)}. \tag{5}$$

This normalization procedure is crucial for rule combination, as will be explained in the next section.

Layer 3: rule layer

This layer implements the links relating preconditions (normalized node) to consequences (output node). The connection criterion is that each rule node has only one antecedent link from a normalized node of a linguistic variable. Hence there are $\prod_i n_i$ rule nodes in the initial form of NFNN structure. We mention that there is still no weight adjustment in this layer. The output of the jth rule node is

$$out_j^3 = \prod_{i=1}^{n} \mu'_{A_{ik}}(x_i), \tag{6}$$

where k is determined by the connection criterion, or, in another form,

$$out_j^3 = \prod_{i=1}^{n} \mu'_{A_i^j}(x_i). \tag{7}$$

Layer 4: output layer

All consequence links are fully connected to the output nodes and interpreted directly as the strength of the output action. This layer performs centroid defuzzification to obtain the numerical output:

$$y = \sum_{j=1}^{m} \beta^j \prod_{i=1}^{n} \mu'_{A_i^j}(x_i). \tag{8}$$

Thus, the overall net output is treated as a linear combination of the consequences of all rules instead of the complex composition of a rule of inference and the defuzzification process.

In the following, we will start from (8) and show that the output y of the NFNN system is equal to the output of the simplified fuzzy inference system stated in (2). To begin with, from the connection criterion between layers 2 and 3, we obtain the equation

$$\prod_{i=1}^{n} \sum_{j=1}^{n_i} \mu_{A_{ij}}(x_i) = \sum_{j=1}^{m} \prod_{i=1}^{n} \mu_{A_i^j}(x_i). \tag{9}$$

Substituting (5) into (8), we have

$$y = \sum_{j=1}^{m} \beta^j \prod_{i=1}^{n} \frac{\mu_{A_i^j}(x_i)}{\sum_{k=1}^{n_i} \mu_{A_{ik}}(x_i)} = \frac{\sum_{j=1}^{m} \beta^j (\prod_{i=1}^{n} \mu_{A_i^j}(x_i))}{\prod_{i=1}^{n} \sum_{k=1}^{n_i} \mu_{A_{ik}}(x_i)}. \tag{10}$$

Applying (9) to the denominator of the above equation, we can obtain the same result as in (2). This means that the proposed NFNN structure is equivalent to the simplified fuzzy inference system.

2.2. Supervised learning

The adjustment of the parameters in the proposed NFNN can be divided into two tasks, corresponding to the IF (premise) part and THEN (consequence) part of the fuzzy logical rules. In the premise part, we need to initialize the center and width for Gaussian functions. To determine these initial terms, a self-organization-map (SOM) [11] and fuzzy-c-means (FCM) [15] are commonly used. Another simple and intuitive method of doing this is to use normal fuzzy sets to fully cover the input space. Since the final performance will depend mainly on supervised learning, we choose normal fuzzy sets in this paper. In the consequence part, the parameters are output singletons. These singletons are initialized with small random values, as in a pure neural network.

A gradient-descent-based BP algorithm [14] is employed to adjust NFNN's parameters. The goal is to minimize the error function

$$E = \tfrac{1}{2}(d - y)^2, \tag{11}$$

where y is the output of the NFNN and d is the desired output for the ith input pattern. If w_{ij} is the adjusted parameter, then the learning rule is

$$w_{ij}(t+1) = w_{ij}(t) - \eta \frac{\partial E}{\partial w_{ij}} + \alpha \Delta w_{ij}(t)$$

and

$$\Delta w_{ij}(t) = w_{ij}(t) - w_{ij}(t-1), \tag{12}$$

where η is the learning rate and α, $0 < \alpha < 1$, is the momentum parameter.

Substituting (3)–(8) into (12), we obtain the back-propagated error signals δ and the update rules for the NFNN:

$$\delta^4 = d - y, \tag{13}$$

$$\delta_j^3 = \delta^4 \beta^j, \tag{14}$$

$$\delta_{ij}^2(t) = \sum_p \delta_p^3 out_p^3 - \mu'_{A_{ij}}(x_i) \sum_{k=1}^{n_i} \sum_q \delta_q^3 out_q^3, \tag{15}$$

$$\beta^j(t+1) = \beta^j(t) + \eta \delta^4 out_j^3 + \alpha \Delta \beta^j(t), \tag{16}$$

$$m_{ij}(t+1) = m_{ij}(t) + \eta \delta_{ij}^2 \frac{2(x_i - m_{ij})}{\sigma_{ij}^2} + \alpha \Delta m_{ij}(t), \tag{17}$$

$$\sigma_{ij}(t+1) = \sigma_{ij}(t) + \eta \delta_{ij}^2 \frac{2(x_i - m_{ij})^2}{\sigma_{ij}^3} + \alpha \Delta \sigma_{ij}(t), \tag{18}$$

where the subscripts p and q in (15) denote, respectively, all the rule nodes connected to the jth term node and the kth term node of x_i.

