ELSEVIER

Fuzzy Sets and Systems 91 (1997) 15–35

FUZZY sets and systems

A multiclass neural network classifier with fuzzy teaching inputs

Kuo-Hsiu Chen, Hong-Ling Chen, Hahn-Ming Lee*

Department of Electronic Engineering, National Taiwan Institute of Technology, Taipei, Taiwan

Received August 1995; revised May 1996

Abstract

A multiclass neural network classifier with fuzzy teaching inputs is proposed. The classifier creates each class by aggregating a fuzzy prototype and several fuzzy exemplars in the hidden layer. Fuzzy inputs and all the nodes in the hidden layer are represented by trapezoidal fuzzy numbers. The classifier is trained by a two-pass learning algorithm. In pass one, a very fast one-epoch algorithm PECFUH (Prototype Expansion and Contraction of FUzzy Hyperbox) or FUNLVQ (FUzzy Number's Learning Vector Quantization) is used to train the prototypes. These prototypes will classify as many fuzzy input instances as possible. Afterward, exemplars that mean the exceptions, like the "holes", in pattern space will be generated and expanded in pass two to classify those fuzzy input instances that cannot be correctly classified by the prototypes. A few-epoch FENCE (Fuzzy Exemplar Nested Creation and Expansion) training algorithm is proposed to create the exemplar nodes. Due to the training in pass one, the number of exemplar nodes is reduced and the learning speed is very fast during pass two. In addition, on-line adaptation is supplied in this model and the computational load is lightened. Also, nonlinearly separable instances and overlapping classes can be handled well. Furthermore, this classifier has good generalization ability for the training instances with don't-care information. The experimental results manifest that the training and recalling are fast. At the same time, they illustrate that required nodes are few. © 1997 Elsevier Science B.V.

Keywords: Multiclass classification; Fuzzy neural network; Fuzzy teaching inputs; Prototypes; Exemplars

1. Introduction

Pattern classifiers have wide applicability and play the significant role in many areas, such as the recognition of characters, speech [6,22,28], automatic control [14,27,36], rule-based reasoning [1], and image processing [10], etc. For many applications their input attributes should be represented as fuzzy set [3,15,16] in order to model and process uncertain and ambiguous data. In addition, neural networks adopt numerical computation with fault-tolerance, massively parallel computing and trainable property [17,23,30], so they are suitable for classification algorithms. Thus, we propose a neural network classifier to handle fuzzy inputs.

Recently, the researches on pattern classifiers that integrate neural networks and fuzzy set theories [9,13] have accelerated. Most of these

* E-mail: hmlee@et.ntit.edu.tw.

PII S0165-0114(96)00145-5

researches incorporate the learning ability of neural network classifiers into fuzzy systems [5, 29]. Some researches utilize fuzzy set theory to improve the learning ability of neural network classifiers [4, 18]. However, there are many applications in real world whose input attributes are linguistic or cannot be measured directly as real vectors [8, 11]. They may have ambiguous status and do not have precisely defined criteria of membership [38]. Accordingly, how to build a multiclass neural network classifier with fuzzy inputs handling capability is an important subject.

In our earlier paper [24], a three-layer neural network architecture for classification of fuzzy inputs was introduced. This paper extends the previously proposed model to handle multiclass instances. In addition, we improve the learning speed and the generalization ability to handle don't-care information [7].

The proposed classifier creates classes by aggregating a fuzzy prototype [2, 6] and several fuzzy exemplars [37] into a single class. The prototypes will classify as many fuzzy input instances as possible. Afterward, exemplars that mean the exceptions, like the "holes", in pattern space will be generated and expanded to classify those fuzzy input instances that cannot be correctly classified by the prototypes. Each input is represented as an *LR*-type fuzzy set [39]. Also the prototypes and exemplars are represented in the same way. This model aims to avoid some drawbacks of heavy computational load and high memory requirement. Nonlinearly separable instances and overlapping classes can also be classified by this model. The number of exemplars generated is reduced. Also, on-line adaptation is supplied. The experimental results manifest that both the learning and recalling are fast. Furthermore, the generalization ability to handle don't-care information is very good.

We first describe the input representation in Section 2. The classifier's structure is detailed in Section 3. Section 4 shows the learning procedure. Four experiments and their results are listed in Section 5. The characteristics discussion and the comparison with other models are described in Section 6 and Section 7, respectively. Section 8 gives a conclusion.

2. Input representation

Each feature of all the input data, the network's prototypes and exemplars are represented as a trapezoidal fuzzy number [39] in an *LR*-type fuzzy interval [12]. This is because the *LR*-representation of fuzzy sets can increase computational efficiency without limiting the generality beyond acceptable limits, and the fuzzy interval can represent various types of information [39]. In Fig. 1, the membership function for a trapezoidal fuzzy number $\tilde{W}$ is

$$\mu_{\tilde{W}}(x) = \begin{cases} L\left(\dfrac{w1 - x}{a}\right) & \text{for } x \leqslant w1, \\ 1 & \text{for } w1 \leqslant x \leqslant w2, \\ R\left(\dfrac{x - w2}{b}\right) & \text{for } x \geqslant w2, \end{cases} \qquad (1)$$

where $L(x) = R(x) = \max(0, 1 - x)$ [39]. In what follows, we use an upper case letter with a symbol "$\sim$" to indicate a fuzzy number and lower case letters to denote crisp values.

In this classifier, there are three operations used and the computation load of those operations is light. Let $\tilde{W} = (w1, w2, a, b)_{LR}$, $\tilde{V} = (v1, v2, c, d)_{LR}$, and $z = (z, z, 0, 0)_{LR}$, then the basic operations on trapezoidal fuzzy numbers are listed below:

(1) Addition

$$\tilde{W} + \tilde{V} = (w1 + v1, w2 + v2, a + c, b + d)_{LR}$$

(2) Subtraction

$$\tilde{W} - \tilde{V} = (w1 - v2, w2 - v1, a + d, b + c)_{LR}$$

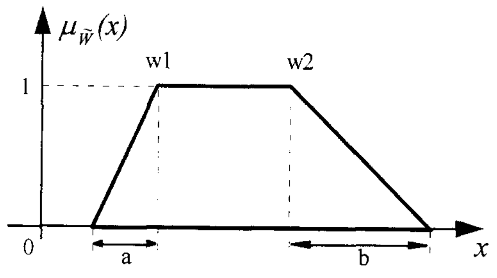


Fig. 1. The membership degree of a trapezoidal fuzzy number.

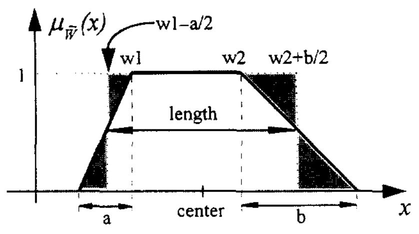


Fig. 2. The center and length of a trapezoidal fuzzy number $\tilde{W} = (w1, w2, a, b)_{LR}$. The sizes of the two gray regions in the left side are equal. Also the right side's two gray areas are equal.

(3) Multiplication with real number

If $z > 0$, $z * \tilde{W} = (z * w1, z * w2, z * a, z * b)_{LR}$

Otherwise

$$z * \tilde{W} = (z * w2, z * w1, -z * b, -z * a)_{LR}.$$

We use the Center Of Area (COA in short [21]) of a fuzzy number as the defuzzification method. To simplify the computation, the defuzzification just takes the center of the rectangle formed from $(w1 - a/2)$ to $(w2 + b/2)$ in the x-axis as shown in Fig. 2. In addition, to measure the size of a fuzzy input pattern or a fuzzy node, we define the length of a fuzzy number. The center and the length are defined as below and are shown in Fig. 2.

(1) center

$$\begin{aligned}\mathrm{COA}(\tilde{W}) &= \int_{w1-a}^{w2+b} x \cdot \mu_{\tilde{W}}(x)\,\mathrm{d}x \Big/ \int_{w1-a}^{w2+b} \mu_{\tilde{W}}(x)\,\mathrm{d}x \\ &\approx (w2 + b/2 + w1 - a/2)/2 \\ &= (w2 + w1)/2 + (b - a)/4. \qquad (2)\end{aligned}$$

(2) length

$$\begin{aligned}\mathrm{LEN}(\tilde{W}) &= (w2 + b/2) - (w1 - a/2) \\ &= w2 - w1 + (b + a)/2. \qquad (3)\end{aligned}$$

In the remainder of this paper, the pattern space will be re-scaled to the unit-hypercube I^n to simplify the equations. All the input data will be normalized into $[0, 1]$.

3. The classifier structure

This classifier is a three-layer neural network. Fig. 3 illustrates the complete structure of this classifier. This model consists of L input nodes, M exemplar nodes, N prototype nodes, and N output nodes. Where the L, M, and N are positive integers.

The input layer contains one node for each input feature. Each input feature is represented as a trapezoidal fuzzy number in LR-type fuzzy set [39]. These nodes are used only to hold input values and to distribute these values to all nodes in the hidden layer.

The hidden layer consists of two sets of nodes: prototype nodes and exemplar nodes. Each of these nodes represents a fuzzy vector. The prototype node indicates the prototype of each class and will classify the most of the input instances [25]. The exemplar nodes represent the exception in pattern space and are generated for classifying those training instances that cannot be correctly classified by the prototypes [31]. Therefore, the number of exemplar nodes generated depends on the complication of classes.

The output layer contains a node for each class. The weight vectors $\boldsymbol{g}_k$ and $\boldsymbol{h}_k$ are adjusted such that when an input vector is presented to the classifier, this classifier's behavior will be like to check whether the input vector is contained in some

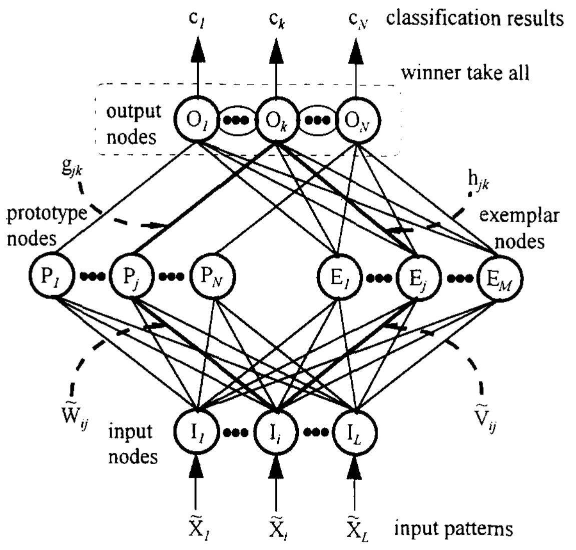


Fig. 3. The structure of proposed neural network model that contains L input nodes, N prototype nodes, M exemplar nodes (i.e. $N + M$ hidden nodes) and N output nodes.

exemplars. If so, the winner exemplar (i.e. the closest and the innermost exemplar) will send a correct classification output to the output node. Otherwise, it will be classified by the prototype of each class.

