Contributed article

A neuro-fuzzy framework for inferencing

Sukumar Chakraborty[a], Kuhu Pal[b], Nikhil R. Pal[c,*]

[a]Interra Information Technologies (India) Pvt. Ltd, 223-226, First floor, SDF Building, Sector-V, Salt Lake City, Calcutta 700091, India
[b]Pragatinagar, Chinsurah, Hooghly, WB 712102, India
[c]Electronics and Communication Sciences Unit (ECSU), Indian Statistical Institute, 203 BT Road, Calcutta 700035, India

Received 17 January 2001; revised 29 October 2001; accepted 29 October 2001

Abstract

Earlier we proposed a connectionist implementation of compositional rule of inference (COI) for rules with antecedents having a single clause. We first review this net, then generalize it so that it can deal with rules with antecedent having multiple clauses. We call it COIN, the compositional rule of inferencing network. Given a relational representation of a set of rules, the proposed architecture can realize the COI. The outcome of COI depends on the choice of both the implication function and the inferencing scheme. The problem of choosing an appropriate implication function is avoided through neural learning. COIN can automatically find a 'good' relation to represent a set of fuzzy rules. We model the connection weights so as to ensure learned weights lie in [0,1]. We demonstrate through extensive numerical examples that the proposed neural realization can find a much better representation of the rules than that by usual implication and hence results in much better conclusions than the usual COI. © 2002 Elsevier Science Ltd. All rights reserved.

Keywords: Fuzzy reasoning; Neuro-fuzzy system; Compositional rule of inferencing; Fuzzy relation

1. Introduction

Fuzzy sets (Zadeh, 1965) are generalization of crisp sets and have greater flexibility to capture faithfully various aspects of incompleteness or imperfection in information, and can be used to model human reasoning/thinking process. Let A and B be two fuzzy sets defined on the universes X and Y, respectively. Consider a simple rule: *If* x *is* A *then* y *is* B. Now given the fact, *If* x *is* A', we like to infer y *is* B', such that the closer the A' to A, the closer would be B' to B, where A' and B' are fuzzy sets on X and Y, respectively. Thus, the problem of fuzzy inferencing is as follows:

Premise 1: If x *is* A *Then* y *is* B
Premise 2: x *is* A'
Conclusion: y *is* B'

Neural networks (NNs) (Haykin, 1994), like fuzzy logic systems, are excellent at developing systems that can perform information processing similar to what our brain does. The concept of artificial NNs has been inspired by biological NNs, which enjoy the following characteristics:

- They are non-linear devices, highly parallel, robust and fault tolerant.
- They have a built-in capability to adapt its synaptic weights to changes in the surrounding environment.
- They can easily handle imprecise, fuzzy, noisy and probabilistic information.
- They can generalize from known tasks or examples to unknown ones.

Artificial NN is an attempt to mimic some or all of these characteristics. Although the development of NN is inspired by the model of brains, its purpose is not just to mimic a biological neural net, but to use principles from nervous systems to solve complex problems in an efficient manner.

Both fuzzy systems and NN have been successfully used in many applications (Haykin, 1994; Lee, 1990; Scharf & Mandic, 1985; Self, 1990; Sugeno, 1985; Suh & Kim, 1994; Zadeh, 1988). Apart from the learning ability of NN, it has inherent robustness and parallelism. Fuzzy logic, on the other hand, has the capability of modeling vagueness, handling uncertainty, and can support human type reasoning. Integration of these two soft computing paradigms (often known as neuro-fuzzy computing) is, therefore, expected to result in more intelligent systems (Gupta & Rao, 1994; Pal & Pal, 1996).

In the recent past extensive research work is going on for integration of fuzzy systems with NNs (Hayashi, Buckley,

& Czogala, 1992; Ishibuchi, Fujioka, & Tanaka, 1993; Keller, 1990, 1993; Keller & Tahani, 1992a,b; Keller & Yager, 1989; Keller, Krishnapuram, & Rhee, 1992a; Keller, Hayashi, & Chen, 1993; Nie & Linkens, 1992, 1995; Shann & Fu, 1995; Takagi & Hayashi, 1991). The objective here is to combine the expert knowledge or operators' experience and reasoning ability of fuzzy systems with the computational capabilities of NNs in an efficient manner to solve complex problems (Hayashi et al., 1992; Ishibuchi et al., 1993; Keller, 1990, 1993; Keller & Tahani, 1992a,b; Keller & Yager, 1989; Keller, Yager, & Tahani, 1992b; Keller et al., 1992a, 1993; Nie & Linkens, 1992, 1995; Pal, Pal, & Keller, 1998; Shann & Fu, 1995; Takagi & Hayashi, 1991). The integration of fuzzy logic and NN often is done in two ways—a fuzzy system implemented in a neural architecture (neural fuzzy system) and a NN equipped with the capability of handling fuzzy information (fuzzy NN). Several attempts have been made in both of these directions. Of course, there are several hybrid systems which may not be categorized strictly to either of these two classes.

Keller et al. (1992a) proposed a neural implementation of fuzzy reasoning. Pal et al. (1998) analyzed the system by Keller et al. (1992a) and derived learning rules for finding good parameters for this network. For a special case, Pal et al. showed how the optimal parameters can be computed, and demonstrated the method with some examples. A new architecture is also proposed by Pal et al. (1998) which exhibits better characteristics than the network by Keller et al. (1992a).

The philosophy behind the models proposed by Keller et al. (1992a) and Pal et al. (1998) is a kind of similarity-based reasoning. The more the similarity between the antecedent of a rule and the given fact, the more close would be the inferred conclusion or the consequent of the rule. Pal and Pal (1999) proposed a scheme for realization of compositional rule of inference (COI) in a neural framework. But the method by Pal and Pal (1999) can deal with only simple rules with one antecedent clause. Here, we generalize the system to implement COI with several clauses in the antecedent, in a connectionist framework. We explain how the connection weights should be modeled so that a learning rule can be designed to ensure the weights to lie in [0,1]. Note that, this is not a multi-layer perceptron which loses the fuzzy reasoning structure and acts as a black-box type function approximator.

2. Neural realization of compositional rule of inference

2.1. Compositional rule of inference

Let $A = \{\mu_A(x_i)/x_i, i = 1, 2, \ldots, n_A; x_i \in X\}$ be a fuzzy set defined on X, $B = \{\mu_B(y_j)/y_j, j = 1, 2, \ldots, n_B; y_j \in Y\}$ be a fuzzy set defined on Y and let A and B define a rule: *If x is A Then y is B*. Now using a T-norm, the fuzzy rule can be written as $A \rightarrow B = a$ relation $R = A \times B$ on $X \times Y$, such that $R = \sum_{X\times Y} \mu_R(x_i, y_j)/(x_i, y_j) = \sum_{X\times Y} T\{\mu_A(x_i), \mu_B(y_j)\}$.

Now given a fact x is $A' = \{\mu'_A(x_i), i = 1, 2, \ldots, n_A\}$, the conclusion B' of y is B' can be obtained by the composition of A' and R. The composition of A' and R results in a fuzzy set B' defined on Y as

$$B' = A' \circ R = \text{Proj}(\text{Ce}(A') \cap R) \text{on } Y. \tag{1}$$

Here, Ce and Proj are the cylindrical extension and projection operators, respectively (Klir & Folger, 1988), and $\circ$ denotes the composition operator. If the intersection in Eq. (1) is performed with the minimum operator and projection with the maximum operator, then we get

$$\mu'_B(y_j) = \max_{x_i} \min\{\mu'_A(x_i), \mu_R(x_i, y_j)\}. \tag{2}$$

This is known as max–min composition. Similarly, the max–prod composition is defined as:

$$\mu'_B(y_j) = \max_{x_i} \{\mu'_A(x_i) \cdot \mu_R(x_i, y_j)\}. \tag{3}$$

Usually, any T-norm can be used to perform the intersection operation resulting in max–T composition.

Given a set of N rules R_i, $i = 1, \ldots, N$ the composition can be done with respect to each rule separately, and then the different B'_i can be aggregated to get a resultant conclusion B'. On the other hand, the relations representing different rules can be aggregated first and then the composition operator can be directly applied to R. In the later case, the composite relation R can be obtained as

$$R = \bigcup_{i=1}^{N} R_i = \bigcup_{i=1}^{N} A_i \times B_i. \tag{4}$$

Normally, $B' = A \circ R \neq B$, where $\circ$ is a composition operator implemented through max and a T-norm. This may be the case even when R represents only a single and simple rule like *If x is A Then y is B*. This is a very undesirable property. Moreover, the use of COI raises two important issues: which function should be used for implication and which T-norm (Klir & Folger, 1988) should be used for inferencing. Even for a given inferencing scheme, say max–min COI, the choice of the implication function has a significant impact on the conclusion drawn. Lee (1990) explains this issue.