3. Rule combination

In general, a fuzzy neural system with more rules will take more parameters and will provide better performance. In fact, however, some of these rules are unnecessary or redundant. A rule elimination method is a method that eliminates unnecessary rules by simply abandoning rules with relatively small consequence weights. The purpose of a rule combination method, on the other hand, is (a) to eliminate redundant preconditions of fuzzy rules, and (b) to combine certain pairs of fuzzy logical rules into a single, logically equivalent rule. The reason rules can be combined is very clear. For example, if a continuous function is of the form $f(x_1, x_2) = x_2/((x_1 - 4)^2 + x_2)$, then if $x_1 = 4$, we have $f = 1$ for all x_2. Thus f is not affected by x_2 in this case, and x_2 is a redundant input when $x_1 = 4$.

In this section, we first introduce some basic definitions and concepts which will be helpful in deriving the sufficient condition for rule combination. We also provide a rule combination algorithm and give an example to clarify the main idea behind the proposed rule combination method.

3.1. Definitions

Many definitions of logic expressions were introduced in [1] to provide a means for multilevel logic synthesis. In order to treat a mathematical equation as a logic expression, we also have to state some definitions, as follows. We refer interested readers to [1,4] for details.

A variable can be thought of as a *literal* (e.g., a or b), and a *cube* represents the conjunction of its literals (e.g., a, abc, and bcd).

An *expression* is a set of cubes. For example, $y = abc + de + fg$ is an expression consisting of three cubes abc, de, and fg. We say an expression is *cube-free* if no cube divides the expression evenly (e.g., $ab + d$ is cube-free but $ab + ad$ is not, since $ab + ad$ can be divided evenly by a cube a).

The *primary divisors* of an expression f are the set of expressions

$$D(f) = \{f/C \mid C \text{ is a cube}\}.$$

For example, we may define an expression x as

$$\begin{aligned} x &= abe + bce + acf + bcf \\ &= e(ab + bc) + c(fa + fb) \\ &= be(a + c) + cf(a + b). \end{aligned} \tag{19}$$

Then $ab + bc$ and $fa + fb$ are the primary divisors of x and they are obtained by x/e and x/c, respectively. Also, $a + c$ and $a + b$ are the primary divisors of x.

The *kernels* of an expression f are the set of expressions

$$K(f) = \{g \mid g \in D(f) \text{ and } g \text{ is cube-free}\}.$$

In other words, the kernels of an expression f are the cube-free primary divisors of f. The cube C used to obtain the kernel $k = f/C$ is called the *co-kernel* of k. In (19), for example, $a + c$ and $a + b$ are kernels corresponding to co-kernels be and cf, respectively, since they are cube-free primary divisors of x. On the other hand, the primary divisors $ab + bc$ and $fa + fb$ are not kernels of x, because they are not cube-free. The kernels $a + c$ and $a + b$ are also called *level*-0 *kernels*, which have no kernels except themselves. We will use the notation $K^0(f)$ to represent the set of level-0 kernels of f.

Moreover, we define the literal o_j^i as the output of the normalized node corresponding to the jth term node of the ith variable x_i and define the expression O^i as follows:

$$O^i = \sum_{j=1}^{n_i} o_j^i = o_1^i + o_2^i + \cdots + o_{n_i}^i. \tag{20}$$

Notice that, for brevity, we use $\sum$ and $\prod$ to represent *logical sum* and *logical product*, respectively.

Without loss of generality, we consider multi-input–single-output fuzzy inference systems, since a multi-output system can always be decomposed into a group of single-output systems. Let y^* denote the partial summation of the final output y in (8) with the same consequence weight β^*. Then we have

$$y^* = \beta^* \left(\sum_{j=1}^{m'} \prod_{i=1}^{n} o_k^i \right),$$

where k is determined by the connection criterion between normalization nodes and rule nodes and m' is the number of rules with the same consequence weight β^*. When y^* is divided by β^*, we define another expression

$$y\_same = y^*/\beta^* = \sum_{j=1}^{m'} \prod_{i=1}^{n} o_k^i, \tag{21}$$

where *y_same* is expressed in *two-level* sum-of-product form.