3.1. The prototype nodes

In the hidden layer, each class is represented by a prototype node, i.e. the number N of prototype nodes is equal to the number of output nodes. The prototype node shown in Fig. 4 acts as a nearest prototype classifier [2,6]. Each node draws a simple decision hypersurface in the pattern space. These decision surfaces provide an efficient way to classify most of input patterns.

Let input fuzzy pattern $\tilde{\boldsymbol{X}} = (\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_L)$, where $\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_L$ are trapezoidal fuzzy numbers. The connection weight $\tilde{W}_{ij}$ between input $\tilde{X}_i$ and prototype node P_j is also a fuzzy number. We use the similarity degree to indicate which prototype node will be the winner. The similarity degree s_j between input $\tilde{\boldsymbol{X}}$ and the jth prototype's weights $\tilde{\boldsymbol{W}}_j$ is measured by the equation below:

$$s_j = 1 - \sqrt{\frac{1}{L}\textstyle\sum_{i=1}^{L} (\mathrm{COA}(\tilde{X}_i) - \mathrm{COA}(\tilde{W}_{ij}))^2} \quad (4)$$

where $\mathrm{COA}(\tilde{M})$, Center Of Area [21] in short, is the center of the trapezoidal fuzzy number $\tilde{M}$, defined in Section 2. It should be mentioned that the pattern space is normalized into $[0, 1]$.

This similarity degree takes value in $[0, 1]$ and it is the output of the jth prototype node. The shorter the distance between $\mathrm{COA}(\tilde{X}_i)$ and $\mathrm{COA}(\tilde{W}_{ij})$ is, the higher the similarity degree s_j will be.

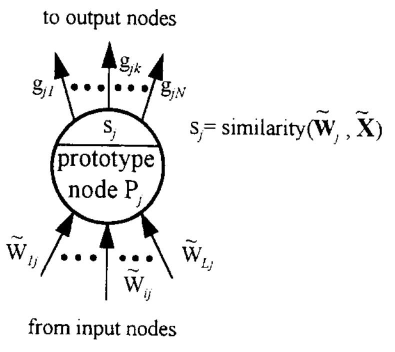

to output nodes
g_{j1}
g_{jk}
g_{jN}
s_j
prototype node P_j
s_j= similarity($\tilde{\mathrm{W}}_j$, $\tilde{\mathrm{X}}$)
$\tilde{\mathrm{W}}_{1j}$
$\tilde{\mathrm{W}}_{ij}$
$\tilde{\mathrm{W}}_{Lj}$
from input nodes

Fig. 4. The prototype node P_j, where $j = 1, 2, \ldots, N$. The crisp value s_j is the similarity degree between input $\tilde{\boldsymbol{X}}$ and prototype's weights $\tilde{\boldsymbol{W}}_j$. Note that only one g_{jk} is value 1 for any P_j, the others are value 0.

3.2. The exemplar nodes

The other node type in the hidden layer is exemplar node illustrated in Fig. 5. The generation of exemplar nodes will be detailed in Section 4.2.

Each exemplar node builds a decision boundary by creating a fuzzy subset of the L-dimensional pattern space. Exemplars used here are nested fuzzy hyperboxes [31]. Each exemplar node represents a fuzzy hyperbox. Nested fuzzy hyperboxes are used to classify nonlinearly separable instances that cannot be separated by hypersurfaces. Also they can handle the exceptions well.

Let input fuzzy vector $\tilde{\boldsymbol{X}} = (\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_L)$, where the connection weight $\tilde{V}_{ij}$ between input $\tilde{X}_i$ and exemplar node E_j is a trapezoidal fuzzy number. Let $\tilde{V}_{ij} = (v1_{ij}, v2_{ij}, a_{ij}, b_{ij})_{LR}$ and $\tilde{X}_i = (y1_i, y2_i, c_i, d_i)_{LR}$. Note that the pattern space is normalized into $[0, 1]$ as mentioned above. To speed up the computation, the subset degree r_j between input $\tilde{\boldsymbol{X}}$ and the jth exemplar's weights $\tilde{\boldsymbol{V}}_j$ is defined as

$$r_j = \begin{cases} 2 - \sqrt{\frac{1}{L}\textstyle\sum_{i=1}^{L} (y1_i - v1_{ij})^2 + (v2_{ij} - y2_i)^2} \\ \quad \text{if } v1_{ij} \leqslant y1_i \text{ and } y2_i \leqslant v2_{ij},\ \forall \tilde{X}_i \in \tilde{X}, \\ 0 \quad \text{otherwise.} \end{cases} \quad (5)$$

According to this subset degree, only the fuzzy hyperboxes that contain the input pattern $\tilde{\boldsymbol{X}}$ will be active. Since the pattern space is normalized into $[0, 1]$, the subset degree r_j will take value in $[1, 2]$, higher than the output value (between zero and one) of any prototype. In addition, the innermost and closest fuzzy hyperbox will take the highest value of subset degree. In this way, exception can be handled well.

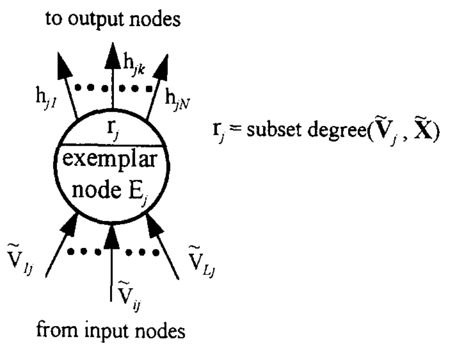


Fig. 5. The exemplar node E_j, where $j = 1, 2, \ldots, M$. The crisp value r_j is the subset degree between input $\tilde{X}$ and exemplar's weights $\tilde{V}_j$. Note that only one h_{jk} is value 1 for any E_j, the others are value 0.

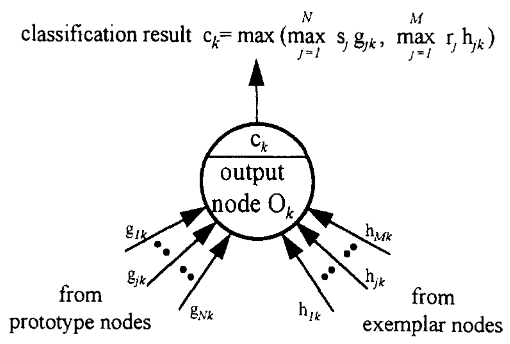


Fig. 6. The output node O_k, where $k = 1, 2, \ldots, N$. For each output node O_k, there is only one g_{jk} whose value is 1, but has several h_{jk} whose values are 1. The crisp value c_k is the classification result or teaching input during learning process. The s_j is the similarity degree between input $\tilde{X}$ and the jth prototype's weights $\tilde{W}_j$. The r_j is the subset degree between the input $\tilde{X}$ and the jth exemplar's weights $\tilde{V}_j$.

3.3. *The output nodes*

The output layer consists of one node for each pattern class. The output node shown in the Fig. 6 receives the information from both the prototype nodes and the exemplar nodes. The weights from prototypes and exemplars are set such that the following behavior of it is achieved. That is, if input $\tilde{X}$ is contained in some fuzzy hyperboxes, classification result is equal to the class of the winner exemplar node no matter what output values of the prototype nodes are. Otherwise, classification result is decided by the prototype nodes.

The connection weight g_{jk} between prototype node P_j and output node O_k is either 1 or 0. The equation for assigning the value to the connection is given below:

$$g_{jk} = \begin{cases} 1 & \text{if prototype node } P_j \text{ is class } k, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

The connection weight h_{jk} between exemplar node E_j and output node O_k is also a binary value and fixed. The value of the connection weight is assigned by the following equation:

$$h_{jk} = \begin{cases} 1 & \text{if exemplar node } E_j \text{ is class } k, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

The output c_k of the output node O_k indicates the degree to which the input vector $\tilde{X}$ fits within the class k. The transfer function for the output nodes is the fuzzy union of the appropriate prototype similarity degree and fuzzy hyperbox subset degree. The output for the class node is defined as

$$c_k = \max\left(\max_{j=1}^{N} s_j g_{jk}, \max_{j=1}^{M} r_j h_{jk} \right) \tag{8}$$

where s_j is the similarity degree between input $\tilde{X}$ and prototype P_j, r_j is the subset degree between input $\tilde{X}$ and exemplar E_j.

Since equation defined above is not a rigorous membership function, the hard decision is used in the classification. The class node with the highest value is located and its node value is set to 1 to indicate that it is the closest pattern class. The remaining class node values are set to 0. Such an operation is usually referred to as a winner-take-all response [20].

4. The learning procedure

The training of this classifier is a two-pass learning procedure shown in Fig. 7. In the first pass, Prototypes Expansion and Contraction of FUzzy Hyperbox (PECFUH) is used to train the prototype nodes. This algorithm is extended from Simpson's fuzzy min-max learning algorithm [33,34]. Since the prototypes represent the major

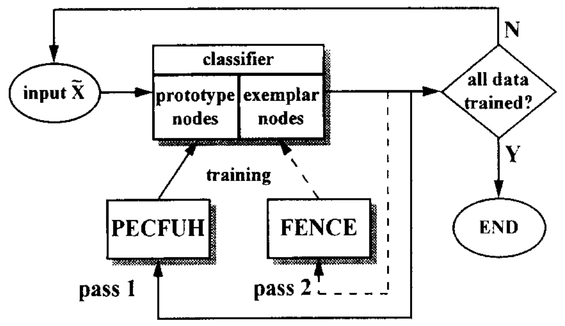


Fig. 7. The learning process of proposed classifier is shown. The solid line represents the training path during pass 1 learning. The training of pass 2 is an error-driven process and its path is represented by dash line.

distribution of corresponding classes, they will classify as many fuzzy input instances as possible.

In the second pass, an on-line adaptation method, Fuzzy Exemplars Nested Creation and Expansion (FENCE) is used to generate the exemplar nodes. The FENCE is extended from our previous work FVGE (Fuzzy Vectors Generation/Expansion) [24]. The exemplar represents an exception status, like a "hole", in the pattern space. The exemplars will be generated and expanded to classify those fuzzy input instances that cannot be correctly classified by the prototypes.

Note that the algorithm used in pass 1 is one-epoch training algorithm. The pass 2 training takes few epochs which two epochs is enough in our experiments, and is executed when a pattern was misclassified by classifier.