This problem has been addressed by many researchers (Mamdani, 1977; Mizumoto, 1985; Mizumoto & Zimmerman, 1982; Mizumoto, Fukami, & Tanaka, 1979a,b,c, 1980; Zadeh, 1975). Zadeh (1975) suggested two relational representations for $A \rightarrow B$

$$\mu_R(x_i, y_j)/(x_i, y_j) = (\mu_A(x_i) \wedge \mu_B(y_j)) \vee (1 - \mu_A(x_i)), \quad \forall (x_i, y_j), \tag{5}$$

and

$$\mu_R(x_i, y_j)/(x_i, y_j) = 1 \wedge \{(1 - \mu_A(x_i) + \mu_B(y_j))\}, \quad \forall (x_i, y_j). \tag{6}$$

Mamdani (1977) defined a relation as:

$$\mu_R(x_i, y_j)/(x_i, y_j) = \mu_A(x_i) \wedge \mu_B(y_j), \qquad \forall(x_i, y_j); \tag{7}$$

while Mizumoto et al. (Mizumoto, 1985; Mizumoto & Zimmerman, 1982; Mizumoto et al., 1979a,b,c, 1980) introduced many ways of computing R including

$$\mu_R(x_i, y_j)/(x_i, y_j) = \begin{cases} 1 & \text{if } \mu_A(x_i) \leq \mu_B(y_j), \\ 0, & \text{otherwise,} \end{cases} \tag{8}$$

and

$$\mu_R(x_i, y_j)/(x_i, y_j) = \begin{cases} 1 & \text{if } \mu_A(x_i) \leq \mu_B(y_j), \\ \mu_B(y_j), & \text{otherwise.} \end{cases} \tag{9}$$

They also discussed the use of implication rules of multi-valued logic to define the relation R. For example

$$\mu_R(x_i, y_j)/(x_i, y_j) = (1 - \mu_A(x_i)) \wedge \mu_B(y_j), \tag{10}$$

and

$$\mu_R(x_i, y_j)/(x_i, y_j) = \begin{cases} 1 & \text{if } \mu_A(x_i) \leq \mu_B(y_j), \\ \dfrac{\mu_B(y_j)}{\mu_A(x_i)}, & \text{otherwise.} \end{cases} \tag{11}$$

Mizumoto and Zimmerman (1982) compared 15 reasoning methods; i.e. 15 different fuzzy relations for *If x is A Then y is B* using the max–min composition operator. They have considered both generalized modus-ponens and generalized modus-tollens. Their investigation showed that Zadeh's implication function does not perform satisfactorily; i.e. does not produce a conclusion that conforms to our intuition, while Mamdani's method is not bad. But the implication functions proposed by Mizumoto et al. such as Eqs. (8) and (9) are satisfactory. The generalization of implication rules of multi-valued logic, such as Eqs. (10) and (11) are not very good also. Thus, we see that it is very difficult to pick up a particular implication operator which will produce conclusions, that will consistently agree with our intuition.

In this investigation, we restrict ourselves only to generalized modus-ponens and instead of trying to pick a suitable implication operator, given the T-norm for COI, we will try to find an 'optimal' relation R that can produce conclusions conforming to our intuition. In other words, when the T-norm for the COI is fixed, we want to find the best representation of the relation R without binding ourselves to any particular implication function. In addition to that, we want to realize it through a NN so that we can exploit the usual benefits of connectionist models.

So far, we have discussed about rules whose antecedent has only one clause, now let us consider rules with antecedents having more than one clause. If the antecedent has two clauses, then the rules will look like *If x is A and y is B Then z is C*, where

$A = \{\mu_A(x_i)/x_i, i = 1, 2, \ldots, n_A; x_i \in X\}$ be a fuzzy set defined on X,

$B = \{\mu_B(y_i)/y_i, i = 1, 2, \ldots, n_B; y_i \in Y\}$ be a fuzzy set defined on Y, and

$C = \{\mu_C(z_i)/z_i, i = 1, 2, \ldots, n_C; z_i \in Z\}$ be a fuzzy set defined on Z.

The relation R now becomes $R = A \times B \times C$ such that

$$R = \sum_{X \times Y \times Z} \mu_R(x_i, y_j, z_k)/(x_i, y_j, z_k)$$

$$= \sum_{X \times Y \times Z} T\{\mu_A(x_i), \mu_B(y_j), \mu_C(z_k)\}.$$

2.2. COIN: compositional rule of inference net

Before presenting the net with two clauses in the antecedent, let us first discuss how COI with rules having single clause in the antecedent can be realized. Fig. 1 shows the architecture for COIN when the rules have only one antecedent clause. It is a two layered net, the input layer has n_A nodes, $I_i, i = 1, \ldots, n_A$ and the output layer has n_B nodes, O_i, $i = 1, \ldots, n_B$. Here, n_A is the number of quantization levels of an antecedent fuzzy set and n_B is the same for a consequent fuzzy set. Each node of the input layer is connected to all n_B nodes in the output layer.

Let $w_{ij} = r_{ij}$ be the connection weight between the ith input node and jth output node. Thus, the jth node in the output layer is connected to the set of n_A input nodes and these weights correspond to the jth column of the relation matrix R. The relation R is now represented by the connection weights. Let us denote the input to the ith node, I_i, in the input layer by m_i. An input layer node simply transfers the input value unattenuated to the second layer nodes. The activation function of a second (or output) layer node is defined as:

$$o_j = \max_i\{T\{w_{ij}, m_i\}\}. \tag{12}$$

Here, T is a T-norm. With these activation functions, it is easy to check that if $W(=R)$ is a relation representing the rule *If x is A Then y is B*, then if the

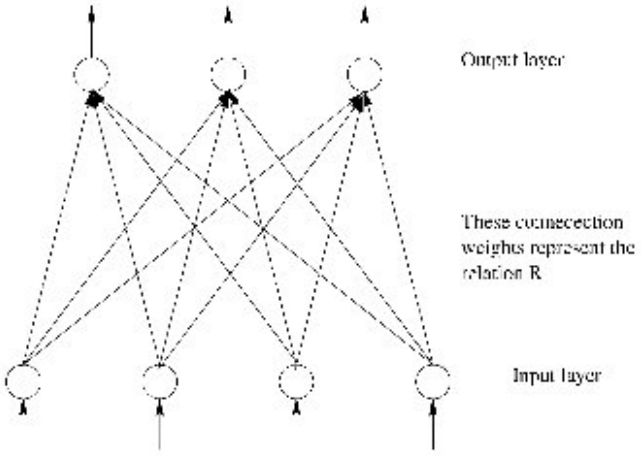

Fig. 1. Architecture of COIN for rules with single clause in the antecedent.

fuzzy set $A' = \{\mu'_A(x_i), i = 1, \ldots, n_A\}$ representing the fact x *is* A' is given to the network as input, the network will produce an output fuzzy set $B' = \{o_j = \mu_B(y_j), j = 1, \ldots, n_B\}$ where $B' = A' \circ R$. A straight forward substitution will yield the result. So if we set the connection weights properly we can realize COI by COIN (Pal & Pal, 1999). Normally, R is generated as the Cartesian product

$$R = A \times B = [r_{ij} = T(\mu_A(x_i), \mu_B(y_j))]. \tag{13}$$

If there are more than one rule, then the final R is obtained as the union of R_is, where R_i is the relation matrix corresponding to the ith rule; $R = \bigcup_i R_i$.

What can we achieve out of this? It is a neural realization of COI. It enjoys features like parallelism, robustness and learning capability of NNs. But the original problem of COI, i.e. the output may be quite off from the desired one still persists. We already mentioned about the extreme example: If $R =$ *If* x *is* A *then* y *is* B and the fact is x *is* A then the conclusion $B' = A \circ R$ may not be (usually is not) equal to B. There are two possible factors that contribute to this problem: (i) the implication operator used, i.e. the method of generating R from A and B and (ii) the method of inferencing (the COI). We want to find an appropriate relation R exploiting the learning capability of NN.

For notational simplicity, we represent a rule *R: If x is A Then y is B* by the pair (A, B). Suppose we want to learn the rule or relation (A, B), given some training data (A_i, B_i), $i = 1, \ldots, N$, where the pair (A_i, B_i) represents a rule of the form *If* x *is* A_i *Then* y *is* B_i and each A_i is defined on X and each B_i is defined on Y; $A_i = \{\mu_{A_i}(x_k), k = 1, \ldots, n_A\}$ and $B_i = \{\mu_{B_i}(y_k), k = 1, \ldots, n_B\}$. Again for clarity, we write $\mu_{A_i}(x_k) = a_{i,k}$, $k = 1, \ldots, n_A$; $i = 1, \ldots, N$ and $\mu_{B_i}(y_k) = b_{ik}$, $k = 1, \ldots, n_B$; $i = 1, \ldots, N$. We will discuss the issue of generation of training data in the implementation section. Let the conclusion obtained from x *is* A_i be $B'_i = \{o_{ij} = b'_{ij}, j = 1, \ldots, n_B\}$, $i = 1, \ldots, N$; $B'_i = A_i \circ R$, $i = 1, \ldots, N$. Note that, in the present context B'_i is the output vector of the network when the connection weights represent the relation R and A_i is given as the input.

In order to get an optimal set of connection weights and hence an optimal relation R, we minimize the error E,

$$E = \sum_{i=1}^{N} \|B'_i - B_i\|^2 = \sum_{i=1}^{N} \sum_{j=1}^{m} (b'_{ij} - b_{ij})^2$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{m} \left(\max_k \{a_{ik} \cdot w_{kj}\} - b_{ij} \right)^2. \tag{14}$$

In Eq. (14), we assumed the max–prod (T-norm = product) inferencing. Use of gradient descent to minimize Eq. (14) with respect to w_{kj} poses some problems because of two reasons: use of the operator max in Eq. (14) and gradient descent, as such, does not guarantee that every w_{ij} will lie in [0,1] even if we start so.