3.2. *The rule combination theorem and algorithm*

Now we are ready to establish the theorem to determine whether rules can be combined in the NFNN system.

Rule Combination Theorem. *In the NFNN system, let O^i and y_same be defined as in* (20) *and* (21), *respectively, and let $K^0(y\_same)$ be the set of level-*0 *kernels of y_same. If there exists an $O^i \in K^0(y\_same)$, $1 \leqslant i \leqslant n$, then some rules can be combined into a single equivalent rule.*

Proof. If there exists an $O^i \in K^0(y\_same)$, $1 \leqslant i \leqslant n$, then the expression *y_same* can be represented in the form $y\_same = (co) * O^i + s\_o\_p$, where *co* is the co-kernel corresponding to O^i and the term *s_o_p* is the algebraic quotient of *y_same/co* in sum-of-product form. Since numerically O^i is equal to unity, i.e.,

$$O^i = \sum_{j=1}^{n_i} \mu'_{A_{ij}}(x_i) = \sum_{j=1}^{n_i} \frac{\mu_{A_{ij}}(x_i)}{\sum_{j=1}^{n_i} \mu_{A_{ij}}(x_i)} = 1,$$

where we have applied (4), *y_same* can be simplified as $y\_same = co + s\_o\_p$. This means that some rules can be combined into a single equivalent rule. □

With slight modification, the theorem stated above can also be extended to an MIMO system. The theorem tells us that instead of finding all the kernel sets we just need to check whether the expression *y_same* has level-0 kernel O^i. In practical application, we can repeatedly apply this theorem until no rules can be combined. An algorithm for rule combination is stated below.

Rule Combination Algorithm

```
Let the expression y_same be in sum-of-product form.
check = YES   /*YES == 1 NO == 0*/
$temp_i = 0$ for $i = 1$ to $n$
WHILE(check)
 {
  FOR $i = 1$ TO $n$
    IF literal $o_j^i$ appears in at least one cube of y_same for $j = 1$ to $n_i$
      WHILE(find(i))
       {
        $C = \sum_{k=1}^{n_i} (cube_k)$
          IF $cube_j$ divided by literal $o_j^i$ has the same quotient $q$ for $j = 1$ to $n_i$
            {
             $y\_same = y\_same - C + q$
             $temp_i = 1$
            }
       }
    IF $temp_i == 0$ for $i = 1$ to $n$
      $check$ = NO
        ELSE
          {
            $check$ = YES
            $temp_i = 0$ for $i = 1$ to $n$
          }
    }
find($i$)
  {
   IF we can find a "new" set of $n_i$ cubes, denoted by $cube_j$ ($j = 1$ to $n_i$), from
     $y\_same$ such that the $cube_j$ has literal $o_j^i$ for $j = 1$ to $n_i$
     return YES
       ELSE return NO
  }
```

The inner while loop of the algorithm checks whether O^i is the kernel of *y_same*; if it is, then *y_same* can be simplified. Since a kernel may have at least one corresponding co-kernel, we need the inner while loop. Moreover, the function find ( ) finds a new candidate C, which is several cubes in sum-of-product form, from *y_same*. Suppose C can be factored as a product of O^i and a cube q; then we replace C by q in the expression *y_same*. To ensure that O^i is not a kernel of *y_same*, we have the outer while loop in the algorithm.

Fig. 2 presents an example to illustrate the proposed algorithm. The rules of the original NFNN system with the same consequence weight are shown in Fig. 2(a). Then we have

$$y\_same = o_1^1 o_1^2 o_1^3 + o_1^1 o_1^2 o_2^3 + o_1^1 o_2^2 o_1^3 + o_2^1 o_1^2 o_2^3 + o_3^1 o_1^2 o_1^3 + o_3^1 o_1^2 o_2^3 + o_3^1 o_2^2 o_1^3. \tag{22}$$

We apply the rule combination algorithm first to check whether $O^1 = o_1^1 + o_2^1 + o_3^1$ is the kernel of *y_same*. The answer is "yes", and *y_same* becomes

$$\begin{aligned} y\_same &= o_1^2 o_2^3 (o_1^1 + o_2^1 + o_3^1) + o_1^1 o_1^2 o_1^3 + o_1^1 o_2^2 o_1^3 + o_3^1 o_1^2 o_1^3 + o_3^1 o_2^2 o_1^3 \\ &= o_1^2 o_2^3 + o_1^1 o_1^2 o_1^3 + o_1^1 o_2^2 o_1^3 + o_3^1 o_1^2 o_1^3 + o_3^1 o_2^2 o_1^3. \end{aligned} \tag{23}$$