In the following sections, we describe the PECFUH and FENCE in detail. Furthermore we illustrate them with a simple two-dimensional example. An alternative training method, FUNLVQ (FUzzy Number's Learning Vector Quantization) for pass 1 is proposed in Section 4.3.

4.1. The PECFUH algorithm

There are four steps in this algorithm. They are creation, expansion, overlap test, and contraction. Before training, the weights from the input nodes to hidden nodes and from the hidden nodes to output nodes are set to zero. After the initialization, when an input pattern is entered, the prototype with the same class will be created (if the prototype not established yet) or expanded to include the input pattern. If any overlap occurs among the weights of prototypes, the overlap will be eliminated using a contraction process. Note that only the parts that have membership value one of the prototype are contracted. The overlaps between the spread regions of each prototype are still remained. The flowchart of PECFUH is shown in Fig. 8 and a simple example illustrates the above-mentioned processes is shown in Figs. 12–14.

Now, we detail the four operations of the PECFUH. Before training, all the weights in this model are initialized to zero.

(a) *Creation*: When an input pattern for a new class enters, the weight vector $\tilde{W}_j = (\tilde{W}_{1j}, \ldots, \tilde{W}_{ij}, \ldots, \tilde{W}_{Lj})$ from input nodes to the jth prototype node P_j that belongs to the class is set as the current pattern. Let $\tilde{W}_{ij} = (w1_{ij}, w2_{ij}, a_{ij}, b_{ij})_{LR}$ and input pattern $\tilde{X}_i = (y1_i, y2_i, c_i, d_i)_{LR}$ for each the ith feature, then

$$w1_{ij} = y1_i, \qquad w2_{ij} = y2_i,$$

$$a_{ij} = c_i \quad \text{and} \quad b_{ij} = d_i.$$

In addition, the weight g_{jj} connecting the jth prototype node and the class's output node is set to one.

(b) *Expansion*: When one pattern enters, if the class node and its prototype node for the input pattern are established, that the jth prototype's weight $\tilde{W}_j$ will be expanded to this pattern $\tilde{X}$. Let $\tilde{W}_{ij} = (w1_{ij}, w2_{ij}, a_{ij}, b_{ij})_{LR}$ and $\tilde{X}_i = (y1_i, y2_i, c_i, d_i)_{LR}$ for each the ith feature, then the expanded prototype's weight $\tilde{W}'_{ij} = (w1'_{ij}, w2'_{ij}, a'_{ij}, b'_{ij})_{LR}$ is modified by

$$w1'_{ij} = \mathrm{Min}(w1_{ij}, y1_i), \qquad w2'_{ij} = \mathrm{Max}(w2_{ij}, y2_i),$$

$$\text{If } w1_{ij} \geqslant y1_i, \quad \text{then } a'_{ij} = c_i; \quad \text{otherwise } a'_{ij} = a_{ij},$$

$$\text{If } w2_{ij} \leqslant y2_i, \quad \text{then } b'_{ij} = d_i; \quad \text{otherwise } b'_{ij} = b_{ij}.$$

Fig. 9 illustrates above-mentioned expanding operations in the feature i.

(c) *Overlap test*: When the weights of one prototype $\tilde{W}_{j^1}$ are expanded, there are four possible overlapping cases between the weights and the other prototypes' weights $\tilde{T}_{j^2}$ in each feature. Let $\tilde{W}_{ij^1} = (w1_{ij^1}, w2_{ij^1}, a_{ij^1}, b_{ij^1})_{LR}$ and $\tilde{T}_{ij^2} = (t1_{ij^2}, t2_{ij^2}, c_{ij^2}, d_{ij^2})_{LR}$ for all the ith feature, where $j^1, j^2 = 1, 2, \ldots, N$ and

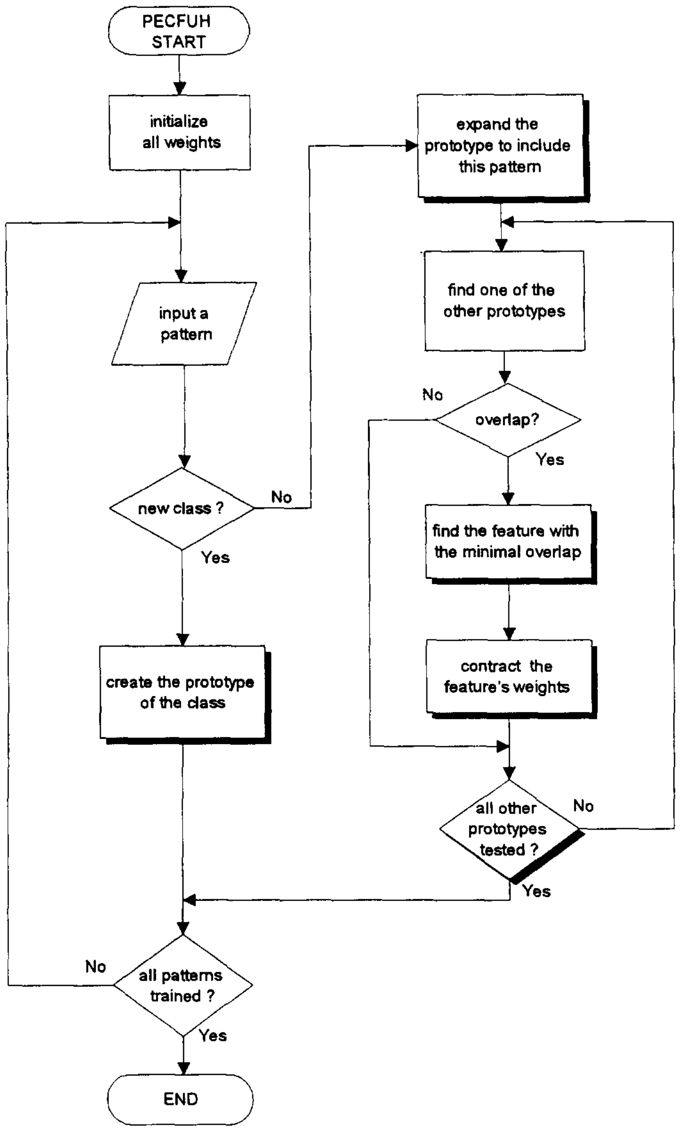

PECFUH START
initialize all weights
input a pattern
new class ?
No
Yes
create the prototype of the class
expand the prototype to include this pattern
find one of the other prototypes
overlap?
No
Yes
find the feature with the minimal overlap
contract the feature's weights
all other prototypes tested ?
No
Yes
all patterns trained ?
No
Yes
END

Fig. 8. The flowchart of PECFUH that is used to train the prototype nodes.

$j^1 \neq j^2$. The four overlapping cases are described below and the subscripts of weights are omitted for simplicity:

Case 1: $w1 \leqslant t1 < w2 \leqslant t2$, then $\delta_i = w2 - t1$,

Case 2: $t1 \leqslant w1 < t2 \leqslant w2$, then $\delta_i = t2 - w1$,

Case 3: $w1 < t1 \leqslant t2 < w2$,
then $\delta_i = \min(w2 - t1, t2 - w1)$,

Case 4: $t1 < w1 \leqslant w2 < t2$
then $\delta_i = \min(w2 - t1, t2 - w1)$,

where δ_i is the length that will be contracted in feature i. If one of the four overlapping cases is checked out in every feature, the overlap between two prototypes exists. If so, the contraction process is applied in one feature. To determine which feature's weights will be contracted in next step, the minimal δ_i, $i = 1, 2, \dots, L$, must be found and then set $\lambda = i$ for later use. The illustration of the overlapping cases is shown in Fig. 10.

(d) *Contraction*: According to the λ found in (c), the two fuzzy numbers $\tilde{W}_{\lambda j^1} = (w1_{\lambda j^1}, w2_{\lambda j^1}, a_{\lambda j^1}, b_{\lambda j^1})_{LR}$ and $\tilde{T}_{\lambda j^2} = (t1_{\lambda j^2}, t2_{\lambda j^2}, c_{\lambda j^2}, d_{\lambda j^2})_{LR}$ will be contracted by one of the following four cases in the λth feature, where $j^1, j^2 = 1, 2, \ldots, N$ and $j^1 \neq j^2$. The weights $\tilde{W}'_{\lambda j^1} = (w1'_{\lambda j^1}, w2'_{\lambda j^1}, a'_{\lambda j^1}, b'_{\lambda j^1})_{LR}$ and $\tilde{T}'_{\lambda j^2} = (t1'_{\lambda j^2}, t2'_{\lambda j^2}, c'_{\lambda j^2}, d'_{\lambda j^2})_{LR}$ are modified from $\tilde{W}_{\lambda j^1}$ and $\tilde{T}_{\lambda j^2}$ after contraction. These four cases are described below and the subscripts of weights are omitted for simplicity:

Case 1: $w1 \leqslant t1 < w2 \leqslant t2$, then
$t1' = w2' = (w2 + t1)/2$,
$b' = b + w2 - w2'$, $c' = c + t1' - t1$

Case 2: $t1 \leqslant w1 < t2 \leqslant w2$, then
$w1' = t2' = (t2 + w1)/2$,
$a' = a + w1' - w1$, $d' = d + t2 - t2'$.

Case 3a: $w1 < t1 \leqslant t2 < w2$ and
$t2 - w1 \leqslant w2 - t1$, then
$w1' = t2$, $a' = a + w1' - w1$.

Case 3b: $w1 < t1 \leqslant t2 < w2$ and
$t2 - w1 > w2 - t1$, then
$w2' = t1$, $b' = b + w2 - w2'$.

Case 4a: $t1 < w1 \leqslant w2 < t2$ and
$t2 - w1 \leqslant w2 - t1$, then
$t2' = w1$, $d' = d + t2 - t2'$.

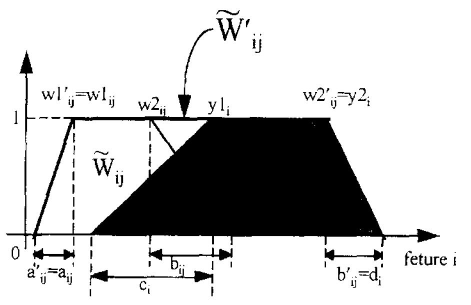


Fig. 9. The expansion from a fuzzy weight $\tilde{W}_{ij} = (w1_{ij}, w2_{ij}, a_{ij}, b_{ij})_{LR}$ to an input pattern's weight $\tilde{X}_i = (y1_i, y2_i, c_i, d_i)_{LR}$ in the ith feature. The two weights are shown by thin line and by shaded region, respectively. The expanded fuzzy number $\tilde{W}'_{ij} = (w1'_{ij}, w2'_{ij}, a'_{ij}, b'_{ij})_{LR}$ is represented by bolder line. That is $w1'_{ij} = w1_{ij}$, $w2'_{ij} = y2_i$, $a'_{ij} = a_{ij}$, $b'_{ij} = d_i$.

