Neural Comput & Applic (2008) 17:297–309
DOI 10.1007/s00521-007-0115-9

ORIGINAL ARTICLE

# Designing a decompositional rule extraction algorithm for neural networks with bound decomposition tree

**Jia Sheng Heh · Jen Cheng Chen · Maiga Chang**

Received: 13 September 2006 / Accepted: 30 March 2007 / Published online: 2 June 2007

**Abstract** The neural networks are successfully applied to many applications in different domains. However, due to the results made by the neural networks are difficult to explain the decision process of neural networks is supposed as a black box. The explanation of reasoning is important to some applications such like credit approval application and medical diagnosing software. Therefore, the rule extraction algorithm is becoming more and more important in explaining the extracted rules from the neural networks. In this paper, a decompositional algorithm is analyzed and designed to extract rules from neural networks. The algorithm is simple but efficient; can reduce the extracted rules but improve the efficiency of the algorithm at the same time. Moreover, the algorithm is compared to the other two algorithms, M-of-N and Garcez, by solving the MONK's problem.

**Keywords** Neural network · Boolean rule · Rule extraction

J. S. Heh
Department of Information and Computer Engineering,
Chung Yuan Christian University,
32099 Chung Li, Taiwan

J. C. Chen (✉)
Department of Electronic Engineering,
Chung Yuan Christian University,
32099 Chung Li, Taiwan
e-mail: clement1972@gmail.com

M. Chang
National Science and Technology Program for e-Learning,
P.O. Box 9-187, 32099 Chung-Li, Taiwan

## 1 Introduction

### 1.1 Motivation

Although both expert systems and neural networks are typical systems in the domain of artificial intelligence, the basic components of these two kinds of systems are different. The knowledge base of expert systems is a set of rules which are stored in symbolic form, while neural networks encode learned knowledge within an established structure with adjustable weights in numerical form. Hence, it is difficult to transfer the training results of a neural network to the knowledge base of an expert system. The solutions represented by the knowledge base of an expert system could be explained and understood by users [11]. The explainable feature is the main advantage of expert system.

In contrast, neural networks have excellent abilities for classifying data and learning from inputs [7], but it is difficult to describe the decision process of a neural network or to merge more than one trained neural network [1, 23]. The key for neural networks to overcome this deficiency is *rule extraction*. Rule extraction is the process of discovering the knowledge which is encoded within a neural network and representing these knowledge pieces in symbolic form, just as a rule base in an expert system. Once the rules which could be used to represent the neural network are extracted, the expert system then can use those rules as its knowledge base and gain the advantages of both Neural Networks and Expert Systems [4, 24].

### 1.2 Related works

Although the neural-network-based systems performed well, it still can not make users feel comfortable due to the 'black box' effects of the neural networks. Therefore, some rule extraction methods for neural networks had been presented and classified into two categories, the pedagogical and the decompositional algorithms [1, 12]. The pedagogical rule extraction algorithm aims to extract rules that map inputs directly into outputs [12]. Some typical algorithms of this category are ''VIA'' [18], ''rule-extraction-as-learning'' [10] and ''RULENEG'' [3]. Most of pedagogical algorithms are not effective when the size of the neural network increases, such as any real world problem. In order to improve the efficiency, decompositional algorithms focus on heuristically searching and extracting rules in neurons of neural networks individually. The existing decompositional algorithms include ''KT'' [8], ''Subset'' [4, 5], ''MofN'' [4, 5], ''RULEX'' [13, 14], ''Setiono'' [15, 16], ''Tsukimoto'' [6], and ''Garcez''[2].