The first problem is easily solved by using any soft version of the max operator. We use

$$\mathrm{SM}(x_1, \ldots, x_n) = \frac{\sum_{i=1}^{n} x_i \exp(-s x_i)}{\sum_{i=1}^{n} \exp(-s x_i)}. \tag{15}$$

There are other possible choices too. Note that $\lim_{s \to -\infty} \mathrm{SM}(x_1, \ldots, x_n) = \max\{x_1, \ldots, x_n\}$. Therefore, by choosing a reasonably big (negative) value for s, practically, we can realize the max operator and yet we can use calculus to derive the learning rules. Thus, the activation function (12) of a second layer node becomes

$$o_j = \mathrm{SM}\{w_{ij} \cdot m_i; i = 1, \ldots, n_A\}. \tag{16}$$

To solve the second problem we model the connection weights $r_{ij} = w_{ij} = \exp(-K_{ij}^2)$, where K_{ij} is unrestricted in sign. With this choice, irrespective of the sign and magnitude of K_{ij}, w_{ij} will always be in [0,1], i.e. $0 \le w_{ij} \le 1$. We can now learn K_{ij} with gradient descent search but we use $w_{ij} = \mathrm{e}^{(-K_{ij})^2}$ as the connection weight. Using Eqs. (15) and (16) the output of the jth node, when the input to the net is $a = \{a_1, a_2, \ldots, a_{n_A}\}$, becomes

$$o_j = b'_j = \frac{\sum_{k=1}^{n_A} a_k w_{kj} \exp(-s a_k w_{kj})}{\sum_{k=1}^{n_A} \exp(-s a_k w_{kj})}, \tag{17}$$

$$o_j = b'_j = \frac{\sum_{k=1}^{n_A} a_k \exp\left(-K_{kj}^2\right) \exp\left(-s a_k \exp\left(-K_{kj}^2\right)\right)}{\sum_{k=1}^{n_A} \exp\left(-s a_k \exp\left(-K_{kj}^2\right)\right)}. \tag{18}$$

Thus, the learning rule for K_{ij} can be shown (Pal & Pal, 1999) to be

$$K_{ij}(t) = K_{ij}(t-1) - \eta_K (b'_{ij} - b_{ij}) \sum_{l=1}^{N} \frac{a_{li} \exp\left(-s a_{li} \, \mathrm{e}^{(-K_{ij}^2(t-1))}\right) K_{ij}(t-1) \mathrm{e}^{(-K_{ij}^2(t-1))} \left[-B_l + B_l s \, \mathrm{e}^{(-K_{ij}^2(t-1))} a_{li} - s A_l \right]}{B_l^2}, \tag{19}$$

where

$$A_l = \sum_{k=1}^{n} a_{lk} w_{kj} \exp(-s a_{lk} w_{kj}), \tag{20}$$

and

$$B_l = \sum_{k=1}^{n} \exp(-s a_{lk} w_{lj}). \tag{21}$$

Here, η_K is the learning co-efficient of K_{ij}. Eq. (19) is a batch update formula. For an online update, the formula is the same as Eq. (19) except the summation over all points $l = 1$ to N will not be there. Note that, replacement of the 'max' operator in the error function (14) by its softer version SM in Eq. (15) makes the error function 'smooth', as the first and second derivatives exist everywhere. So the steepest descent will converge to either a local minima or a saddle point, which can be checked using the Hessian matrix at that point. Minimization of this error function will lead to a network that can draw conclusions conforming to our intuition.

One might be tempted to call the COIN as a one layer perceptron. But that is NOT the case because COIN uses COI (max–prod) to get the conclusion and of course does not use the sigmoidal activation function.

Once the values of K are obtained, the final relation R can be obtained and then we can calculate the values of $B_i' = \{b_{ij}'\}$ using Eq. (17).

In real life, antecedents with single clause are rarely seen. So we must generalize our network so that it can handle rules having arbitrary number of clauses in the antecedent. Next, we propose a modification in the architecture of the network so that it can tackle rules with two clauses in the antecedent. Then we will show how we can extend the net to learn rules with more than two clauses in the antecedent.

Let us consider a rule R: *If x is A and y is B Then z is C*, as described earlier, where $A = \{\mu_A(x_i)/x_i, i = 1, 2, \ldots, n_A; x_i \in X\}$ be a fuzzy set defined on X, $B = \{\mu_B(y_i)/y_i, i = 1, 2, \ldots, n_B; y_i \in Y\}$ be a fuzzy set defined on Y, and $C = \{\mu_C(y_i)/y_i, i = 1, 2, \ldots, n_C; z_i \in Z\}$ be a fuzzy set defined on Z.

For rules with two clauses in the antecedent, we propose the network architecture, as shown in Fig. 2. The net has three layers, input layer, one hidden layer, and then the output layer. The input layer has $n_A + n_B$ nodes divided into two logical groups, the first group has n_A nodes and the second group has n_B nodes. These nodes actually represent the two input fuzzy sets A and B. We will call them group I_A and group I_B. The individual nodes will be referred as $I_{A_1}, I_{A_2}, \ldots, I_{A_{n_A}}, I_{B_1}, I_{B_2}, \ldots, I_{B_{n_B}}$. Second layer has $n_A \cdot n_B$ nodes logically partitioned into n_A groups, each group contains n_B nodes. We call these groups as group H_1, group $H_2, \ldots$, and group H_{n_A}. The individual nodes are recognized as $H_{1_1}, H_{1_2}, \ldots, H_{1_{n_B}}$; $H_{2_1}, H_{2_2}, \ldots, H_{2_{n_B}}; \ldots; H_{n_{A_1}}, H_{n_{A_2}}, \ldots, H_{n_{A_{n_B}}}$. The third layer or the output layer has n_C nodes to represent the output fuzzy set. The output nodes are referenced as $O_1, O_2, \ldots, O_{n_C}$. In Fig. 2, $n_A = 3$, $n_B = 2$ and $n_C = 3$. The three groups of nodes in the second layer are indicated by three different shades. The connections between layers are made as described later:

1. The ith node of group I_A in the input layer is connected to

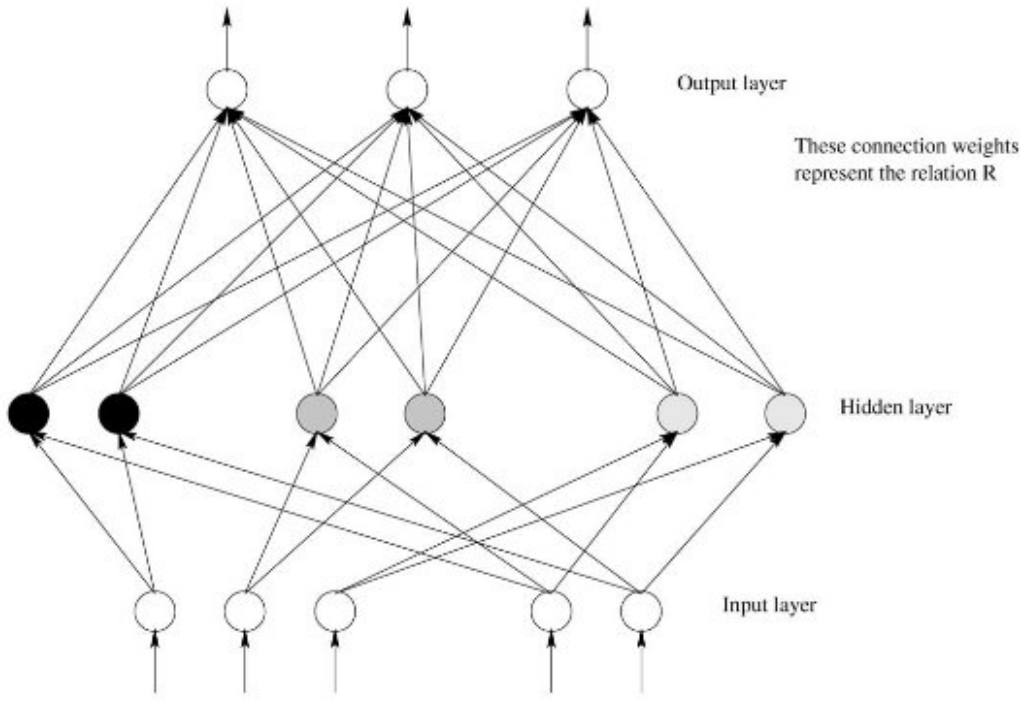

Fig. 2. Architecture of COIN for rules with two clauses in the antecedent.

all the n_B nodes of group H_i in the hidden layer. Symbolically, each of the nodes I_{A_i}; $i = 1, 2, \ldots, n_A$ is connected to nodes H_{i_j}; $j = 1, 2, \ldots, n_B$.

2. The jth node of group I_B in the input layer is connected to the jth node of every group, group H_1 to group H_{n_A}, i.e. each of the nodes I_{B_j}; $j = 1, 2, \ldots, n_B$ is connected to nodes H_{i_j}; $i = 1, 2, \ldots, n_A$.
3. Each node in the hidden layer is connected to every node in the output layer, i.e. each of the nodes H_{i_j}; $i = 1, 2, \ldots, n_A$; $j = 1, 2, \ldots, n_B$ is connected to the nodes O_k; $k = 1, 2, \ldots, n_C$.
4. The strength of the connections from the input layer to the hidden layer are all set to unity ($= 1$).
5. The activation function of the nodes in the hidden layer is a T-norm. For example, if it is the 'min' function, then it selects the minimum of the inputs and pass it as output. This layer actually computes the relation A *AND* B. The number of outputs from the hidden layer is $n_A \cdot n_B$.
6. The activation function of the nodes in the output layer is the soft–max as described earlier.
7. The number of connections between the hidden and the output layer is $n_A \cdot n_B \cdot n_C$. These connection weights actually represent the relation $R = A \times B \times C$. The connection strengths in this layer are learned using gradient descent technique. We model these weights by $\exp(-K_{ijk}^2)$, so that it remains positive and less than one after training, as described earlier.