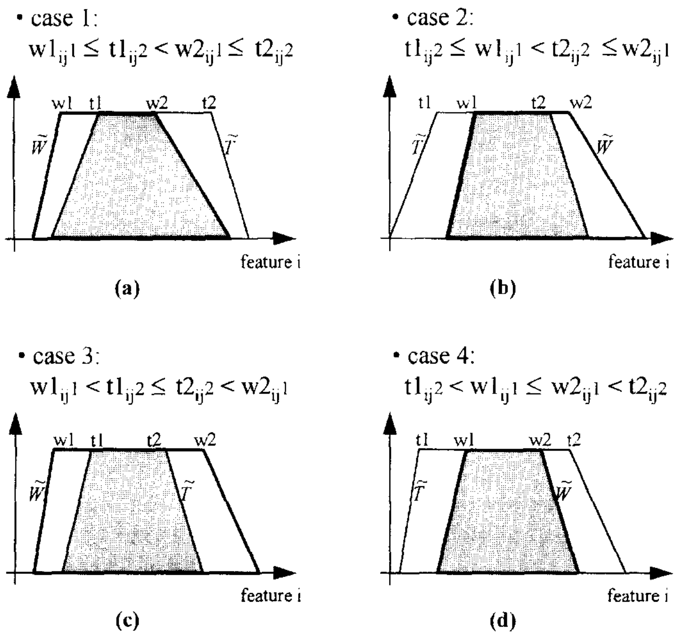


Fig. 10. The four overlapping cases between two fuzzy weights $\tilde{W}_{ij^1}$ and $\tilde{T}_{ij^2}$. Let $\tilde{W}_{ij^1} = (w1_{ij^1}, w2_{ij^1}, a_{ij^1}, b_{ij^1})_{LR}$ and $\tilde{T}_{ij^2} = (t1_{ij^2}, t2_{ij^2}, c_{ij^2}, d_{ij^2})_{LR}$ in ith feature. The subscript of each variable is omitted to simplify the diagram. The bolder line forms a prototype's weight $\tilde{W}_{ij^1}$ in feature i and the thin line forms another prototype's weight $\tilde{T}_{ij^2}$. The overlap between the two weights is shown by shaded region.

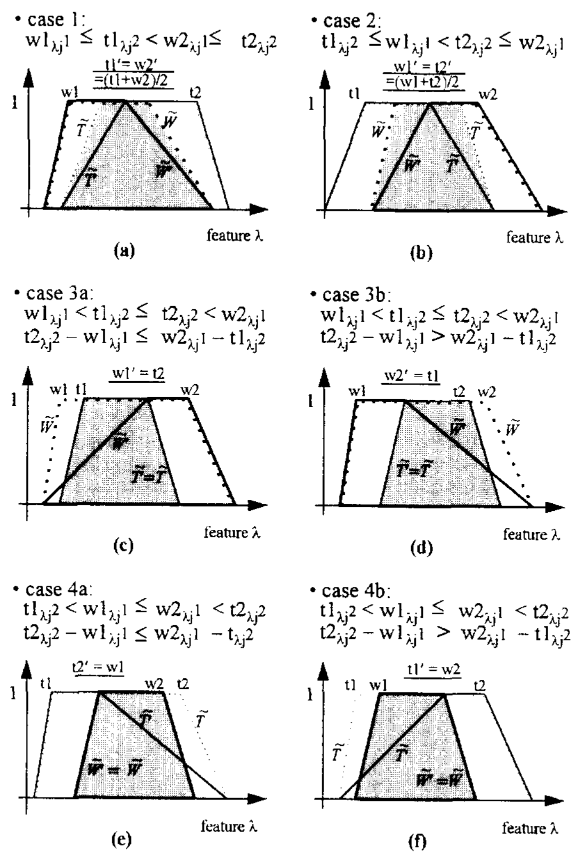
Case 4b: $t1 < w1 \leqslant w2 < t2$ and
$t2 - w1 > w2 - t1$, then
$t1' = w2$, $c' = c + t1' - t1$.

By finding the minimal δ_i and the above-mentioned contraction process, the new instance can be learnt and the old fuzzy information can be maintained. Furthermore, this will make the length of the fuzzy numbers (the size of the prototypes) as wide a range as possible, and make the classifier one with better generalization ability. The illustration of the contraction process is shown in Fig. 11.

A simple two-attribute, four-instance example shown in Fig. 12 is given to illustrate the above-mentioned PECFUH operations. Fig. 13 shows the creation, expansion, and overlap test process when the first three randomly selected patterns entered. Fig. 14 contains the expansion, overlap test, and contraction process when the last pattern in training sequence entered.

4.2. *The FENCE algorithm*

The FENCE learning process begins if any training instance $\tilde{X}$ cannot be correctly classified by prototypes and existent exemplars. If so, an exemplar E that is nearest to $\tilde{X}$ and satisfies following (a)–(c) will be found. Note that, we use the same similarity function of prototype nodes that is described in Section 3.1 to achieve the "nearest" computation.



• case 1: $w1_{\lambda j}1 \le t1_{\lambda j}2 < w2_{\lambda j}1 \le t2_{\lambda j}2$
$\underline{t1' = w2'}$ $\underline{=(t1+w2)/2}$
• case 2: $t1_{\lambda j}2 \le w1_{\lambda j}1 < t2_{\lambda j}2 \le w2_{\lambda j}1$
$\underline{w1' = t2'}$ $\underline{=(w1+t2)/2}$
• case 3a: $w1_{\lambda j}1 < t1_{\lambda j}2 \le t2_{\lambda j}2 < w2_{\lambda j}1$, $t2_{\lambda j}2 - w1_{\lambda j}1 \le w2_{\lambda j}1 - t1_{\lambda j}2$
$\underline{w1' = t2}$
• case 3b: $w1_{\lambda j}1 < t1_{\lambda j}2 \le t2_{\lambda j}2 < w2_{\lambda j}1$, $t2_{\lambda j}2 - w1_{\lambda j}1 > w2_{\lambda j}1 - t1_{\lambda j}2$
$\underline{w2' = t1}$
• case 4a: $t1_{\lambda j}2 < w1_{\lambda j}1 \le w2_{\lambda j}1 < t2_{\lambda j}2$, $t2_{\lambda j}2 - w1_{\lambda j}1 \le w2_{\lambda j}1 - t_{\lambda j}2$
$\underline{t2' = w1}$
• case 4b: $t1_{\lambda j}2 < w1_{\lambda j}1 \le w2_{\lambda j}1 < t2_{\lambda j}2$, $t2_{\lambda j}2 - w1_{\lambda j}1 > w2_{\lambda j}1 - t1_{\lambda j}2$
$\underline{t1' = w2}$
feature λ
(a)
(b)
(c)
(d)
(e)
(f)

Fig. 11. The contraction for the four overlapping cases between the two weights, $\tilde{W}_{\lambda j^1} = (w1_{\lambda j^1}, w2_{\lambda j^1}, a_{\lambda j^1}, b_{\lambda j^1})_{LR}$ and $\tilde{T}_{\lambda j^2} = (t1_{\lambda j^2}, t2_{\lambda j^2}, c_{\lambda j^2}, d_{\lambda j^2})_{LR}$ for the λth feature. The subscript of each variable is omitted to simplify the diagram. Both the two weights are shown in dash-line and their overlap is shown by shaded region. The bolder line forms a contracted prototype's weight $\tilde{W}'_{\lambda j^1} = (w1'_{\lambda j^1}, w2'_{\lambda j^1}, a'_{\lambda j^1}, b'_{\lambda j^1})_{LR}$ and the thin line forms another contracted prototype's weight $\tilde{T}'_{\lambda j^2} = (t1'_{\lambda j^2}, t2'_{\lambda j^2}, c'_{\lambda j^2}, d'_{\lambda j^2})_{LR}$ in the λth feature. The variables shown in underline indicate the modified variables. Considering the members with the membership value one, there is no overlap between the two contracted weights.

• The training example :

Patterns	Class
$\tilde{X}1_1 = (0.1, 0.2, 0.05, 0.10)_{LR}$ $\tilde{X}1_2 = (0.4, 0.6, 0.10, 0.10)_{LR}$	d1=1
$\tilde{X}2_1 = (0.5, 0.7, 0.10, 0.15)_{LR}$ $\tilde{X}2_2 = (0.8, 0.9, 0.05, 0.05)_{LR}$	d2=1
$\tilde{X}3_1 = (0.4, 0.5, 0.05, 0.05)_{LR}$ $\tilde{X}3_2 = (0.5, 0.6, 0.05, 0.05)_{LR}$	d3=2
$\tilde{X}4_1 = (0.6, 0.8, 0.10, 0.10)_{LR}$ $\tilde{X}4_2 = (0.2, 0.3, 0.15, 0.10)_{LR}$	d4=2

• Randomly selected training sequence during PECFUH: $\tilde{X}3$, $\tilde{X}2$, $\tilde{X}4$, $\tilde{X}1$.

Fig. 12. A two-class training example for PECFUH learning algorithm. The subscript means the ith feature of the input pattern. The desired-out of each input pattern is indicated as class 1 or class 2. The training order of input data is randomly selected.