Fig. 2(b) shows the result. Because O^1 is no longer a kernel of the new *y_same*, we try $O^2 = o_1^2 + o_2^2$ and repeat the above procedure. Then we obtain

$$\begin{aligned} y\_same &= o_1^2 o_2^3 + o_1^1 o_1^3 (o_1^2 + o_2^2) + o_3^1 o_1^2 o_1^3 + o_3^1 o_2^2 o_1^3 \\ &= o_1^2 o_2^3 + o_1^1 o_1^3 + o_3^1 o_1^2 o_1^3 + o_3^1 o_2^2 o_1^3. \end{aligned} \tag{24}$$

The resulting connection links in NFNN are shown in Fig. 2(c). We can still extract the kernel O^2 from the above expression, thus

$$\begin{aligned} y\_same &= o_1^2 o_2^3 + o_1^1 o_1^3 + o_3^1 o_1^3 (o_1^2 + o_2^2) \\ &= o_1^2 o_2^3 + o_1^1 o_1^3 + o_3^1 o_1^3. \end{aligned} \tag{25}$$

The result is illustrated in Fig. 2(d). Since the expression *y_same* never has the kernels O^2, O^3, and O^1, we stop the procedure and can guarantee that no further rules can be combined.

In fact, the basic idea behind the theorem and algorithm stated above is similar to the three criteria for rule combination presented in [12]: (1) the rules to be combined into a single rule node must have exactly the same consequences, (2) some preconditions are common to all these rules, and (3) the union of other