The above rule extraction algorithms are designed for multilayer perceptions (MLP) neural network. There are some other algorithms can extraction rules from difference types of neural networks. In the radial basis function (RBF) neural network, Roger and Sun [29] provided a basic ideal that a RBF network structure can be one-to-one correspondence to fuzzy rules (Takagi-Sugeno fuzzy system [28]). With the similar concept, Leng et al. [30] designed a hybrid neural network which called the self-organising fuzzy neural network (SOFNN). The SOFNN has the capability to encode fuzzy rules in the resulting network. Thus it can extract Takagi-Sugeno fuzzy rules directly from the SOFNN. Beside SOFNN, Castellano et al. [33] had also demonstrated a neuro-fuzzy network which can generate TS fuzzy rules with the similar ideal. McGarry et al. [31] proved a algorithm, is called RULEX (for Local Rule EXtraction), which can extract rules from RBF with a diffidence way. RULEX is tackles these issues by extracting rules at two levels: hREX extracts rules by examining the hidden unit to class assignments while mREX extracts rules based on the input space to output space mappings. Beside RBF, McGarry also offered a technique which can extract propositional IF..THEN type rules from the Kohonen self-organizing feature map (SOM) with Malone et al. [34].

Ultsch et al. [32] presented a system is called CONnectionist Knowledge Acquisition Too (CONKAT). In CONKAT, it uses a Kohonen network and a competitive learning network for learning and its rule extractor uses a modified ID3 algorithm builds a minimal decision tree [21].

### 1.3 Objective

In those rule extraction algorithms for MLP, some decompositional algorithms use weight pruning for more effectively processing [4, 9, 15, 16], however, this kind of algorithms does not guarantee that the pruned network would be equivalent to the original one [2]. Therefore, the accuracy of extracted rules might be decreased. Most of the decompositional algorithms are requested additional times to retrain the pruned network with original training data. This paper tries to develop a decompositional algorithm which could extract rules directly from trained neural network without retraining and keep high accuracy and low complexity at mean time. Such an algorithm is designed by using the concept of bound decomposition of neural network's weights. The algorithm, Bound Decomposition Tree algorithm, offers an efficient method to extract rule from a neuron. Also, the algorithm had successful been applied into some real applications, such as the expert system of Chinese medicine [20], the advance process control expert system [22, 25, 26] and the web management [27].

### 1.4 Outline of this paper

In Sect. 2, inputs of single neuron are modeled as a cube, and relations between these inputs and weights are defined for constructing Boolean rules. At the finial subsection, an algorithm (Bound Decomposition Tree, BDT) is provided to extract rules from a neuron. Section 3 proposed a pruning method in BDT to decrease the number and the complexity of the extracted rules. In Sect. 4, the BDT algorithm is compared with other two algorithms, M-of-N and Garcez, with the standard MONK's problem. Section 5 gives a simple conclusions and possible future works.

## 2 Bound decomposition tree algorithm

### 2.1 Neural network and cube of input vectors

A *neuron* is an information processing unit that is the fundamental component of a neural network [7]. Generally, a Boolean logic neuron which has $N$ binary inputs can be expressed as:

$$\mathbf{x} = [x_1, x_2, \ldots, x_N], \quad x_i \in \{0, 1\} \tag{1}$$

The output of the neuron is

$$y = f(u(x)) \tag{2}$$

$$u(x) = \sum\nolimits_{i=1}^{N} w_i x_i + w_0 = \mathbf{w}^T \mathbf{x} + w_0 \tag{3}$$

where $\mathbf{w} = [w_1, w_2, \ldots, w_N]$ is the weight vector and $\boldsymbol{w}_0$ is the bias.