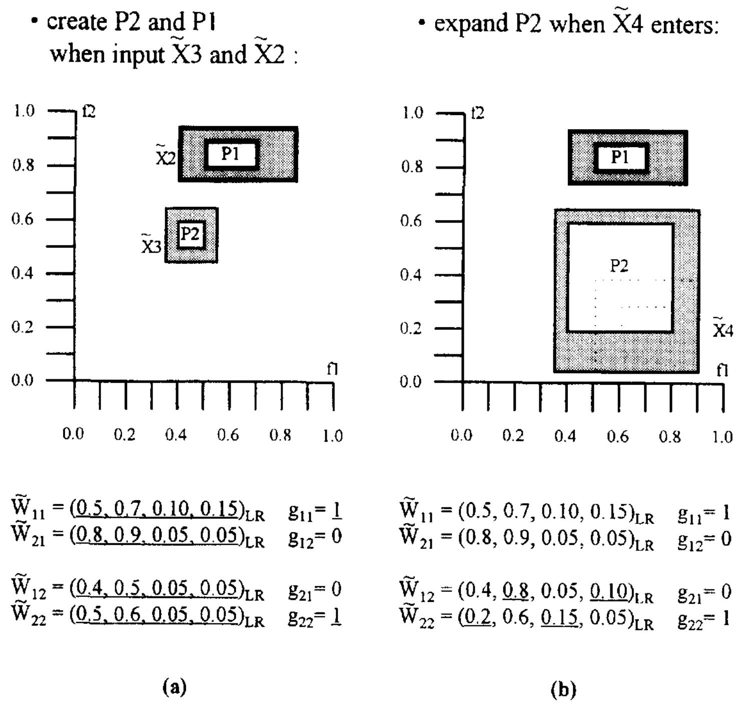

• create P2 and P1 when input X̃3 and X̃2 :
• expand P2 when X̃4 enters:
f1
f2
P1
P2
X̃2
X̃3
X̃4
W̃11 = (0.5, 0.7, 0.10, 0.15)LR g11 = 1
W̃21 = (0.8, 0.9, 0.05, 0.05)LR g12 = 0
W̃12 = (0.4, 0.5, 0.05, 0.05)LR g21 = 0
W̃22 = (0.5, 0.6, 0.05, 0.05)LR g22 = 1
W̃11 = (0.5, 0.7, 0.10, 0.15)LR g11 = 1
W̃21 = (0.8, 0.9, 0.05, 0.05)LR g12 = 0
W̃12 = (0.4, 0.8, 0.05, 0.10)LR g21 = 0
W̃22 = (0.2, 0.6, 0.15, 0.05)LR g22 = 1
(a)
(b)

Fig. 13. The creation and expansion of PECFUH under the training data listed in Fig. 12. In this figure, all the modified variables are shown in underline. The f1 and f2 are the first and the second feature, respectively. The white boxes have the membership value one, and the gray boxes mean the membership degree is between zero and one. When the first two patterns $\tilde{X}3$ and $\tilde{X}2$ were entered, the weights of P2 and P1 are created and shown in panel (a). In panel (b), $\tilde{X}4$ entered. Because it belongs to class 2, the weights of prototype P2 are expanded to include it. The $\tilde{X}4$ is shown by dash line. There is no overlap between expanded P2 and P1.

(a) The class of E is the same as the class of $\tilde{X}$. Clearly, if the class of an exemplar is not same as the pattern's class, the exemplar cannot expand to this pattern.

(b) If E expands to $\tilde{X}$ resulting in E', no overlap occurs between E' and the other exemplars. The expansion of the exemplar is the same as the method used in PECFUH. The overlap test is slightly different from PECFUH since the FENCE algorithm allows the nested exemplars generating. That is, if the two exemplars' overlapping case as shown in Section 4.1 is case 3 for every feature or case 4 for every feature, then the nested status occurs. In addition, the exemplars with the same class are allowed to overlap. In this way, the number of the generated exemplar nodes can be reduced.

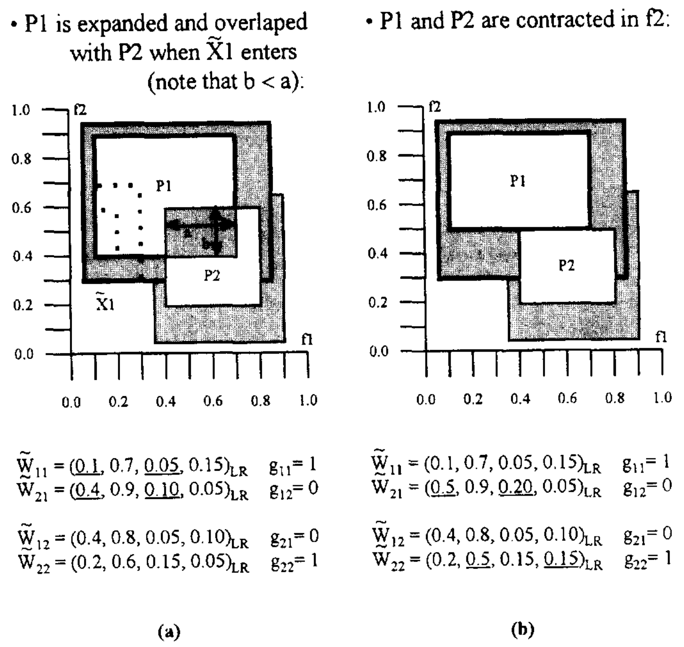
(c) The size of expanded E' cannot exceed the allowed value. The size of an exemplar E_j is computed by

$$\text{size}(E_j) = \frac{1}{L}\sum_{i=1}^{L} \text{LEN}(\tilde{V}_{ij}) \tag{9}$$

where the LEN($\cdot$) is the length of a fuzzy number defined in Section 2 and the L is the number of input features. Furthermore, $\tilde{V}_{ij}$ is the weight from ith input node to jth exemplar node. We then use the size parameter ρ to limit the size of exemplar nodes:

$$\text{size}(E') \leqslant \begin{cases} \rho & \text{if the winner is a prototype,} \\ \rho \times \text{size}(E_{\text{fal}}) & \text{if the winner is exemplar } E_{\text{fal}}. \end{cases}$$

$$\text{size}(E_{\text{sub}}) \leqslant \rho \times \text{size}(E')$$


• P1 is expanded and overlaped with P2 when X̃1 enters (note that b < a):
• P1 and P2 are contracted in f2:
f2
f1
P1
P2
X̃1
1.0
0.8
0.6
0.4
0.2
0.0
0.0 0.2 0.4 0.6 0.8 1.0
(a)
(b)

(a)

$\tilde{W}_{11} = (\underline{0.1}, 0.7, \underline{0.05}, 0.15)_{LR}$ $g_{11} = 1$
$\tilde{W}_{21} = (\underline{0.4}, 0.9, \underline{0.10}, 0.05)_{LR}$ $g_{12} = 0$

$\tilde{W}_{12} = (0.4, 0.8, 0.05, 0.10)_{LR}$ $g_{21} = 0$
$\tilde{W}_{22} = (0.2, 0.6, 0.15, 0.05)_{LR}$ $g_{22} = 1$

(b)

$\tilde{W}_{11} = (0.1, 0.7, 0.05, 0.15)_{LR}$ $g_{11} = 1$
$\tilde{W}_{21} = (\underline{0.5}, 0.9, \underline{0.20}, 0.05)_{LR}$ $g_{12} = 0$

$\tilde{W}_{12} = (0.4, 0.8, 0.05, 0.10)_{LR}$ $g_{21} = 0$
$\tilde{W}_{22} = (0.2, \underline{0.5}, 0.15, \underline{0.15})_{LR}$ $g_{22} = 1$

Fig. 14. Continuing the operations in Fig. 13. In panel (a), when $\tilde{X}1$ formed by dash line enters, the weights of P1 are expanded to include $\tilde{X}1$. Now, the overlap occurs between expanded P1 and P2, and its area is formed by $a \times b$. Because a is greater than b, the second feature's weights of P1 and P2 are contracted. The final weights of this classifier after PECFUH are shown in panel (b).

where the E_{fal} is the winner exemplar, but it is misfired for current pattern. The E_{sub} is the exemplar node that is nested in the expanded E'.

If the classifier can find one match exemplar, E_{mat}, which is nearest to $\tilde{X}$ and satisfies conditions (a)–(c) shown above, then each feature's fuzzy number of the match exemplar will be expanded to include the instance $\tilde{X}$. If no E_{mat} can be found, a new exemplar node will be generated for this instance. The nested exemplars may be generated in the classifier.

We can summarize this on-line learning algorithm FENCE as follows (see Fig. 15 for flowchart): When an input pattern was misclassified:

Step 1: For each exemplar E, if the class of this exemplar is not the same as the class of current pattern $\tilde{X}$, then go to step 4.

Step 2: Expand E to current pattern $\tilde{X}$. Name the expanded exemplar E'. If overlap occurs between E' and any other exemplars, then go to step 4.

Step 3: If the expanded E' does not exceed the allowed size, then put E to the E_{mat} list.

Step 4: If some exemplars are not tested, then go to Step 1.

Step 5: If we can find an E_{mat} that is the nearest exemplar to $\tilde{X}$, then expand the E_{mat} to $\tilde{X}$. Otherwise, generate a new exemplar for $\tilde{X}$.

By setting an appropriate size parameter ρ, two epochs are enough to correctly classify the training set during our experiments. The illustration of the FENCE process under the simple data listed in Fig. 12 is shown in Fig. 16.

According to the weights of this classifier trained by PECFUH after pass one, this classifier will be trained in this second pass by FENCE with two epochs. In first epoch, the patterns' entering order is randomly selected as $\tilde{X}2, \tilde{X}4, \tilde{X}1, \tilde{X}3$. When $\tilde{X}2$ enters, the P1 sends a highest degree to the output layer. Thus, the model correctly classifies $\tilde{X}2$ as class 1. For the coming $\tilde{X}4$ and $\tilde{X}1$, the winner nodes are P2 and P1, respectively. That means the two patterns are also correctly classified by the classifier.

When $\tilde{X}3$ enters, the similarity degree between it and P1 is greater than the degree between it and P2. This is a misclassification. The model classifies $\tilde{X}3$ as class 1, but the desired-out of $\tilde{X}3$ is class 2. Due to none of the exemplar generated, the model

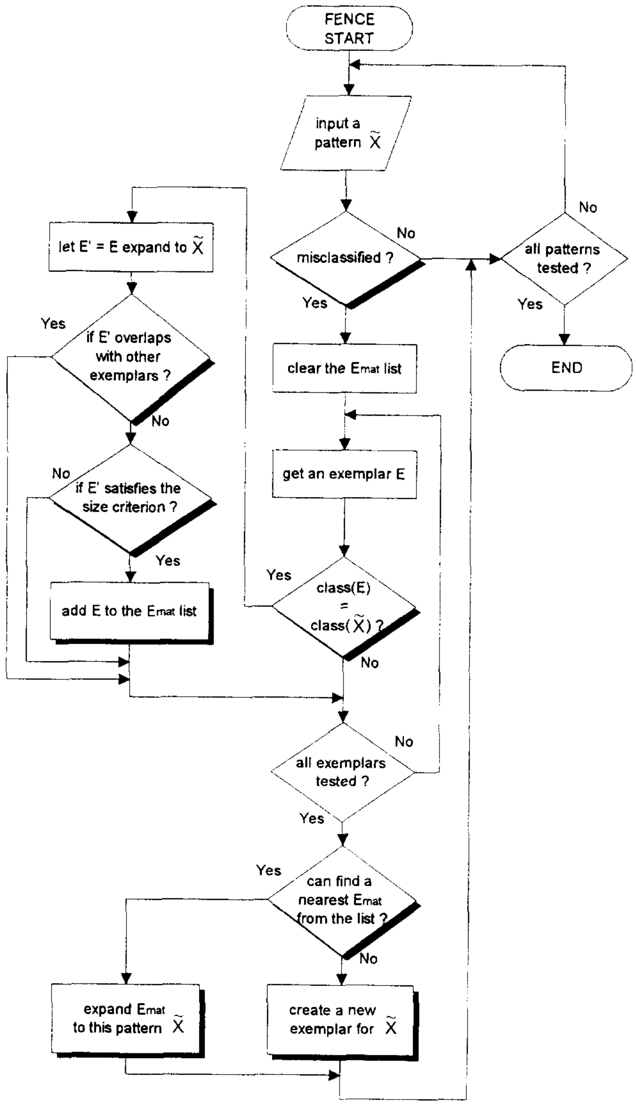

FENCE START
input a pattern $\tilde{X}$
misclassified ?
No
Yes
all patterns tested ?
No
Yes
END
clear the Emat list
get an exemplar E
class(E) = class($\tilde{X}$) ?
Yes
No
let E' = E expand to $\tilde{X}$
if E' overlaps with other exemplars ?
Yes
No
if E' satisfies the size criterion ?
No
Yes
add E to the Emat list
all exemplars tested ?
No
Yes
can find a nearest Emat from the list ?
Yes
No
expand Emat to this pattern $\tilde{X}$
create a new exemplar for $\tilde{X}$

Fig. 15. The flowchart of FENCE algorithm that is used to train the exemplar nodes.

generates a new exemplar node E1 for $\tilde{X}3$. The first epoch ends.