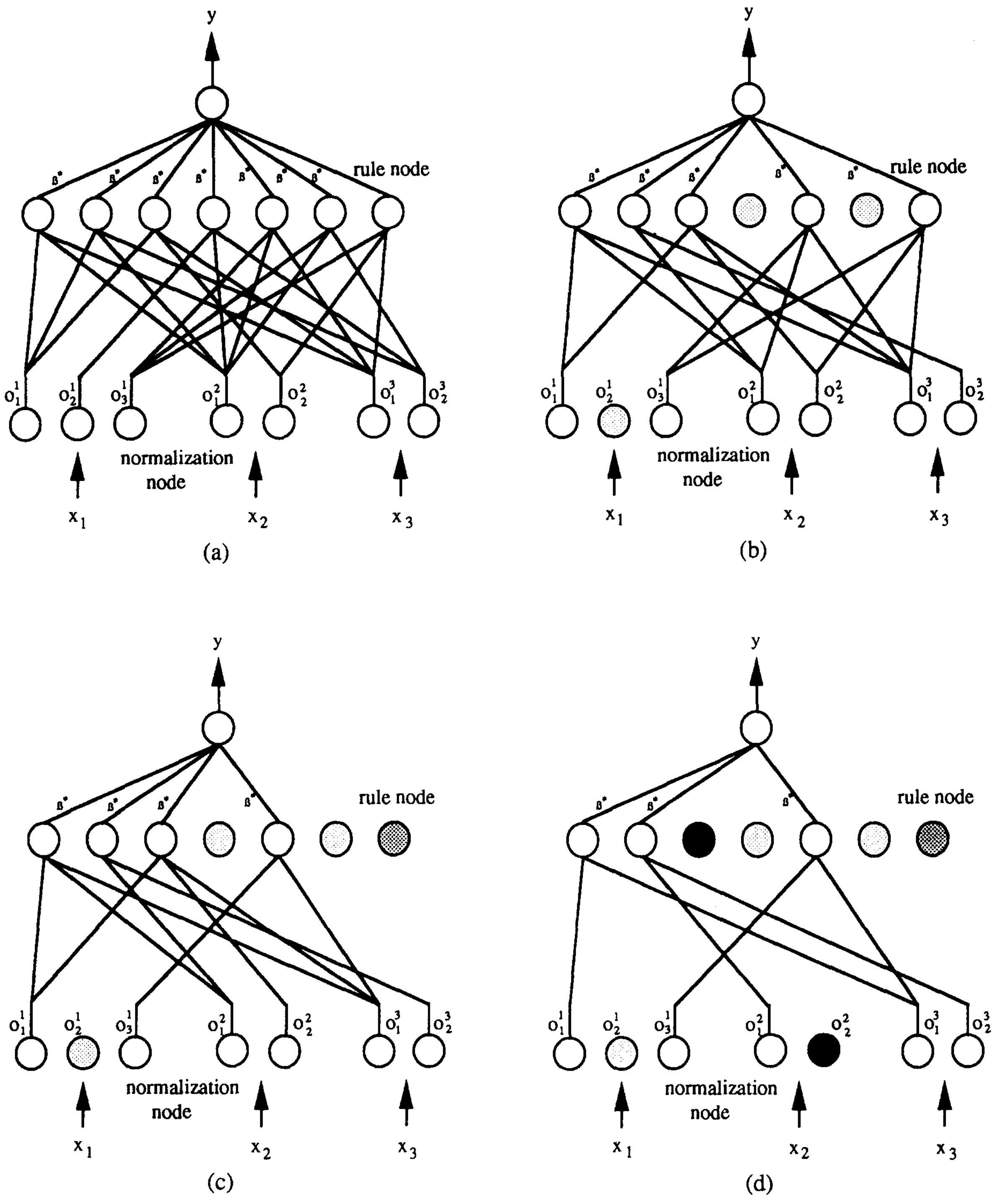

Fig. 2. Example illustrating rule combination.

preconditions of these rule nodes constitutes the entire term set of some input linguistic variables. The difference between their rule combination method and ours is as follows. First, they apply rule combination before supervised learning, while we do it after supervised learning. Second, in their method the "same consequences" referred to in criterion (1) are still fuzzy terms while in our method they are singleton weights. Third, the rules they are combined in their method are just a subset of the rules with the same consequences. They do not provide an efficient method for finding the rules to be combined, while we do. Fourth, the specially designed NFNN provides a proof that all rules that can be combined have been found, whereas their method offers no such proof.

3.3. *Minimization problem in rule combination*

Although the rule combination algorithm presented in the last subsection provides an intuitive and simple method for rule combination, it leaves an unsolved minimization problem. For example, the example in (22) has the final reduced form shown in (25) if we extract the kernels in the order O^1, O^2, and O^3. But if we extract the kernel O^3 first, we will have

$$y\_same = o_1^1 o_1^2 + o_1^1 o_2^2 o_1^3 + o_2^1 o_1^2 o_2^3 + o_3^1 o_1^2 + o_3^1 o_2^2 o_1^3, \tag{26}$$

which can be reduced no further. This means that the proposed rule combination algorithm could be extended to solve the minimization problem by extracting the kernels in every possible order.

The expression of y_same in (21) is in fact a *multiple-valued function* (*mvi-function*), whose input and output variables can take two or more values. y_same is an mvi-function that has n multiple-valued inputs and one binary-valued output. Moreover, each input has n_i-valued logic representations. Thus, we refer the interested reader to [13] for a discussion of multiple-valued minimization; Ref. [13] presents an extension of the original complementation algorithm of the program ESPRESSO [2] for binary-valued functions.

3.4. *Consequence weights with the same value*

In the NFNN system, consequence weights with the same value will be iteratively found for rule combination. We have found in practical simulation results that rules seldom have exactly the same consequence weights without any approximation. In this subsection we propose a method for coping with this problem.

Consider all the existing consequence weights to be sorted in increasing order so that they form a sequence $beta_{iter}$, where *iter* is the number of iterations. Thus, we obtain

$$beta_{iter} = \langle b_1, b_2, \ldots, b_{g(iter)} \rangle, \tag{27}$$

where $g(iter)$ is a function of *iter*. It is clear that $g(1) = m$ for the initial m consequence weights. Furthermore, we denote by $B_{iter,i}$ the set of successive $r(iter)$ elements in $beta_{iter}$, where $i = 1, 2, \ldots, g(iter) - r(iter) + 1$ and $r(iter)$ is a function of *iter* that is set by the user at each iteration. That is,

$$B_{iter,i} = \{b_i, b_{i+1}, b_{i+2}, \ldots, b_{i+r(iter)-1}\}, \tag{28}$$

we emphasize that $b_i \leqslant b_{i+1} \leqslant \cdots \leqslant b_{i+r(iter)-1}$ is still satisfied. We define a *difference measure* (*DM*) for set B to represent the degree of difference between these elements in B:

$$DM_{iter,i} = DM(B_{iter,i}) = \frac{|b_{i+r(iter)-1} - b_i|}{|median(B_{iter,i})|}, \tag{29}$$

where *median* is a function that finds the median number of a set (if there are two medians in a set, take the average of the both medians as the output median). This means that the smaller the value of $DM_{iter,i}$ is, the larger the degree of nearness for the $r(iter)$ values in $B_{iter,i}$. For example, the set $\{1,3,5\}$ will have a larger DM value then the set $\{2,3,4\}$, but the DM value of the set $\{0.1, 0.2, 0.3\}$ is equal to that of the set $\{1,2,3\}$. Furthermore, the *median* function can be replaced by the *average* function, although the latter is more complicated. The user also has to set a parameter e_b. If all the values of $DM_{iter,i}$ are greater than e_b, there will be no elements in $beta_{iter}$ that are approximately equal. Thus at this iteration the NFNN system stops.