The active function $f(\cdot)$ possesses the properties is

$$f(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases} \tag{4}$$

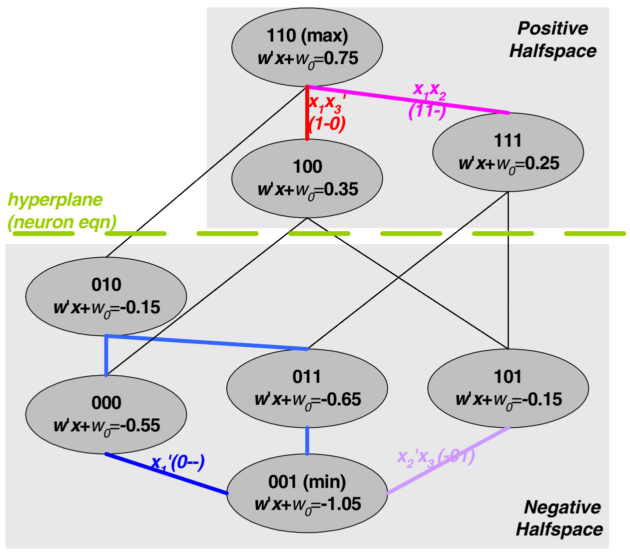

**Fig. 1** The cube lattice

With the above description, to extract the rules from the neuron, it requests to design an effective algorithm to search all possible inputs **x** which suit for $u(\mathbf{x}) \geq 0$. For better understanding, some definitions are described below.

**Definition 1** *(cube)* A cube is a set which contains all input vectors of **x** in a neuron as shown in Fig. 1 and can be expressed as $\boldsymbol{cube}(\mathbf{w}^*)$, where $\mathbf{w}^* = [w_1, w_2, ..., w_N, w_0]$ includes the weights and bias of the neuron.

The input term $\mathbf{x}_i$ of neuron is Boolean logic form (either 0 or 1), then $u(\mathbf{x})$ can be just simply represented as the sum of $w_i$ which $x_i = 1$

$$u(\mathbf{x}) = \sum\nolimits_{i,x_i=1} w_i + w_0 \tag{5}$$

It is easy to find that the absolute maximum of $u(\mathbf{x})$ occurs at $x_i = 1$ for the weights $w_i$ got positive value and $x_i = 0$ for the weights $w_i$ is negative; and the minimum of $u(\mathbf{x})$ occurs at $x_i = 1$ with $w_i$ is negative and $x_i = 0$ with $w_i$ is positive. Accordingly, the upper bound of the cube is the sum of the bias $w_0$ and all $w_i$ with positive values; and the lower bound of the cube is the sum of $w_0$ and all $w_i$ with negative values.

**Definition 2** *(bound of cube)* The *bound* of a cube is the maximum and minimum of $u(\mathbf{x})$ in the cube and can be shown as

$$\boldsymbol{bound}(\boldsymbol{cube}(\mathbf{w}^*)) = [Lbound, Ubound] \tag{6}$$

where the *lower bound* ***Lbound*** is the minimum of $u(\mathbf{x})$ in the cube; and the *upper bound* ***Ubound*** is the maximum of $u(x)$ in the cube

$$Lbound = \sum\nolimits_{i,w_i<0} w_i + w_0 \tag{7}$$

$$Ubound = \sum\nolimits_{i,w_i>0} w_i + w_0$$

*Example 1* Assume that a neuron has three inputs $\{x_1, x_2, x_3\}$, and the corresponding weights are $\{w_1, w_2, w_3\}$, and the bias is $w_0$.

$$w^* = [w_1, w_2, w_3, w_0] = [0.9, 0.4, -0.5, -0.55]$$

and then

$$\begin{aligned}&\text{bound}(\text{cube}(\text{w}^*)) = [\text{lbound}, \text{ubound}]\\ &= [-0.5 + (-0.55), 0.9 + 0.4 + (-0.55)] = [-1.05, 0.75]\end{aligned}$$

## 2.2 Sub-cube

In order to extract out the rules of a neuron for activating the neuron, the cube of the neuron must be sorted into sub-cubes and to find out a sub-cube with bound values lie between the range [0, ∞] (or [–∞, 0]). That means all inputs $x$ of the cube would make the neuron active (or inactive). What we have done here is to assign $x_1$ to be sorted into 0 and 1, so that the original cube is divided into two sub-cubes (one sub-cube with $x_1$ be assigned to be 0; the other with $x_1$ be assigned to be 1). Similarly, each obtained sub-cube can be sorted again and divided into smaller sub-cubes with $x_2$ be assigned. Then with more $x_i$ terms be assigned into sub-cubes, the smaller subset of the sub-cube of input vectors can be obtained.