The connection weights between the hidden and the output layer are updated by gradient descent technique, which is briefly described later.

Let E be the instantaneous error function for some input (A_i, B_i), for notational simplicity we drop the index i. Then we can write

$$E = \sum_{k=1}^{n_c} (o_k - r_k)^2, \tag{22}$$

where o_k is the actual output of the node O_k and r_k is the expected output of that node, $k = 1, 2, \ldots, n_c$.

Thus, we get

$$\frac{\partial E}{\partial w_{ijk}} = 2(o_k - r_k)\frac{\partial o_k}{\partial w_{ijk}}. \tag{23}$$

Here, $W_{ijk} = (\exp(-w_{ijk}))^2$ is the connection weight between the nodes H_{i_j} and O_k, and

$$o_k = \frac{\sum_{i=1}^{n_A}\sum_{j=1}^{n_B} I_{ij} W_{ijk} \exp(-sI_{ij}W_{ijk})}{\sum_{i=1}^{n_A}\sum_{j=1}^{n_B} \exp(-sI_{ij}W_{ijk})}. \tag{24}$$

After some algebraic manipulation we get

$$\frac{\partial o_k}{\partial w_{ijk}} = 2W_{ijk}I_{ij}w_{ijk}\exp(-sI_{ij}w_{ijk})\frac{(sI_{ij}W_{ijk} - 1)\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\exp(-sI_{ij}W_{ijk}) - s\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} I_{ij}W_{ijk}\exp(-sI_{ij}W_{ijk})}{\left(\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\exp(-sI_{ij}W_{ijk})\right)^2}. \tag{25}$$

The weight updating continues till there is no significant change in weights between two successive update cycles.

To learn rules with N clauses in the antecedent viz *If* x_1 *is* A_1 *and* x_2 *is* A_2 *and*…x_N *is* A_N *then* b *is* B, the network can be easily extended again with just one hidden layer. The hidden layer will compute x_1 *is* A_1 *and* x_2 *is* A_2 *and*…x_N *is* A_N and output layer will compute the inferred conclusion. Let n_i be the number of quantization level of x_i, i.e. the fuzzy set A_i is represented by a vector of size n_i. So the input layer will have $\sum_{i=1}^{N} n_i = N_1$ nodes. Let us denote the input layer nodes by $I_{i,j}$, $i = 1, \ldots, N$; $j = 1, \ldots, n_i$. Nodes $I_{i,j}$, $j = 1, \ldots, n_i$ represent the fuzzy set A_i. The hidden layer will have $N_2 = \prod_{i=1}^{N} n_i$ nodes. Let us denote the hidden layer nodes by $H_{i_1,i_2,\ldots,i_N}$; $i_1 = 1, \ldots, n_1$; $i_2 = 1, \ldots, n_2$; …; $i_N = 1, \ldots, n_N$. Here, the hidden node $H_{i_1,i_2,\ldots,i_N}$ is connected to nodes I_{j,i_j}; $j = 1, \ldots, N$ of the input layer. All connection weights between the input and the hidden layer are set to unity ($= 1$). Each hidden layer node uses a T-norm on the set of inputs received by the node. Thus, the hidden layer outputs correspond to the relational representation of the antecedent clause. The output layer has N_3 nodes, where N_3 is the number of quantization level of the output domain. The hidden and the output layer are completely connected and these connections represent the relational representation of the entire rule. The activation functions of the hidden and the output layers remain the same as that of the previous case.

We have tested our proposed scheme on rules with one clause in the antecedent and also two clauses in the antecedent, and got quite satisfactory results.

3. Evaluation of the network

In order to evaluate the performance of net, we used several indices including even the difference of fuzzy entropy (Pal & Bezdek, 1994). Here, we mention only two indices, Avg (average distance) and Max (maximum distance) as used by Keller et al. (1992a) and Pal et al. (1998).

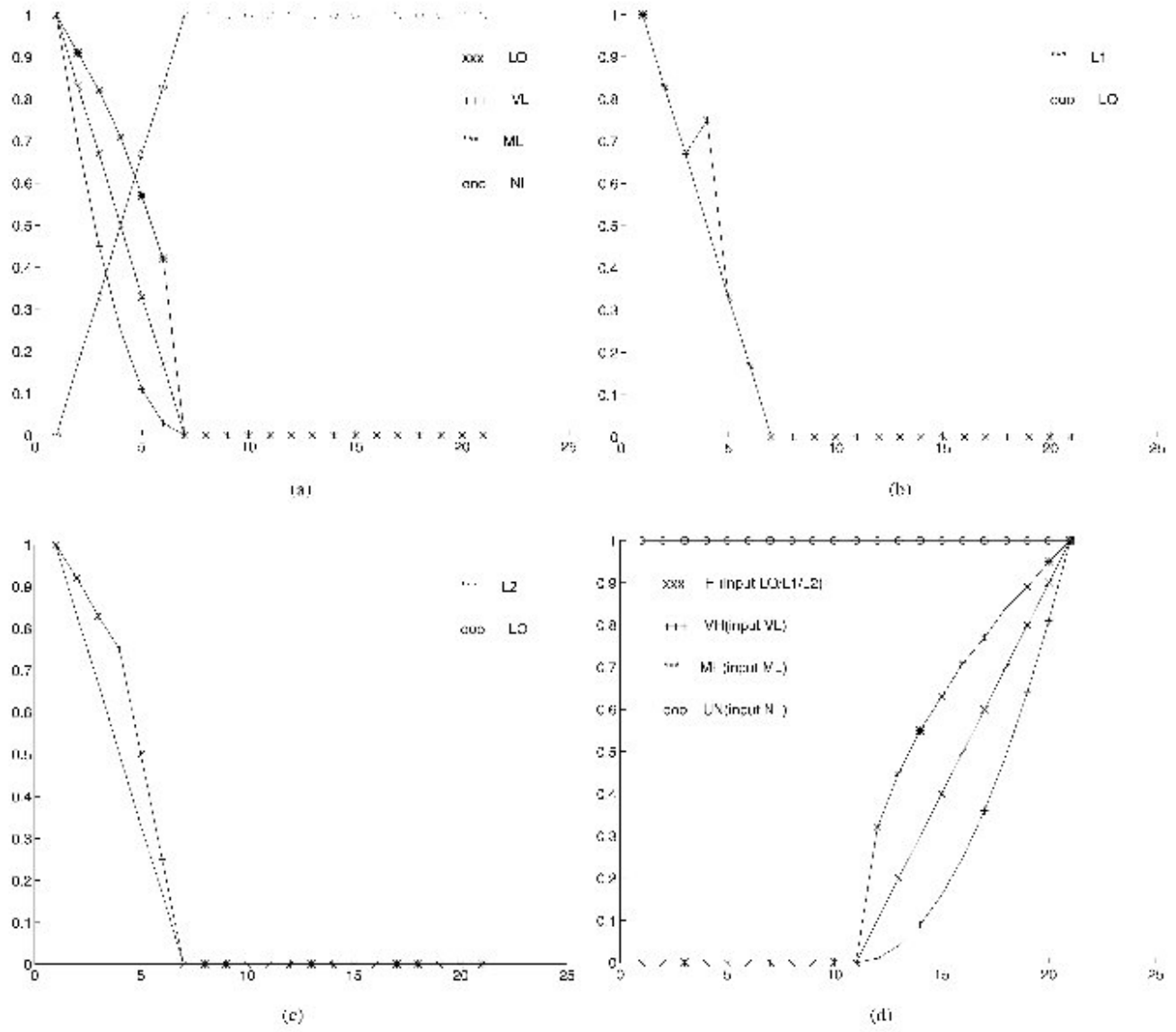

Fig. 3. (a) The different input fuzzy sets. (b) The input fuzzy sets LO and L1. (c) The input fuzzy sets LO and L2. (d) Output fuzzy sets corresponding to the inputs in (a), (b) and (c).

Average distance (Avg): Let $B = \{b_i\}$ be the vector representing the desired (target) output and $B' = \{b_i'\}$ be the output produced by the net.

Then $\text{Avg} = \sum_{i=1}^{m} |b_j - b_j'|/m$, where m is the number of elements in the universe on which B is defined.

Maximum distance (Max): Max is defined as $\text{Max} = \max_j\{|b_j - b_i'|\}$.

Note that Keller et al. computed Max and Avg with respect to B, when the network was set for the relation *If x is A Then y is B*. But in the present investigation, we propose to compute them as distances from the target fuzzy set in the training data. In other words, for the training data (A_i, B_i), we compute Avg and Max using the pair (B_i, B_i') is the conclusion suggested by the net when A_i was the input. The idea is extendible to antecedent with multiple clauses in a straight forward manner.

4. Data for learning

First, we discuss how data can be generated for antecedents with single clause. Tuning K_{ij} (or K_{ijk}) raises an important issue, what would be the training data! Suppose we want to learn *if x is LOW then y is HIGH*. As a first choice what comes to our mind is to use the data corresponding to the pair (*LOW*, *HIGH*). Let X be the vector containing the membership values corresponding to *LOW* and Y be the same for *HIGH*. Now when X is given as an input to the net, suppose the network produces an output vector Y'. We can now learn K_{ij}, so that $\|Y - Y'\|^2$ is minimum.

This is not a good choice as the net may learn the relation *If x is LOW Then y is HIGH* quite well, but the net may be so much biased to this relation that it may fail to realize the

right kind of generalization and the reasoning network might behave like an ordinary multi-layer perceptron, i.e. approximate the input–output mapping (using a inferencing paradigm) loosing its reasoning ability. We generate the training data using the same concept as in Pal and Pal (1999) and Pal et al. (1998).