In the second epoch of this second pass, the patterns are entered by another random order $\tilde{X}2, \tilde{X}3, \tilde{X}1, \tilde{X}4$. According to this sequence, the model indicates that the winnes are P1, E1, P1, and P2 respectively. At the same time, the corresponding results are classes 1, 2, 1, and 2. Clearly, there is no misclassification after this epoch.

Now, the classifier is correctly trained for this simple example listed in Fig. 12, and the final weights of all the hidden nodes are shown in panel (b) of Fig. 16. The operations of pass one training PECFUH for this example is shown in Figs. 13 and 14. Fig. 16 shows the second pass training process – FENCE.

The FENCE learning is to generate or expand exemplar nodes to correctly classify those instances

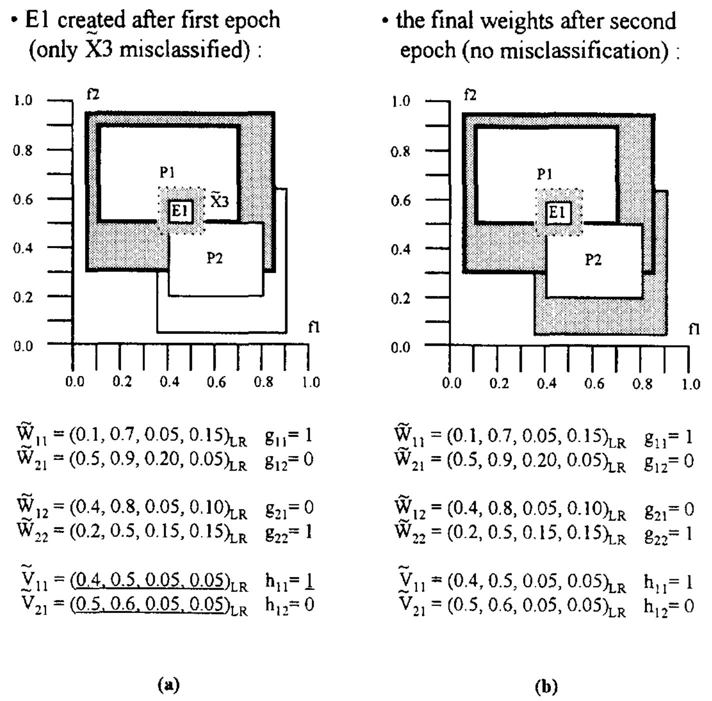

• E1 created after first epoch (only $\tilde{X}3$ misclassified) :
• the final weights after second epoch (no misclassification) :
f2
f1
P1
P2
E1
X3
1.0
0.8
0.6
0.4
0.2
0.0
$\tilde{W}_{11} = (0.1, 0.7, 0.05, 0.15)_{LR}$ $g_{11} = 1$
$\tilde{W}_{21} = (0.5, 0.9, 0.20, 0.05)_{LR}$ $g_{12} = 0$
$\tilde{W}_{12} = (0.4, 0.8, 0.05, 0.10)_{LR}$ $g_{21} = 0$
$\tilde{W}_{22} = (0.2, 0.5, 0.15, 0.15)_{LR}$ $g_{22} = 1$
$\tilde{V}_{11} = \underline{(0.4, 0.5, 0.05, 0.05)}_{LR}$ $h_{11} = \underline{1}$
$\tilde{V}_{21} = \underline{(0.5, 0.6, 0.05, 0.05)}_{LR}$ $h_{12} = 0$
$\tilde{V}_{11} = (0.4, 0.5, 0.05, 0.05)_{LR}$ $h_{11} = 1$
$\tilde{V}_{21} = (0.5, 0.6, 0.05, 0.05)_{LR}$ $h_{12} = 0$
(a)
(b)

Fig. 16. The operations of FENCE algorithm under the training data shown in Figs. 12–14. When the four instances enter the classifier with a randomly selected sequence $\tilde{X}2, \tilde{X}4, \tilde{X}1, \tilde{X}3$, the first three patterns are correctly classified, but $\tilde{X}3$ is misclassified by the classifier. So, an exemplar node E1 is generated for $\tilde{X}3$ and all the E1's weights are set as shown in underline in panel (a). In panel (b), the patterns are entered according to another randomly selected training sequence $\tilde{X}2, \tilde{X}3, \tilde{X}1, \tilde{X}4$. Due to the exception that is caused by $\tilde{X}3$, there is no misclassification in this second epoch.

that cannot be classified by the prototype nodes and existent exemplar nodes. It is important for a classifier to be able to learn new information without destroying the old one (so called on-line learning). Clearly, on-line learning ability is supplied in this model if the FENCE learning is applied during operation.

4.3. *The FUNLVQ algorithm*

In addition to the PECFUH learning algorithm mentioned in Section 4.1, FUzzy Number's Learning Vector Quantization [20] (FUNLVQ) can be used to train the prototype nodes. This algorithm is extended from the learning rule of LVQ [18,20]. The initial weights from input nodes to prototype nodes are set as random values in [0.45, 0.55]. In addition, the weight connecting the ith prototype node and the ith output node is set to one. The remain weights are all initialized to zero. The similarity degree of an input vector and the prototype node of each class defined in Section 3.1 is computed. Then the prototype node that has highest degree is declared to be the winner. If the winner node is the same as the class of the input vector, it is moved toward the input pattern. The fuzzy weight $\tilde{W}_{ij}$ is updated by

$$\tilde{W}'_{ij} = \tilde{W}_{ij} + \alpha(\tilde{X}_i - \tilde{W}_{ij}). \tag{10}$$

where α is the learning rate initialized in [0, 1]. If the input pattern is incorrectly classified, the winner is then moved away from it according to the following equation:

$$\tilde{W}'_{ij} = \tilde{W}_{ij} - \alpha(\tilde{X}_i - \tilde{W}_{ij}). \tag{11}$$

The training of this algorithm is repeated for only one epoch in our experiments. If the epochs are many and learning rate is high, the spreads of the fuzzy numbers of the prototypes will be extended to cover all the feature space. Thus, the training epoch is one and the used learning rate is not high.

5. Experiments

Four experiments, Knowledge Base Evaluator (KBE) [19], Heart Disease Database [35], IRIS data [35], and Balance data [35] are used to illustrate the working of our model. All of these experiments were coded in C language under the software environment of PDP programs [26] on a Sun/SPARC 2 (with 28 MIPS). The inputs of the KBE and the Balance data are linguistic terms. These two experiments illustrate the fundamental applications of the proposed classifier with fuzzy inputs. The *LR*-representation of fuzzy sets can represent various types of information, so crisp inputs can be processed well in the proposed model without any modification. The applications of crisp input are shown in the other two experiments: the heart disease data and the iris data.

5.1. The knowledge-based evaluator

The 18 instances of knowledge-based evaluator (KBE) [19] were used as training instances. Inputs of each feature are linguistic terms. The output of the KBE is Suitability, which takes the term: Good, Fair, or Poor, indicating that the application of the expert system on a domain is good, fair or poor. These training instances and the membership functions used for each attribute are illustrated in Table 1 and Fig. 17, respectively.

This experiment includes two trial cases. Both the cases apply the PECFUH in the first pass and the FENCE in the second pass. The PECFUH takes one epoch and no parameter is used. Then the FENCE takes two epochs and sets the size parameter ρ as 0.7. For instances 2 and 3, we generated each possible term for the don't-care condition. Thus, total 340 training instances are generated from the original 18 training instances.

The first trial uses the randomly selected two thirds of the 340 instances to train the model and

Table 1
The 18 instances of KBE

Instance	Feature						
	Worth	Employee acceptance	Solution available	Easier solution	Teachability	Risk	Suitability (Output)
1	High	Positive	None	None	Frequent	Low	Good
2	Negative	**	**	**	**	**	Poor
3	Low	**	**	**	**	High	Poor
4	Moderate	Neutral	Adequate	Complete	Difficult	High	Poor
5	Low	Negative	None	Partial	Frequent	Low	Poor
6	High	Negative	Partial	None	Difficult	Moderate	Fair
7	High	Positive	Partial	Complete	Frequent	High	Poor
8	High	Positive	Partial	Partial	Possible	Low	Good
9	Low	Positive	Adequate	None	Frequent	Low	Fair
10	High	Negative	Partial	None	Frequent	High	Fair
11	Low	Positive	None	Complete	Difficult	Moderate	Poor
12	Low	Neutral	Adequate	Complete	Frequent	Low	Fair
13	Low	Neutral	None	None	Difficult	Low	Fair
14	Moderate	Positive	Adequate	None	Difficult	High	Poor
15	High	Negative	Adequate	Partial	Frequent	High	Poor
16	High	Negative	Partial	Complete	Possible	Low	Fair
17	Moderate	Negative	None	Partial	Difficult	High	Fair
18	Moderate	Neutral	Adequate	Partial	Difficult	Low	Poor