If $DM_{iter,I} = \min_{\forall i} DM_{iter,i}$ (of course, $DM_{iter,I} < e_b$ is satisfied), the consequence weights treated as the same in the NFNN system will be determined formally. Let V be the set of all these consequence weights of the form

$$V_{iter} = \{v \mid v \in beta_{iter} \text{ and } |v - median(B_{iter,I})| \leqslant |median(B_{iter,I})| * T\,B\%\}, \tag{30}$$

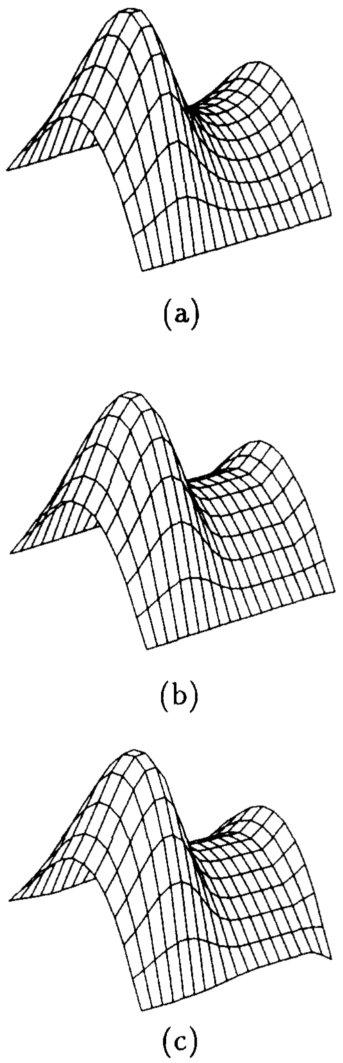

Fig. 3. The performance surfaces of f: (a) desired; (b) after rule combination; and (c) after rule elimination.

where $TB\%$ is the tolerance bound set by the user. Since at least $\min_{i=1,\dots,n} n_i$ consequence weights must be the same for possible rule combination, the number of elements in V_{iter} must be greater than $\min_{i=1,\dots,n} n_i$.

Once the set V_{iter} is obtained, we can construct the expression *y_same* and proceed to do rule combination. When several rules are combined into an equivalent one, the combined consequence weights will be deleted from $beta_{iter}$. A new equivalent consequence weight, the average of these combined consequence weights, will be added in $beta_{iter}$ to replace the respective combined consequence weights. Therefore, we construct a new $beta_{iter}$ for the next iteration. On the contrary, if no rules are combined that $beta_{iter}$, $DM_{iter,I}$, and V_{iter} will be unchanged in the next iteration. In such cases, the $DM_{iter,I}$ in the next iteration is redefined as the least $DM_{iter,i}$ that is not yet chosen in the former iterations. In fact, the simulation results in the next section indicate that this method if highly efficient.

4. Numerical example

The following example of a continuous function is presented to illustrate the proposed procedure for rule combination:

$$f(x_1,x_2)=\frac{\sin(\pi x_2)}{2+\sin(\pi x_1)} \quad \text{for} \quad -1 \leqslant x_1 \leqslant 1 \text{ and } 0 \leqslant x_2 \leqslant 1.$$

The initial structure of the NFNN uses seven term nodes for x_1 and five for x_2, i.e., in this case we have 7×5 initial rules. Since the optimal choice of the number of term nodes is still a difficult problem, we tried several cases and found that the case of 7×5 initial rules is acceptable. Suppose one epoch of learning takes 247 time points. The supervised learning is continued for 300 epochs of training and the sum of squared error is computed for each epoch of learning as

$$SSE=\sum_{k=1}^{247}(y(k)-\hat{y}(k))^2.$$

The desired input–output relation of f is shown in Fig. 3(a). The fuzzy sets for these linguistic term nodes are normally and uniformly initialized. We choose $\eta = 0.01$ and $\alpha = 0.9$ for supervised learning. The parameters