Our objective is not just to learn the relation *If x is A Then y is B*. The net should learn in such a manner that when the input is A', the net should produce a B', so that the similarity/dissimilarity between A and A' is reflected between B and B'. Moreover, when the input is NOT A, the output should be least specific i.e. the UNKNOWN. Therefore, the training set should contain at least (A, B) and (*NOT A*, *UNKNOWN*). In addition, it can also include pairs like (A', B'). For example, in case of *if x is LOW then y is HIGH*, we can generate training data using the following cases:

If x is VERY LOW Then y is VERY HIGH
If x is MORE OR LESS LOW Then y is MORE OR LESS HIGH
If x is NOT LOW Then y is UNKNOWN

Thus, to make the net learn *If x is LOW Then y is HIGH* we train it using these four sets of training vectors (*LOW*, *HIGH*), (*VERY LOW*, *VERY HIGH*), (*MORE OR LESS LOW*, *MORE OR LESS HIGH*), (*NOT LOW*, *UNKNOWN*). These four are very natural choices for (*LOW*, *HIGH*) relation. Some other distorted cases like (LOW', $HIGH'$) may also be added to the set. One can, of course, argue against the use of (*VERY LOW*, *VERY HIGH*) as this may cause the system make a conclusion which is more specific than *HIGH*. But this is not important in the present case. Our intention is to justify that we can always have some reasonable and consistent training data.

We emphasize here that the above set of four vectors is needed just to learn the relation *If x is LOW Then y is HIGH*. It does not imply that the net is also trained for the relation *If x is VERY LOW Then y is VERY HIGH* or *If x is MORE OR LESS LOW Then y is MORE OR LESS HIGH*. Because to learn the relation (*VERY LOW*, *VERY HIGH*), the training set may contain (*LOW*, *HIGH*), (*VERY LOW*, *VERY HIGH*), (*MORE OR LESS LOW*, *MORE OR LESS HIGH*) and must contain (*NOT VERY LOW*, *UNKNOWN*).

5. Results and discussion

5.1. Results with single clause

Here, we use the same membership functions with the same quantization levels as in Pal et al. (1998), i.e. each fuzzy set is quantized into 21 levels. In order to learn the relation (LOW, HIGH), we used four rules (LOW, HIGH), (VERY LOW, VERY HIGH), (MORE OR LESS LOW, MORE OR LESS HIGH) and (NOT LOW, UNKNOWN) for training. The input membership functions used for training are depicted in Fig. 3(a)–(c) and the corresponding different output membership functions are shown in Fig. 3(d). The membership functions corresponding to Fig. 3(a)–(c) are included in Table 1. Table 2 shows the target outputs (corresponding to Fig. 3(d)) for all input conditions depicted in Table 1. It should be noted that we have generated two distorted versions of LOW (LO), named as L1 and L2, which are shown separately, L1 in Fig. 3(b) and L2, Fig. 3(c). However, the target outputs for all three inputs (i.e. LO, L1 and L2) are the same, HIGH. All outputs are shown in Fig. 3(d).

Table 1
Different input fuzzy sets used for training

Lab	1	3	5	7	9	11	13	15	17	19	21
LO	1.0	0.67	0.33	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L1	1.0	0.67	0.33	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L2	1.0	0.83	0.50	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ML	1.0	0.82	0.57	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
VL	1.0	0.45	0.11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
NL	0.0	0.33	0.67	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
HI	0.0	0.0	0.0	0.0	0.0	0.0	0.20	0.40	0.60	0.80	1.0

Table 2
The target outputs for inputs in Table 1

Lab	1	3	5	7	9	11	13	15	17	19	21
LO	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.40	0.60	0.80	1.0
L1	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.40	0.60	0.80	1.0
L2	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.40	0.60	0.80	1.0
ML	0.00	0.00	0.00	0.00	0.00	0.00	0.45	0.63	0.77	0.89	1.0
VL	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.16	0.36	0.64	1.0
NL	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Note that, all the membership functions discussed earlier are defined using 21 equispaced points on the respective domains of discourse. However, in the tabular representation of the membership functions, we show only 11 points (every alternate points of the actual data used). We emphasize that for all our computation and graphical representation, we use 21 points, but for clarity of presentation in the tables to follow, we show only 11 points.

In order to start the learning process we need to initialize the connection weights w_{ij} or equivalently we need to initialize K_{ij}. We use two schemes named Scheme I and Scheme II.

Scheme I: We initialize K_{ij} using the training data. Let R_i, $i = 1, \ldots, N$ be N rules or relations that are to be learned. Then define

$$R = [r_{ij}]_{n\times m} = \bigcup_{i=1}^{N} R_i. \tag{26}$$

Since $w_{ij} = \exp(-K_{ij})^2 = r_{ij}$, we set

$$K_{ij} = \sqrt{\log \frac{1}{r_{ij}}} \qquad \forall i, j. \tag{27}$$

Table 3
The output fuzzy sets obtained by the soft–max COIN with $s = -10$

Lab	1	3	5	7	9	11	13	15	17	19	21
Before tuning											
LO	0.15	0.15	0.15	0.15	0.15	0.15	0.34	0.57	0.73	0.86	0.97
L1	0.25	0.25	0.25	0.25	0.25	0.25	0.36	0.56	0.72	0.85	0.97
L2	0.28	0.28	0.28	0.28	0.28	0.28	0.36	0.56	0.71	0.84	0.96
ML	0.31	0.31	0.31	0.31	0.31	0.31	0.37	0.55	0.71	0.84	0.96
VL	0.05	0.05	0.05	0.05	0.05	0.05	0.35	0.58	0.74	0.87	0.99
NL	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
After tuning											
LO	0.00	0.00	0.00	0.00	0.00	0.00	0.16	0.36	0.56	0.78	1.0
L1	0.00	0.00	0.00	0.00	0.00	0.00	0.18	0.39	0.56	0.78	1.0
L2	0.01	0.01	0.01	0.01	0.01	0.01	0.29	0.51	0.71	0.86	1.0
ML	0.01	0.01	0.01	0.01	0.01	0.01	0.38	0.53	0.70	0.86	1.0
VL	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.16	0.39	0.64	1.0
NL	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Thus, in Scheme I, COIN with the initial rule matrix will simulate the usual COI but with soft–max. We can expect this as a good initialization of the net. In all simulation results reported we used product to implement the implication operator and max to realize the union operator, in Eq. (26).

Scheme II: In this scheme, we randomly initialize r_{ij} and hence K_{ij}. We use the random number generator to obtain uniformly distributed values over [0,1] as the initial r_{ij} and from there we generate K_{ij} using Eq. (27). Obviously, such an initialization will produce, before training, results quite far from the desired one.

With a view to learn the relation *If x is LOW Then y is HIGH*, we used the following four rules to obtain the initial R: (LOW, HIGH), (VERY LOW, VERY HIGH), (MORE OR LESS LOW, MORE OR LESS HIGH) and (NOT LOW, UNKNOWN).

The output fuzzy sets (conclusions) computed by the net before tuning are shown in Table 3. Here, we find that the output produced by the net is not very good. Table 3 depicts the output fuzzy sets obtained after tuning of K for 30 000 iterations. Comparison of these tables with the targeted fuzzy sets (see Table 2) shows that after tuning of K we get a very good match between the targeted output and calculated output. In this case, we used $s = -10$, and we find that the total square error computed by Eq. (14) came down from 4.33 (before tuning) to 0.13 (after tuning).

Each of the six Fig. 4(a)–(f) displays desired output and the output produced by the COIN and max–prod COI for one of the six cases (Table 4). In each case, we find that conventional COI is the worst and COIN produces the best result making an excellent match with the desired one.

These points are further highlighted in Table 5, which summarizes the performance of COIN. Table 5 depicts that both Max and Avg are the least for COIN after tuning and in fact these two indices exhibit very low values suggesting very good match between the desired and the computed conclusions. Obviously, before tuning the indices are quite high. For the present initialization (without tuning), COIN with soft–max approximates the usual COI.

For the sake of comparison, we have also implemented the usual max–min COI. Note that this is not realized by COIN. The results are included in Table 4. They are not quite satisfactory (columns 8 and 9 of Table 5). In fact, max–min appears worse compared to max–prod.

In Pal and Pal (1999), COIN with single antecedent clause has been extensively tested for different s and also compared with nets of Pal et al. (1998). Performance of COIN is found to be quite consistent and satisfactory. Moreover, COIN was found to outperform the net by Pal et al. (1998).

So far, using COIN, we found a very good performance of the proposed net when the initial R was calculated using Eq. (26). To see its robustness with respect to initialization, we initialized R using Scheme II. We have experimented with several such random R and obtained good results. We report here only one typical result. Table 6 shows the initialization used. Since Scheme II uses random initialization, in Table 6, we show the complete relation used, so that readers can replicate the results (Table 6 shows the actual values rounded to two digits). The outputs obtained by COIN before tuning of K are shown in Table 7, and we find that the outputs are, as expected, pretty bad. Table 7 depicts the outputs obtained after tuning of K using soft–max COIN for 30 000 iterations. Comparing Table 2 with Table 7, we find a very good match between desired and the computed outputs although we started with a very bad initial R.

Table 8 presents the different performance indices before and after tuning of K for the initialization in Table 6.

Comparing Table 8 with Table 5, we find that even with a very bad random initialization, COIN can find a good realization of the relation R so that the computed outputs agree very nicely with the desired ones.