** Don't care.

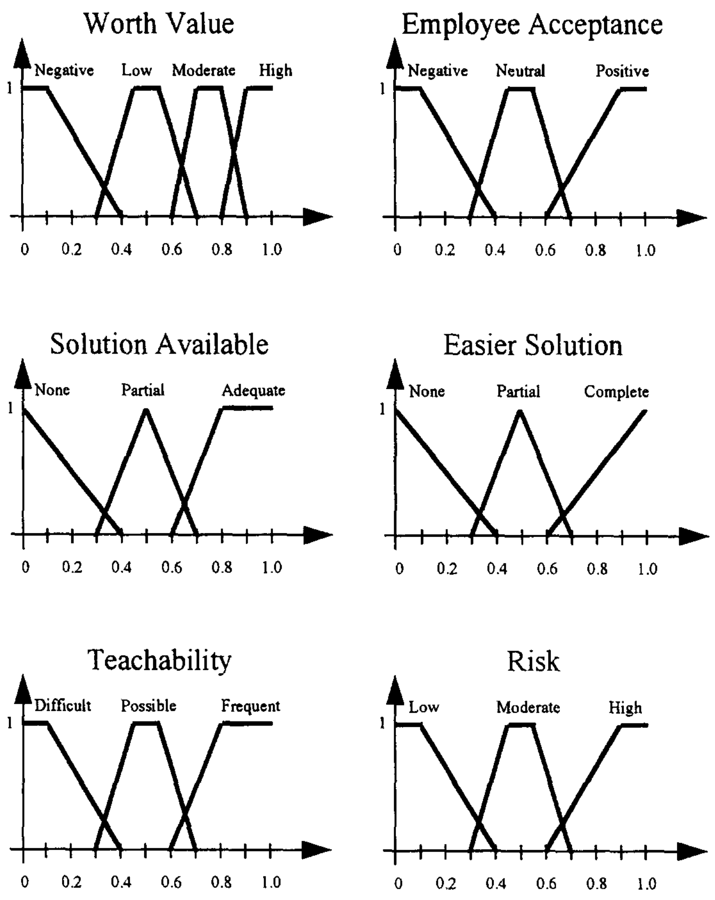


Fig. 17. The membership functions used for KBE.

Table 2
The classification results on KBE

	Number of hidden nodes	Training set errors	Testing set errors	Successful rate of testing set
Training instances: 2/3 * 340 Testing instances: 1/3 * 340	4–8	0 (of 227)	2–5 (of 113)	95.6–98.2%
Training set: 18 (with don't-care) Testing set: 340	6–8	0 (of 18)	1 (of 340)	99.7%

tests the model by the remaining instances. The second trial uses the 18 instances training set with the don't care term represented as LR-type trapezoidal fuzzy number $(0, 1, 0, 0)_{LR}$ to train the classifier, and then uses the 340 instances to test this model.

We try ten times for each case and the average result is shown in Table 2. In case one, most of the misclassified instances of the testing set are the instances without don't-care. The second trial has one misclassification. This misclassification occurs because the pattern generated from the instance 2 is misclassified as "fair". The pattern "*negative, positive, none, none, frequent, low*" is so close to the "fair" class, but its desired-out is class "poor". That means the classifier has very good ability to take care of the don't-care information in this experiment. This ability can help the classifiers to reduce a lot of training time.

5.2. *The heart disease database*

This database created by the Cleveland Clinic Foundation [35] contains 76 attributes, but most published experiments refer to using a subset of 14 of them. Values of each attribute are normalized in [0, 1]. There are two classes: healthy and diseased heart, and 303 instances in this database. However, seven instances were removed because of missing attribute values. In our experiment, two thirds of the instances in each class are randomly selected as the training set. The others are used for testing. Without losing generality, instances are randomly permuted and are divided into training and testing set. We try ten times for this experiment and the average result is shown in Table 3.

The training epochs of the proposed model are three, including one for the first pass and two for the second pass. Note that the FUNLVQ described in Section 4.3 is applied during pass one, and the learning rate α is set as 0.03. The parameter box-size ρ used in FENCE during pass two is set as 0.3. Table 3 shows that the epochs needed fewer steps compared to other models [32]. In addition, our model improves the success rate. Also the average number of generated nodes 18.5 is smaller than our previous FVGE model's 24.8.

5.3. *The IRIS data*

This data set was created by Fisher [35]. It is one of the best known examples to be found in the pattern recognition literature. This problem is to classify a set of iris flowers into three classes: Virginica, Setosa, and Versicolor, using four continuous features: sepal length, sepal width, petal length, and petal width. The data set consists of 150 instances, 50 for each class. One class is linearly separable from the other two, but the other two are overlapping classes. Due to having these characteristics, it is well suited for evaluating the performance of proposed neural network classifier.

The FUNLVQ and FENCE are used to train the classifier. We randomly selected 25 patterns as training patterns in each class. The remaining patterns are used as testing set. The learning rate of FUNLVQ is set to 0.2. After training, 4 to 8 nodes are generated in hidden layer. There is no misclassification in the training set, but average 3.2 misclassifications in the testing set from twenty trials. The comparison with the FMMC classifier [34] is given in Table 4.

5.4. *The balance data*

There are 625 instances in this data set [35] to model psychological experimental results. Each instance is classified into one of three classes, i.e. the balance scale tip to the left, balanced, or tip to the right. The four features illustrated in Fig. 18 are the left weight, the left distance, the right weight, and the right distance. For each instance, if the product of the weight and distance of the left side is

Table 3
The classification results on Heart disease database

Model	Epochs	Training set correctness (%)	Testing set correctness (%)
ID3[a]		100	71.2 ± 5.2
Perceptron[a]	769 ± 1430	63.1 ± 8.3	60.5 ± 7.9
BP[a]	5000 ± 0.0	96.0 ± 1.0	80.6 ± 3.1
FVGE[b]	9.5 ± 4.5	99.5 ± 1.0	81.2 ± 5.0
Proposed model	3	99.75 ± 0.25	81.9 ± 4.0

[a] Gotten from [32].
[b] Our previous work [24].

greater, equal, or smaller than the right's, the instance belongs to the left, balance, or right class, respectively.

In this experiment, the number of generated nodes and training time needed with and without pass one training is shown. Two thirds of the instances in each class are randomly selected as the training set. The others are used for testing. In the first test, we applied the FUNLVQ in pass one and set its learning rate α as 0.008. The best results are shown in Table 5, when the box-size parameter ρ of the FENCE is set to 0.5 during the second pass. The best result means the less number of hidden nodes and less training time needed at the same time. In the second test, the classifier is just trained by pass two algorithm FENCE with $\rho = 0.7$. That is the best result in our experiment. Note that the values in Table 5 are obtained by averaging ten experiments. Furthermore, the epochs of FENCE in the two tests are 2, and the result proves that two epochs are enough to correctly classify the training set. Table 5 shows that this model can save 31.6% generated nodes and 2.6% training time needed, if the pass one training is applied. In addition, the generalization ability improves 5.5%. These results satisfy our original thought.

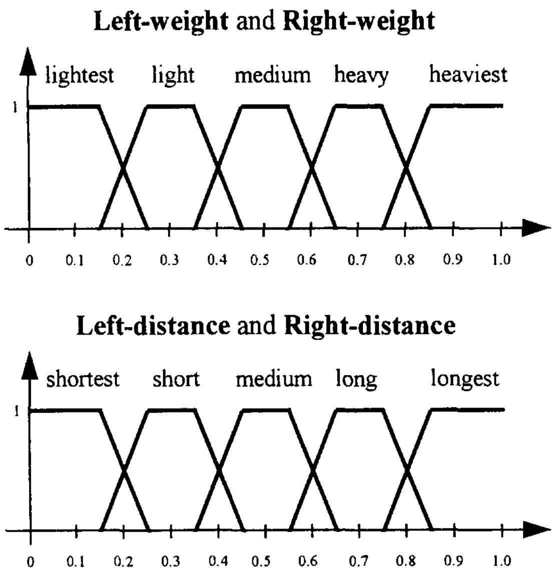


Fig. 18. The membership functions for the balance data.

6. Discussion

The results on the above-mentioned simulations show the workings of our proposed approach. In what follows, we illustrate the characteristics of our model.

6.1. On-line learning

This classifier is able to learn new patterns and even new exemplars without destroying old

Table 4
The classification results on IRIS data

Model	Hidden nodes	Hyperbox size	Misclassifications	Success rate
FMMC [35]	48	0.0175	2	97.3%
Proposed model	4–8 (avg. 5.7)	0.05	1–5 (avg. 3.2)	95.7%

Table 5
The classification results on Balance data

	Hidden nodes	Hyperbox size	Training time needed (ms)	Training set correctness (%)	Testing set correctness (%)
Pass one + Pass two	72.0	0.5	4276.0	100	78.7
Only pass two	105.2	0.7	4389.4	100	73.2

memory [33]. As the pass one learning was not executed when recalling, if the original structure misclassifies a new pattern, an appropriate exemplar node is expanded or a new exemplar node is created automatically by FENCE. Thus all the training data need not be collected beforehand, that is, knowledge can be accumulated incrementally.

6.2. Deal with fuzzy input and low computational load

The input representation of proposed model is trapezoidal fuzzy numbers [39]. Because the fuzzy number can represent various types of information described in Section 2, the fuzzy information can be handled well. Also, the basic operations of the fuzzy numbers are simple and computational load is low in comparison to other model (such as Ishibuchi's model) [16]. The model's computational loads depend on the sampling frequency for the inputs [24].

6.3. Fast learning and low memory requirement

In proposed model, the time needed to train the classifier only by FENCE is more than that needed by FENCE and any one PECFUH or FUNLVQ of pass one learning. Both PECFUH and FUNLVQ are one-epoch learning algorithm. During pass one learning, more epochs take more time and no better result for reducing the number of fuzzy hyperboxes generated by FENCE in all the above experiments. That is because FENCE spends most of the time on overlap checking and weight updating. If the classifier generates a large number of fuzzy hyperboxes, it takes a lot of training time. In addition, the pass one training reduces the number of fuzzy hyperboxes during learning. It speeds up the whole learning process and decreases the memory requirements. The fourth experiment, balance data, shows that the total training time needed after pass one and pass two is shorter than the time needed by only pass two training. In addition, the number of hidden nodes is reduced.

6.4. Good generalization ability for don't-care information

This ability can reduce a great mount training instances and can highly speed up the learning. It is very suitable for many rule-based applications that have don't-care terms in some instances. The experimental results listed in Section 5.1 are very encouraging.

6.5. Few tuning parameters

PECFUH does not need any parameter. In FUNLVQ, only the learning rate parameter, as the other LVQ based algorithms, is used. This parameter's constraint is that the learned fuzzy numbers for all the weights are limited between zero and one. The hyperbox size parameter in FENCE has effect on the generalization ability. In general, small box size can reduce the number of fuzzy hyperboxes but has negative effect on its generalization ability. The actual effect of these parameters depends on the distribution and feeding order of the training data.