Table 1
Initial and final parameters of the membership functions

Term sets	Initial parameters		Final parameters	
	Mean	Variance	Mean	Variance
A_{11}	−1.000	0.127	−0.695	0.584
A_{12}	−0.667	0.127	−0.493	0.310
A_{13}	−0.333	0.127	0.013	0.804
A_{14}	−0.000	0.127	0.064	0.004
A_{15}	0.333	0.127	0.378	0.003
A_{16}	0.667	0.127	1.018	0.098
A_{17}	1.000	0.127	1.133	0.017
A_{21}	0.000	0.095	0.021	0.199
A_{22}	0.250	0.095	0.193	0.174
A_{23}	0.500	0.095	0.566	0.219
A_{24}	0.750	0.095	0.819	0.159
A_{25}	1.000	0.095	1.014	0.162

Table 2
The final rules after supervised learning

	Preconditions	Consequence
1	A_{11}, A_{21}	−0.119
2	A_{12}, A_{21}	−0.534
3	A_{13}, A_{21}	−0.090
4	A_{14}, A_{21}	0.710
5	A_{15}, A_{21}	0.481
6	A_{16}, A_{21}	−0.151
7	A_{17}, A_{21}	0.812
8	A_{11}, A_{22}	0.379
9	A_{12}, A_{22}	1.806
10	A_{13}, A_{22}	0.356
11	A_{14}, A_{22}	0.704
12	A_{15}, A_{22}	0.609
13	A_{16}, A_{22}	0.506
14	A_{17}, A_{22}	0.868
15	A_{11}, A_{23}	0.385
16	A_{12}, A_{23}	1.924
17	A_{13}, A_{23}	0.396
18	A_{14}, A_{23}	0.755
19	A_{15}, A_{23}	0.548
20	A_{16}, A_{23}	0.531
21	A_{17}, A_{23}	0.893
22	A_{11}, A_{24}	0.236
23	A_{12}, A_{24}	1.139
24	A_{13}, A_{24}	0.221
25	A_{14}, A_{24}	0.881
26	A_{15}, A_{24}	0.584
27	A_{16}, A_{24}	0.317
28	A_{17}, A_{24}	0.928
29	A_{11}, A_{25}	−0.077
30	A_{12}, A_{25}	−0.350
31	A_{13}, A_{25}	−0.064
32	A_{14}, A_{25}	0.890
33	A_{15}, A_{25}	0.610
34	A_{16}, A_{25}	−0.090
35	A_{17}, A_{25}	0.927

Table 3
The final rules after rule combination

	Preconditions	Consequence
1	A_{11}, A_{21}	−0.119
2	A_{12}, A_{21}	−0.534
3	A_{13}, A_{21}	−0.090
4	A_{14}	0.788
5	A_{15}	0.566
6	A_{16}, A_{21}	−0.151
7	A_{17}	0.886
8	A_{11}, A_{22}	0.379
9	A_{12}, A_{22}	1.806
10	A_{13}, A_{22}	0.356
11	A_{16}, A_{22}	0.506
12	A_{11}, A_{23}	0.385
13	A_{12}, A_{23}	1.924
14	A_{13}, A_{23}	0.396
15	A_{16}, A_{23}	0.531
16	A_{11}, A_{24}	0.236
17	A_{12}, A_{24}	1.139
18	A_{13}, A_{24}	0.221
19	A_{16}, A_{24}	0.317
20	A_{11}, A_{25}	−0.077
21	A_{12}, A_{25}	−0.350
22	A_{13}, A_{25}	−0.064
23	A_{16}, A_{25}	−0.090

of the initial and final membership functions are illustrated in Table 1. The rules obtained after 300 epochs of learning are listed in Table 2 with performance represented by $SSE = 0.069155$ (mean square error is 0.000280).

To find the rules with the same consequence weights, we set $r(\cdot) = 5$, $e_b = 0.3$ and $TB\% = 20\%$ for each iteration. Then we have rules 7, 14, 21, 28, and 35 in Table 2 to be combined in the first iteration. The resulting equivalent rule is rule 7 in Table 3. Rules 5, 12, 19, 26, and 33 in Table 2 are another set of rules that are combined in the fourth iteration. Finally, rules 4, 11, 18, 25, 32 in Table 2 are combined in the fifth iteration. The final rules after rule combination are listed in Table 3. The number of rules has been reduced from 35 to 23. Fig. 3(b) shows the performance surface of the function f after rule combination. The SSE after rule combination is still equal to 0.069155, thus attesting to the feasibility of the proposed system. We also list the number of rules after rule combination under different tolerance bounds in Table 4.