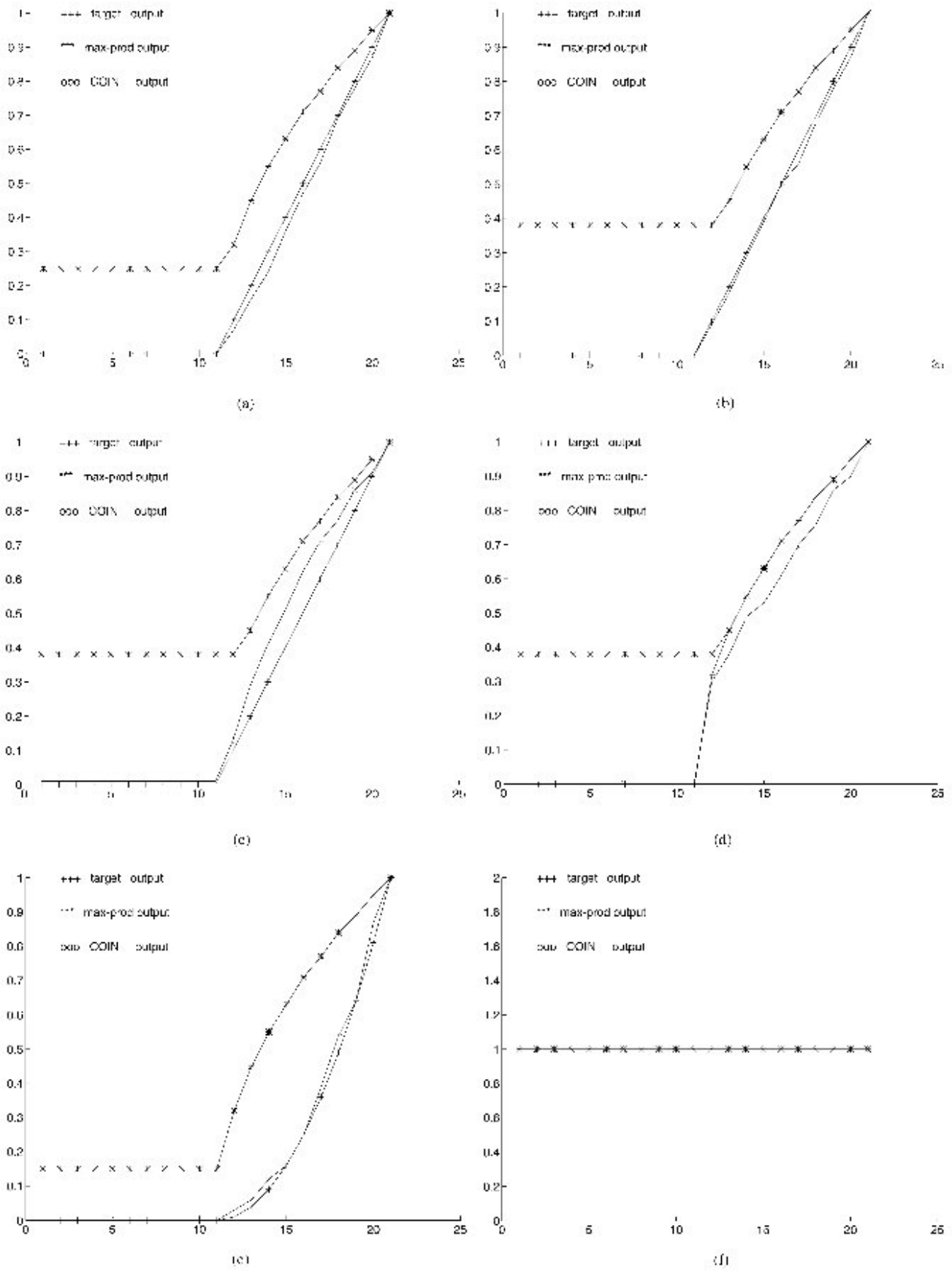

Fig. 4. (a) Comparison of outputs obtained by different methods for input set LO. (b) Comparison of outputs obtained by different methods for input set L1. (c) Comparison of outputs obtained by different methods for input set L2. (d) Comparison of outputs obtained by different methods for input set ML. (e) Comparison of outputs obtained by different methods for input set VL. (f) Comparison of outputs obtained by different methods for input set NL.

Table 4
Output fuzzy sets obtained using the usual (a) max–prod COI, and (b) max–min COI

Lab	1	3	5	7	9	11	13	15	17	19	21
Max–prod COI											
LO	0.25	0.25	0.25	0.25	0.25	0.25	0.45	0.63	0.77	0.89	1.0
L1	0.38	0.38	0.38	0.38	0.38	0.38	0.45	0.63	0.77	0.89	1.0
L2	0.38	0.38	0.38	0.38	0.38	0.38	0.45	0.63	0.77	0.89	1.0
ML	0.38	0.38	0.38	0.38	0.38	0.38	0.45	0.63	0.77	0.89	1.0
VL	0.15	0.15	0.15	0.15	0.15	0.15	0.45	0.63	0.77	0.89	1.0
NL	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Max–min COI											
LO	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.63	0.77	0.89	1.0
L1	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.63	0.77	0.89	1.0
L2	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.63	0.77	0.89	1.0
ML	0.57	0.57	0.57	0.57	0.57	0.57	0.57	0.63	0.77	0.89	1.0
VL	0.33	0.33	0.33	0.33	0.33	0.33	0.45	0.63	0.77	0.89	1.0
NL	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

5.2. Results with multiple clauses

Now we consider rules with antecedent having two clauses. To test our proposed scheme, we have used four rules.

R_1 : *If x is LOW and y is LOW Then z is LOW*
R_2 : *If x is LOW and y is HIGH Then z is LOW*
R_3 : *If x is HIGH and y is LOW Then z is LOW*
R_4 : *If x is HIGH and y is HIGH Then z is HIGH*

Let us denote the two antecedent clauses of the rule R_i by C_{i_1} and C_{i_2}. For example, for R_1, $C_{1_1} = x$ *is LOW* and $C_{1_2} = y$ *is LOW*. By $\overline{C_{i_j}}$, we denote the complement of C_{i_j}, e.g. $\overline{C_{1_1}} = x$ *is NOT LOW*.

Now for each rule R_i, let us define a set R_i' containing four rules $\{R'_{i_1}, R'_{i_2}, R'_{i_3}, R'_{i_4}\}$ with R'_{i_1} having antecedent A'_{i_j}, where

$$A'_{i_1} = C_{i_1} \text{ AND } C_{i_2}$$

Table 5
Performance evaluation of COIN

	COIN before K-tuning		COIN after K-tuning		COI using max–prod		COI using max–min	
Lab	Avg	Max	Avg	Max	Avg	Max	Avg	Max
LO	0.13	0.17	0.02	0.06	0.21	0.25	0.35	0.50
L1	0.18	0.25	0.01	0.04	0.28	0.38	0.35	0.50
L2	0.2	0.28	0.04	0.12	0.28	0.38	0.35	0.50
ML	0.19	0.31	0.03	0.1	0.20	0.38	0.32	0.57
VL	0.16	0.43	0.01	0.06	0.23	0.47	0.33	0.47
NL	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 6
An initial R generated by Scheme II

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0.32	0.12	0.34	0.37	0.49	0.06	0.11	0.03	0.45	0.42	0.30	0.20	0.09	0.00	0.33	0.11	0.18	0.07	0.31	0.42	0.43
0.19	0.45	0.17	0.13	0.37	0.26	0.09	0.20	0.09	0.30	0.22	0.18	0.00	0.49	0.32	0.21	0.37	0.15	0.25	0.34	0.48
0.34	0.24	0.36	0.15	0.13	0.17	0.38	0.11	0.43	0.42	0.46	0.27	0.41	0.26	0.36	0.14	0.27	0.40	0.41	0.14	0.26
0.22	0.40	0.30	0.24	0.25	0.11	0.18	0.36	0.34	0.29	0.07	0.30	0.28	0.25	0.26	0.48	0.14	0.41	0.39	0.05	0.17
0.37	0.25	0.45	0.08	0.48	0.50	0.13	0.31	0.27	0.14	0.43	0.43	0.06	0.30	0.39	0.08	0.18	0.15	0.48	0.09	0.46
0.34	0.38	0.29	0.29	0.02	0.00	0.04	0.38	0.25	0.19	0.10	0.09	0.24	0.44	0.30	0.40	0.39	0.02	0.08	0.48	0.46
0.30	0.01	0.29	0.43	0.18	0.26	0.26	0.32	0.50	0.06	0.06	0.04	0.28	0.30	0.33	0.01	0.04	0.20	0.46	0.18	0.33
0.17	0.35	0.24	0.44	0.36	0.17	0.41	0.10	0.48	0.06	0.39	0.04	0.42	0.32	0.23	0.30	0.24	0.00	0.14	0.09	0.05
0.41	0.09	0.09	0.32	0.05	0.20	0.13	0.18	0.43	0.33	0.38	0.35	0.23	0.12	0.24	0.47	0.36	0.29	0.45	0.29	0.33
0.10	0.06	0.35	0.46	0.43	0.41	0.01	0.40	0.44	0.13	0.17	0.40	0.04	0.40	0.40	0.46	0.48	0.01	0.33	0.13	0.45
0.22	0.10	0.13	0.17	0.32	0.09	0.16	0.39	0.14	0.31	0.48	0.16	0.45	0.28	0.10	0.37	0.10	0.04	0.26	0.44	0.19
0.25	0.49	0.09	0.24	0.09	0.38	0.09	0.13	0.26	0.48	0.48	0.19	0.45	0.27	0.06	0.35	0.46	0.33	0.25	0.16	0.35
0.50	0.01	0.49	0.45	0.19	0.03	0.44	0.35	0.23	0.25	0.07	0.43	0.22	0.06	0.09	0.47	0.48	0.18	0.13	0.33	0.47
0.45	0.05	0.39	0.22	0.33	0.22	0.07	0.19	0.49	0.47	0.25	0.21	0.01	0.19	0.36	0.27	0.03	0.42	0.46	0.28	0.38
0.24	0.08	0.21	0.44	0.38	0.12	0.25	0.29	0.36	0.18	0.20	0.09	0.06	0.46	0.35	0.45	0.03	0.20	0.15	0.43	0.50
0.35	0.28	0.37	0.26	0.29	0.46	0.15	0.03	0.41	0.20	0.23	0.41	0.35	0.17	0.33	0.01	0.16	0.01	0.35	0.38	0.22
0.31	0.20	0.39	0.20	0.38	0.25	0.06	0.37	0.20	0.50	0.09	0.01	0.17	0.23	0.08	0.26	0.46	0.30	0.15	0.25	0.13
0.06	0.46	0.16	0.37	0.50	0.30	0.19	0.40	0.17	0.45	0.11	0.04	0.29	0.19	0.18	0.35	0.45	0.31	0.39	0.42	0.27
0.06	0.10	0.21	0.11	0.29	0.08	0.03	0.01	0.18	0.04	0.02	0.07	0.49	0.18	0.16	0.33	0.06	0.36	0.40	0.48	0.02
0.29	0.20	0.46	0.05	0.47	0.14	0.19	0.47	0.31	0.43	0.45	0.20	0.04	0.03	0.40	0.17	0.32	0.04	0.49	0.42	0.38
0.15	0.32	0.30	0.37	0.49	0.36	0.03	0.32	0.32	0.36	0.26	0.45	0.19	0.48	0.02	0.16	0.43	0.46	0.46	0.47	0.47