6.6. Overlapping classes

FENCE creates some nested fuzzy hyperboxes on the overlapping area of classes. The structure of nested fuzzy hyperboxes depends on the distribution of training instances and their feeding order. In general, the denser the sampling instances on the overlapping area are, the fewer are the misclassifications FENCE makes.

7. Related works

7.1. PECFUH vs. fuzzy min-max classifier

Fuzzy Min-Max Classifier [34] (FMMC) is a single epoch prototype learning algorithm. Due to improving the success rate, the box-size parameter is so small that one node classifies few input patterns (for example, 3 in the IRIS data shown in Table 4). Because of checking the overlaps between nodes, more nodes need much training time. The PECFUH avoids this by setting the number of nodes equal to the number of classes. The comparison among the two algorithms and the other two algorithms described in Section 7.2 is given in Table 6.

Table 6
The comparisons among PECFUH, FENCE, FMMC and NGE

Model	Input vector	Epochs	Distance	Nodes learned	Allow nested nodes	Allow overlap nodes	Parameters	No. of nodes
PECFUH	Fuzzy number	Single	Euclidean	Fuzzy prototype	No	No	None	No. of classes
FENCE	Fuzzy number	Few	Euclidean	Fuzzy exemplar	Yes	Yes/no*	Box-size	***
FMMC	Fuzzy min-max	Single	Hamming	Fuzzy set prototype	No	Yes/no*	Box-size	***
NGE	Real number	Few	Euclidean	Hyper-rectangle exemplar	Yes	Yes	**	***

* The nodes in the same class can overlap.
** Feature adjustment rate.
*** Depends on the overlapping degree of classes.

Table 7
The comparisons between FUNLVQ and LVQ

Model	Learning epochs	Inputs	Operations
FUNLVQ	One	Trapezoidal fuzzy numbers	Fuzzy numbers' operations
LVQ-based model	10–10 000	Real numbers	Real numbers' operations

7.2. FENCE vs. NGE theory

Nested Generalized Exemplar theory [31] (NGE) has a special parameter – feature adjustment rate. It is used to adjust the weights on features to improve the noise tolerance. Because trapezoidal fuzzy number can represent the crisp interval, we can say the FENCE is a generalization of the NGE model.

7.3. FUNLVQ vs. LVQ

The FUNLVQ just takes one epoch on data. This is to avoid the operations on trapezoidal fuzzy number such that the spread of trapezoidal fuzzy number will increase a great deal. The learning rate is set as a small value based on the same reason. The comparison between FUNLVQ and other LVQ-based algorithms [18, 20] is given in Table 7.

8. Conclusion

Neural networks adopt numerical computation with fault tolerance, massively parallel computing and trainable property, so they are suitable for classification algorithm. The contribution of fuzzy set theory lies in their methods to model and process uncertain or ambiguous data, so often encountered in real life. Therefore, to enable a neural network classifier to handle real life situation, one may incorporate the fuzzy set theory into neural networks to get precise and fast classification with fuzzy inputs.

8.1. Results

This on-line adaptive model combines the approaches of prototype classifiers and exemplar classifiers. As PECFUH or FUNLVQ is applied to train the prototype nodes, the proposed classifier takes the advantages of fast learning and low memory requirement. In addition, the fuzzy information can be dealt well with less amount of the computational effort. Besides, exemplars of this model are learnt according to the Fuzzy Exemplar Nested Creation and Expansion (FENCE), so nonlinear separable and overlapping classes can be easily classified. Also, the ability of exception handling and the generalization ability for don't-care information are very encouraging. Furthermore, the three proposed learning algorithms need few parameters and take few epochs.

8.2. Future works

(1) Construct a classifier with special nodes in the hidden layer. These nodes can adapt to prototype nodes or exemplar nodes. That is, the classifier would be auto-configured as a prototype-based classifier or an exemplar-based classifier depending on the different applications.

(2) Provide the fuzzy class outputs to make the soft decision for some suitable applications.

(3) Find the better similarity function between two fuzzy numbers to omit the defuzzification process such that the learning and recalling of the classifier will speed up.

Acknowledgements

The author would like to thank the reviewers for their helpful suggestions on revising this paper. Also this work is supported by National Science Council, Taiwan, under a grant NSC 85-2213-E011-057.

References

[1] V.U. Abe and M.S. Lan, A neural-network-based fuzzy classifier, *IEEE Trans. Systems Man Cybernet.* **25** (1995) 353–361.

[2] M.A. Abounasr and M.A. Sidahmed, Fast learning and efficient memory utilization with a prototype based neural classifier, *Pattern Recognition* **28** (1995) 581–594.

[3] N.P. Archer and S. Wang, Fuzzy set representation of neural network classification boundaries, *IEEE Trans. Systems Man Cybernet.* **21** (1991) 735–742.

[4] J.C. Bezdek, Fuzzy models and computational neural networks, *Tutorial of IEEE Conf. on Neural Networks for Ocean Engineering* (1991) 291–300.

[5] J.C. Bezdek, C.C. Taso and N.R. Pal, Fuzzy Kohonen clustering networks, *Proc. Internat. Conf. on Fuzzy Systems* (1992) 1035–1043.

[6] C.L. Chang, Finding prototypes for nearest neighbor classifiers, *IEEE Trans. Comput.* **C-23** (1974) 1179–1184.

[7] K.H. Chen, H.L. Chen and H.M. Lee, A multiclass neural net classifier with fuzzy teaching inputs, to appear in *EUFIT '95 3rd European Congress on Intelligent Techniques and Soft Computing* (1995) 387–391.

[8] S. Cho, O.K. Ersoy and M. Lehto, An algorithm to compute the degree of match in fuzzy systems, *Fuzzy Sets and Systems* **49** (1992) 285–299.

[9] E. Cox, Integrating fuzzy logic into neural nets, *AI Expert* **6** (1992) 43–47.

[10] K. Demura, M. Kajiura and Y. Anzai, The SOLAR algorithm, *ICNN '94 IEEE Internat. Conf. on Neural Networks* **1** (1994) 125–130.

[11] F. Dicesare, Z. Sahnoun and P.P. Bonissone, Linguistic summarization of fuzzy data, *Inform. Sci.* **52** (1990) 141–152.

[12] D. Dubois and H. Prade, Fuzzy real algebra: some results, *Fuzzy Sets and Systems* **2** (1979) 327–348.

[13] M.M. Gupta and J. Qi, On fuzzy neuron models, *IJCNN '91 Internat. Joint Conf. on Neural Networks* (1991) 431–436.

[14] C.J. Harris, C.G. Moore and M. Brown, *Intelligent Control Aspects of Fuzzy Logic and Neural Nets* (World Scientific, Singapore, 1993).

[15] Y. Hayashi, A neural expert system using fuzzy teaching input, *Proc. Internat. Conf. on Fuzzy Systems* (1992) 485–491.

[16] H. Ishibuchi, R. Fujioka and H. Tanaka, An architecture of neural networks for input vectors of fuzzy numbers, *Proc. Internat. Conf. on Fuzzy Systems* (1992) 1293–1300.

[17] C.I. Kao and Y.H. Kuo, A neural network model based on fuzzy classification concept, *IJCNN Internat. Joint Conf. on Neural Networks*, Vol. 2 (1992) 727–732.

[18] N.B. Karayiannis and P.I. Pai, A fuzzy algorithm for learning vectors quantization, *1994 IEEE Internat. Conf. on SMC* (1994) 126–131.

[19] R. Keller, *Expert System-Development & Approaches* (Prentice-Hall, Englewood Cliffs, NJ, 1987).

[20] T. Kohonen, *Self-Organization and Associative Memory* (Springer, New York, 1989).

[21] B. Kosko, *Neural Networks and Fuzzy Systems – A Dynamical Systems Approach to Machine Intelligent* (Prentice-Hall, Englewood Cliffs, NJ, 1992).

[22] H.K. Kwan and Y. Cai, A fuzzy neural network and its application to pattern recognition, *IEEE Trans. Fuzzy Systems* **2** (1994) 185–193.

[23] H.M. Lee and C.S. Lai, Supervised fuzzy ART: training of neural network for pattern classification via combining supervised and unsupervised learning, *1993 IEEE Internat. Conf. on Neural Networks*, Vol. 1 (1993) 323–328.

[24] H.M. Lee and W.T. Wang, A neural network architecture for classification of fuzzy inputs, *Fuzzy Sets and Systems* **63** (1994) 159–173.

[25] R.P. Lippmann, Pattern classification using neural networks, *IEEE Comm. Magazine* **40** (1989) 185–234.

[26] J.L. McClelland and D.E. Rumelhart, *Explorations in Parallel Distributed Processing* (MIT Press, Cambridge, MA, 1988).

[27] S.C. Newton and S. Mitra, An adaptive fuzzy system for control and clustering of arbitrary data patterns, *IEEE Internat. Conf. on Fuzzy Systems* (1992) 363–370.

[28] S.K. Pal and S. Mitra, Multilayer perceptron, fuzzy sets, and classifier, *IEEE Trans. Neural Networks* **3** (1992) 683–697.

[29] W. Pedrycz and H.C. Card, Linguistic interpretation of self-organizing maps, *Proc. Internat. Conf. on Fuzzy Systems* (1992) 371–378.

[30] D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing*, Vol. 1 (MIT Press, Cambridge, MA, 1987).

[31] S. Salzberg, A nearest hyperrectangle learning method, *Mach. Learning* **6** (1991) 251–276.

[32] J.W. Shjvlik, R.J. Mooney and F.G. Towell, Symbolic and neural learning algorithm: An experimental comparison, *Machine Learning* **6** (1991) 111–145.

[33] P.K. Simpson, Fuzzy min-max neural networks – part 1: Classification, *IEEE Trans. Neural Networks* **3** (1992) 776–786.

[34] P.K. Simpson, Fuzzy min-max neural networks – part 2: Clustering, *IEEE Trans. on Fuzzy Systems* **1** (1993) 32–45.

[35] UCI repository of machine learning database, *ftp://ics.uci.edu/pub/machine-learning-database.*

[36] R.R. Yager and L.A. Zadeh, Eds., *An Introduction to Fuzzy Logic Applications in Intelligent Systems* (Kluwer, Boston, MA, 1993).

[37] T. Yamakawa and M. Furukawa, A design algorithm of membership functions for a fuzzy neuron using example-based learning, *Proc. Internat. Conf. on Fuzzy Systems* (1992) 75–82.

[38] L.A. Zadeh, Fuzzy sets, *Inform. and Control* **8** (1965) 338–353.

[39] H.-J. Zimmermann, *Fuzzy Set Theory and Its Applications* (Kluwer, Boston, 1991).