Table 4
The number of rules after rule combination under different tolerance bounds ($r(\cdot) = 5$ and $e_b = 0.3$)

Tolerance bound	5%	10%	15%	20%	30%
Number of rules	35	31	27	23	23
Sum of squared errors	0.069155				

Table 5
The final rules after rule elimination

	Preconditions	Consequence
1	A_{11}, A_{21}	−0.119
2	A_{12}, A_{21}	−0.534
3	A_{14}	0.788
4	A_{15}	0.566
5	A_{16}, A_{21}	−0.151
6	A_{17}	0.886
7	A_{11}, A_{22}	0.379
8	A_{12}, A_{22}	1.806
9	A_{13}, A_{22}	0.356
10	A_{16}, A_{22}	0.506
11	A_{11}, A_{23}	0.385
12	A_{12}, A_{23}	1.924
13	A_{13}, A_{23}	0.396
14	A_{16}, A_{23}	0.531
15	A_{11}, A_{24}	0.236
16	A_{12}, A_{24}	1.139
17	A_{13}, A_{24}	0.221
18	A_{16}, A_{24}	0.317
19	A_{12}, A_{25}	−0.350

We can also use the rule elimination method to eliminate these rules which are less important, i.e., to eliminate rules with small consequence weights compared with other rules. Hence rules 3, 20, 22, and 23 in Table 3 can be eliminated. Table 5 lists the final 19 rules, and the corresponding performance surface with $SSE = 0.238726$ (mean square error is 0.000967) is shown in Fig. 3(c).

5. Conclusion

In this paper we have explored a procedure for rule combination in an NFNN system. The structure of the proposed NFNN makes the rule combination method efficient and effective. A sufficient condition for rule combination in the NFNN system is derived and an algorithm for performing rule combination is provided. The sufficient condition and the algorithm can be extended to MIMO systems with a slight modification. Simulation results show that when combined with a rule elimination method the rule combination method can greatly reduce the number of rules.

References

[1] R.K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli and A.R. Wang, MIS: a multiple-level logic optimization system, *IEEE Trans. CAD* **6**(6) (1987) 1062–1081.

[2] R. Brayton, G. Hachtel, C. McMullen and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis* (Kluwer Academic Publishers, Boston, MA, 1984).

[3] J.J. Buckley and Y. Hayashi, Can fuzzy neural nets approximate continuous fuzzy functions?, *Fuzzy Sets and Systems* **61** (1994) 43–51.

[4] G. De Micheli, *Synthesis and Optimization of Digital Circuits* (McGraw-Hill, New York, 1994).

[5] B.W. Grant and A.V. Wal, The use of neural networks for automation of fuzzy knowledge base creation, *1st Asian Fuzzy Systems Symp.*, Singapore (1993) 340–345.

[6] M.M. Gupta and D.H. Rao, On the principles of fuzzy neural networks, *Fuzzy Sets and Systems* **61** (1994) 1–18.

[7] S. Horikawa, T. Furuhashi, S. Okuma and Y. Uchikawa, A fuzzy controller using a neural network and its capability to learn expert's control rules, *Proc. Internat. Conf. on Fuzzy logic and Neural Networks*, July, Iizuka, Japan (1990) 103–106.

[8] S. Horikawa, T. Furuhashi and Y. Uchikawa, On fuzzy modelling using fuzzy neural networks with the back-propagation algorithm, *IEEE Trans. Neural Networks* **3**(5) (1992) 801–806.

[9] J.S. Jang, ANFIS: adaptive-network-based fuzzy inference system, *IEEE Trans. Systems Man Cybernet.* **23**(3) (1993) 665–684.

[10] C.C. Jou, On the mapping capability of fuzzy inference systems, *Proc. Internat. Joint. Conf. on Neural Networks*, Baltimore, ML (1992) 708–713.

[11] T. Kohonon, The self-organizing map, *Proc. IEEE* **78**(9) (1990) 1461–1480.

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

[13] R. Rudell and A. Sangiovanni-Vincentelli, Multiple-valued minimization for PLA optimization, *IEEE Trans. CAD/ICAS* **6**(5) (1987) 727–750.

[14] D.E. Rumelhart and J.L. McClelland (Eds.), *Parallel Distributed Processing, Vol. 1* (MIT Press, Cambridge, MA, 1986).

[15] M. Sugeno and T. Yasukawa, A fuzzy-logical-based approach to qualitative modeling, *IEEE Trans. on Fuzzy Systems* **1**(1) (1993) 7–31.

[16] L.X. Wang, *Adaptive fuzzy systems and control* (Prentice-Hall, Englewood Cliffs, NJ, 1994).

[17] C.W. Xu and Y.Z. Lu, Fuzzy modeling identification and self-learning for dynamical systems, *IEEE Trans. Systems Man Cybernet.* **17**(4) (1987) 683–689.