Table 7
Output fuzzy sets obtained by (a) soft–max COIN ($s = -10$) and R in Table 6, (b) soft–max COIN ($s = -10$) after tuning K for 30 000 iterations

Lab	1	3	5	7	9	11	13	15	17	19	21
Soft–max COIN ($s = -10$) and R											
LO	0.19	0.21	0.41	0.11	0.35	0.21	0.13	0.22	0.17	0.21	0.35
L1	0.19	0.22	0.40	0.11	0.34	0.21	0.14	0.22	0.17	0.23	0.35
L2	0.21	0.24	0.40	0.17	0.35	0.27	0.21	0.25	0.21	0.25	0.37
ML	0.21	0.24	0.40	0.17	0.35	0.27	0.20	0.24	0.21	0.25	0.36
VL	0.18	0.21	0.41	0.05	0.35	0.17	0.05	0.20	0.12	0.18	0.33
NL	0.39	0.39	0.42	0.34	0.43	0.41	0.41	0.33	0.43	0.42	0.42
Soft–max COIN ($s = -10$) after tuning K for 30 000 iterations											
LO	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.38	0.59	0.79	1.0
L1	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.40	0.58	0.79	1.0
L2	0.01	0.01	0.00	0.00	0.01	0.01	0.32	0.53	0.66	0.79	1.0
ML	0.01	0.01	0.01	0.01	0.01	0.01	0.31	0.52	0.66	0.79	1.0
VL	0.00	0.00	0.00	0.00	0.00	0.00	0.08	0.17	0.45	0.79	1.0
NL	0.99	0.99	0.99	1.0	0.99	0.99	0.99	0.99	0.99	0.99	0.99

$$A'_{i_2} = \overline{C_{i_1}} \text{ AND } C_{i_2}$$
$$A'_{i_3} = \overline{C_{i_1}} \text{ AND } \overline{C_{i_2}}$$
$$A'_{i_4} = \overline{C_{i_1}} \text{ AND } \overline{C_{i_2}}$$

To learn R_1, a reasonable set of training data would be

$$R'_{1_1} = \textit{if } A'_{1_1} \textit{ then } z \textit{ is LOW}$$
$$R'_{1_2} = \textit{if } A'_{1_2} \textit{ then } z \textit{ is UNKNOWN}$$
$$R'_{1_3} = \textit{if } A'_{1_3} \textit{ then } z \textit{ is UNKNOWN}$$
$$R'_{1_4} = \textit{if } A'_{1_4} \textit{ then } z \textit{ is UNKNOWN}$$

We test the net in two different ways: first, we try to learn a single rule R_i, then we try to learn all four rules R_1, R_2, R_3 and R_4 in a single net.

Again to learn a single rule, we use two different strategies. First, we trained the net with only one rule R_i and then test the output with the four rules in the set R'_i. If the net has learnt the rule R_i 'properly' then outputs for the rules R'_{i_2}, R'_{i_3}, R'_{i_4} will be close to *UNKNOWN*. This case will demonstrate the generalizing ability of our net. In the second case, while learning the rule R_i, we used four rules of the set R'_i as training data and tested the outputs. This can demonstrate the learning capability of our net.

The results we have got for R_1 are listed in Table 9. Here, we find that the results obtained for the second case are much better than those for the first case. This is due to the fact that in the first case we did not use rules R'_{1_2}, R'_{1_3}, R'_{1_4} for learning.

Table 8
Various performance indices when initialization is done using Scheme II

	Before K-tuning		After K-tuning	
Input	Avg	Max	Avg	Max
LO	0.28	0.65	0.01	0.05
L1	0.28	0.65	0.01	0.05
L2	0.28	0.63	0.04	0.13
ML	0.35	0.64	0.05	0.18
VL	0.23	0.67	0.04	0.15
NL	0.61	0.67	0.01	0.01

To initialize the weights between the hidden and the output layer, we used two different schemes, Scheme I and Scheme II as described earlier. Results for both are reported in Table 9. For the other rules, i.e. R_2 to R_4 also, we did the same experiment and got similar results.

Next we tried to learn all four rules by a single net. In this case, our training data set contains four rules R_1, R_2, R_3 and R_4. The results for both kinds of initializations are reported in Table 9.

A natural question comes: how does ordinary max–prod composition performs in this case? To check that, we implemented max–prod COI. Table 10 shows the outputs corresponding to Table 9, respectively, using the max–prod COI. Table 10 shows the results of max–prod COI for training only one rule (here R_1), in two different ways, and Table 10 shows the results obtained while learning all four rules. Comparing Table 9 and Table 10, we find that our COIN demonstrates a significant improvement over the max–prod COI. This is also reflected in Table 11, which summarize performances corresponding to different runs by different methods. Table 11 contains the performance indices of different runs for learning single rule and for all four rules.

Table 11 reveals that when a single rule is used for training, the agreement between the computed and the desired output for that rule is quite good for both COI and COIN, but for the other three cases COIN performs similar to or better than the usual COI. When training is done with all the four rules, as expected COIN performs much better than usual COI, for both schemes.

Similarly, for Table 11 we find that when we try to learn all four rules, COIN with Scheme I initialization outperforms max–prod COI and COIN with random (Scheme II) initialization.

Table 9
Output fuzzy sets produced by COIN when we try to learn (a) the rule R_1 in two different ways, and (b) all four rules R_1–R_4 in a single net

Lab	1	3	5	7	9	11	13	15	17	19	21
Rule R_1 in two different ways											
Learning is done using only R_1 (Scheme I initialization)											
$z(R'_{1_1})$	0.99	0.67	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$z(R'_{1_2})$	0.03	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$z(R'_{1_3})$	0.03	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$z(R'_{1_4})$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Learning is done using only R_1 (Scheme II initialization)											
$z(R'_{1_1})$	0.80	0.67	0.33	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
$z(R'_{1_2})$	0.89	0.86	0.83	0.84	0.90	0.90	0.75	0.76	0.88	0.80	0.68
$z(R'_{1_3})$	0.78	0.80	0.88	0.91	0.81	0.87	0.92	0.86	0.82	0.92	0.90
$z(R'_{1_4})$	0.90	0.90	0.92	0.91	0.90	0.89	0.90	0.91	0.90	0.91	0.88
Learning is done with all four rules belonging to set R'_1 (Scheme I initialization)											
$z(R'_{1_1})$	0.99	0.67	0.33	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
$z(R'_{1_2})$	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
$z(R'_{1_3})$	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
$z(R'_{1_4})$	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Learning is done with all four rules belonging to set R'_1 (Scheme II initialization)											
$z(R'_{1_1})$	0.80	0.67	0.33	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
$z(R'_{1_2})$	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
$z(R'_{1_3})$	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
$z(R'_{1_4})$	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98
All four rules R_1–R_4 in a single net											
With Scheme I initialization											
$z(R_1)$	0.99	0.67	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$z(R_2)$	0.99	0.67	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$z(R_3)$	0.99	0.67	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$z(R_4)$	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.40	0.60	0.80	0.98
With Scheme II initialization											
$z(R_1)$	0.80	0.67	0.33	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
$z(R_2)$	0.80	0.67	0.33	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
$z(R_3)$	0.62	0.67	0.33	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
$z(R_4)$	0.02	0.02	0.02	0.02	0.02	0.02	0.20	0.40	0.60	0.77	0.99

Table 10
Performance of max–prod COI while learning (a) rule R_1 in two different ways and (b) all four rules R_1–R_4

Lab	1	3	5	7	9	11	13	15	17	19	21
Rule R_1 in two different ways											
Results when learning is done with R_1 only											
$z(R'_{1_1})$	0.98	0.53	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$z(R'_{1_2})$	0.03	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$z(R'_{1_3})$	0.03	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$z(R'_{1_4})$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Results when learning is done with all rules belonging to set R'_1											
$z(R'_{1_1})$	0.98	0.53	0.08	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
$z(R'_{1_2})$	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
$z(R'_{1_3})$	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
$z(R'_{1_4})$	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
All four rules R_1–R_4											
Results when learning is done with all four rules											
$z(R_1)$	0.98	0.53	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$z(R_2)$	0.96	0.53	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$z(R_3)$	0.96	0.53	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$z(R_4)$	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.13	0.41	0.70	0.94

Table 11
Performance comparison of various methods when we learn (a) only a single rule R_1 (b) all four rules R_1–R_4 in a single net

Data	One rule training						Four rule training					
	Max–prod COI		COIN with Scheme I initialization		COIN with Scheme II initialization		Max–prod COI		COIN with Scheme I initialization		COIN with Scheme II initialization	
	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max
Only a single rule R_1												
R'_{1_1}	0.04	0.28	0.0	0.01	0.02	0.20	0.08	0.28	0.01	0.01	0.02	0.20
R'_{1_2}	0.99	1.0	0.99	1.0	0.18	0.32	0.10	0.10	0.01	0.01	0.01	0.01
R'_{1_3}	0.99	1.0	0.99	1.0	0.15	0.28	0.10	0.10	0.01	0.01	0.01	0.01
R'_{1_4}	1.0	1.0	1.0	1.0	0.10	0.12	0.0	0.0	0.0	0.0	0.02	0.02

	Max–prod COI		COIN with Scheme I initialization		COIN with Scheme II initialization	
	Avg	Max	Avg	Max	Avg	Max
All four rules R_1–R_4 in a single net						
R_1	0.04	0.28	0.0	0.01	0.02	0.20
R_2	0.04	0.27	0.0	0.01	0.02	0.20
R_3	0.04	0.27	0.0	0.01	0.03	0.38
R_4	0.08	0.27	0.0	0.02	0.01	0.04

6. Conclusions

Keller et al. suggested a NN for fuzzy reasoning and later Pal et al. modified that. These networks implemented a kind of similarity-based reasoning. Pal and Pal then proposed a network for COI with single clause in the antecedent. In this paper, we generalized the net so that it can deal with rules with antecedents having multiple clauses. Given a set of rules, the proposed architecture can realize the COI. It is well known that the outcome of COI depends on the choice of the operator used for implication and also on the inferencing scheme used. There are infinitely many choices for the implication operator. COIN avoids the problem of choosing an appropriate implication function through neural learning. The system automatically finds an optimal relation to represent a set of fuzzy rules. We suggested a suitable modeling of connection weights which forced the learned weights to lie in [0,1]. Two different schemes of initialization of the weights are proposed. Effectiveness of COIN has been demonstrated through extensive numerical examples. We observed that the proposed neural realization can find a much better representation of the rules than that by usual compositional rules of inferencing and hence results in a much better conclusion than the usual COI. COIN is also quite robust with respect to initialization of its weights.

Acknowledgements

We are thankful to the referees for their valuable comments, which have helped us in making this paper more complete and readable. N.R. Pal gratefully acknowledges partial support provided by the Department of Science and Technology, Govt. of India, grant No. III.5(2)/96-ET, to complete this investigation.

References

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

Hayashi, Y., Buckley, J. J., & Czogala, E. (1992). Direct fuzzification of neural network and fuzzified delta rule. *Proceedings of the Second International Conference on Fuzzy Logic and Neural Networks*, IIZUKA, Japan (pp. 73–76).

Haykin, S. (1994). *Neural networks—a comprehensive foundation*, New York: Macmillan.

Ishibuchi, H., Fujioka, R., & Tanaka, H. (1993). Neural networks that learn from fuzzy if–then rules. *IEEE Transactions on Fuzzy Systems*, *1*, 85–97.

Keller, J. (1990). Experiments in neural network architectures for fuzzy logic. *Proceedings of the Second Joint Technical Workshop on Neural Networks and Fuzzy Logic*, Johnson Space Center, Houston (pp. 201–216).

Keller, J. (1993). The inference process in fuzzy logic through artificial neural network structures. *Proceedings of the First European Congress on Fuzzy and Intelligent Technologies*, Aachen, Germany (pp. 195–201).

Keller, J., & Tahani, H. (1992a). Backpropagation neural networks for fuzzy logic. *Information Sciences*, *45*, 1–12.

Keller, J., & Tahani, H. (1992b). Implementation of conjunctive and disjunctive fuzzy logic rules with neural networks. *International Journal of Approximate Reasoning, Special Issue on: Fuzzy Logic and Neural Networks for Control*, *6*, 221–240.

Keller, J., & Yager, R. (1989). Fuzzy logic inference neural networks. *Proceedings on SPIE Symposium on Intelligence Robots and Computer Vision VIII*, Philadelphia (pp. 582–591).

Keller, J., Krishnapuram, R., & Rhee, F. (1992a). Evidence aggregation

networks for fuzzy logic inference. *IEEE Transactions on Neural Networks*, *3*, 761–769.

Keller, J. M., Yager, R. R., & Tahani, H. (1992b). Neural network implementation of fuzzy logic. *Fuzzy Sets and Systems*, *45*, 1–12.

Keller, J., Hayashi, Y., & Chen, Z. (1993). Interpretation of nodes in networks for fuzzy logic. *Proceedings of the Second IEEE International Congress on Fuzzy Systems*, San Francisco, CA (pp. 1203–1207).

Klir, G. J., & Folger, T. A. (1988). *Fuzzy sets, uncertainty, and information*, Englewood Cliffs, NJ: Prentice-Hall.

Lee, C. C. (1990). Fuzzy logic in control systems: Fuzzy logic controller—part I and II. *IEEE Transactions on Systems Man and Cybernetics*, *SMC-20*, 404–435.

Mamdani, E. H. (1977). Application of fuzzy logic to approximate reasoning using linguistic synthesis. *IEEE Transactions on Computers*, *26*, 1182–1191.

Mizumoto, M. (1985). Extended fuzzy reasoning. In M. M. Gupta, A. Kandel, W. Bandler & J. B. Kiszka, *Approximate reasoning in expert systems* (pp. 71–85). North-Holland: Amsterdam.

Mizumoto, M., Fukami, S., & Tanaka, K. (1979a). Fuzzy conditional inference and fuzzy inference with fuzzy quantifiers. *Proceedings of the Sixth International Conference on Artificial Intelligence*, Tokyo (pp. 589–591).

Mizumoto, M., Fukami, S., & Tanaka, K. (1979b). Several methods of fuzzy conditional inference. *Proceedings of IEEE Conference on Decision and Control*, Florida (pp. 777–782).

Mizumoto, M., Fukami, S., & Tanaka, K. (1979c). Some methods of fuzzy reasoning. In M. M. Gupta, R. K. Regade & R. R. Yager, *Advances in fuzzy set theory and applications* (pp. 117–136). North-Holland: Amsterdam.

Mizumoto, M., Fukami, S., & Tanaka, K. (1980). Some considerations on fuzzy conditional inferences. *Fuzzy Sets and Systems*, *4*, 243–273.

Mizumoto, M., & Zimmerman, H. J. (1982). Comparison of fuzzy reasoning methods. *Fuzzy Sets and Systems*, *8*, 253–284.

Nie, J., & Linkens, D. (1992). Fuzzy reasoning implemented by neural networks. *Proceedings on International Joint Conference on Neural Networks*, Vol. II (pp. 702–706).

Nie, J., & Linkens, D. (1995). *Fuzzy-neural control*, Englewood Cliffs, NY: Prentice-Hall.

Pal, N. R., & Bezdek, J. C. (1994). Measuring fuzzy uncertainty. *IEEE Transactions on Fuzzy Systems*, *2*, 107–118.

Pal, S. K., & Pal, N. R. (1996). Soft computing: Goal, tools and feasibility. *Journal of IETE, Special Issue on Neural Networks*, *42*, 195–204.

Pal, K., & Pal, N. R. (1999). A neuro-fuzzy system for inferencing. *International Journal of Intelligent Systems*, *14*, 1155–1182.

Pal, K., Pal, N. R., & Keller, J. M. (1998). Some neural net realizations of fuzzy reasoning. *International Journal of Intelligent Systems*, *13*, 859–886.

Scharf, E. M., & Mandic, N. J. (1985). The application of a fuzzy controller to the control of a multi-degree-freedom robot arm. In M. Sugeno, *Industrial applications of fuzzy control*North-Holland: Amsterdam.

Self, K. L. (1990). Fuzzy logic design. *IEEE Spectrum*, *27*, 42–44.

Shann, J. J., & Fu, H. C. (1995). A fuzzy neural network for rule acquiring on fuzzy control systems. *Fuzzy Sets and Systems*, *71*, 345–357.

Sugeno, M. (1985). *Industrial applications of fuzzy control*, North-Holland: Amsterdam.

Suh, I. H., & Kim, T. W. (1994). Fuzzy membership function based neural networks with applications to the visual servoing of robot manipulators. *IEEE Transactions on Fuzzy Systems*, *2*, 203–220.

Takagi, H., & Hayashi, I. (1991). NN-driven fuzzy reasoning. *International Journal of Approximate Reasoning*, *5*, 191–212.

Zadeh, L. A. (1965). Fuzzy sets. *Information Control*, *8*, 338–353.

Zadeh, L. A. (1975). Calculus of fuzzy restrictions. In L. A. Zadeh, K. S. Fu, K. Tanaka & M. Shimura, *Fuzzy sets and their applications to cognitive and decision processes* (pp. 1–39). New York: Academic Press.

Zadeh, L. A. (1988). Fuzzy logic. *IEEE Computer*, *21*, 83–93.