**Definition 3** *(sub-cube)* A *sub-cube* of a cube is a subset of the input vectors **x** in a neuron. When the first m terms of an input vector $x_i$ in a cube are sorted and assigned, a sub-cube undergo m times sorting is obtained and can be expressed as **cube**(**w^***, $x_1$, $x_2$, ..., $x_m$), where $x_1$, $x_2$, ... , $x_m$ are the assigned values of first m terms of an input vector $x_i$ in the cube.

*Example 2* When only $x_1$ is assigned, the cube can be divided into two sub-cubes,

$$\boldsymbol{cube}(\mathbf{w}^*, \mathbf{1}) \quad \text{and} \quad \boldsymbol{cube}(\mathbf{w}^*, \mathbf{0}).$$

The sub-cube with input vectors that $x_1 = 1$ and $x_2 = 0$ can be expressed as

$$\boldsymbol{cube}(\mathbf{w}^*, \mathbf{10}).$$

As known, each sub-cube is a sub-set of the input vector, and each sub-cube with different inputs would cause different values of $u(\mathbf{x})$.

**Definition 4** *(bound of sub-cube)* The *bound* of a sub-cube is the maximum and minimum of $u(\mathbf{x})$ in the sub-cube and can be shown as

$$\boldsymbol{bound}(\boldsymbol{cube}(\mathbf{w}^*, x_1x_2, \ldots, x_m)) = [\boldsymbol{lbound}, \boldsymbol{ubound}] \quad (8)$$

where the *lower bound* ***lbound*** of the sub-cube is the minimal $u$(x) in the sub-cube, and the *upper bound* ***ubound*** of the sub-cube is the maximal $u$(x) in the sub-cube, where

$$\begin{aligned}&\boldsymbol{lbound}(\boldsymbol{cube}(\mathbf{w}^*, x_1x_2, \ldots, x_m)) \\ &\quad = \min\{u(\boldsymbol{x})|\boldsymbol{x} \in \boldsymbol{cube}(\mathbf{w}^*, x_1x_2, \ldots, x_m)\} \\ &\quad = \sum\nolimits_{i=1,x_i=1}^{m} w_i + \sum\nolimits_{j=m+1,w_j<0}^{n} w_j + w_0\end{aligned} \quad (9)$$

$$\begin{aligned}&\boldsymbol{ubound}(\boldsymbol{cube}(\mathbf{w}^*, x_1x_2, \ldots, x_m)) \\ &\quad = \max\{u(\boldsymbol{x})|\boldsymbol{x} \in \boldsymbol{cube}(\mathbf{w}^*, x_1x_2, \ldots, x_m)\} \\ &\quad = \sum\nolimits_{i=1,x_i=1}^{m} w_i + \sum\nolimits_{j=m+1,w_j>0}^{n} w_j + w_0\end{aligned} \quad (10)$$

*Example 3* According to Example 1, when the neuron's cube is divided into sub-cube with $x_1 = 1$, the sub-cube is ***cube***(**w**$^*$, **1**) and its bound is

$$[\text{lbound, ubound}] = [0.9 + (-0.5) + (-0.55), 0.9 + (0.4) + (-0.55)] = [-0.15, 0.75]$$

When the above sub-cube is divided into two sub-cubes with $x_2$ the bound of the sub-cube ***cube***(**w**$^*$, **10**) is

$$[\text{lbound, ubound}] = [0.9 + (-0.5) + (-0.55), 0.9 + (-0.55)] = [-0.15, 0.35]$$

and the bound of the another sub-cube ***cube***(**w**$^*$, **11**) is

$$[\text{lbound, ubound}] = [(0.9 + 0.4) + (-0.5) + (-0.55), (0.9 + 0.4) + (-0.55)] = [0.25, 0.75]$$

The calculations of bounds via above equations are complicate. Since the bounds of a cube have been counted when the cube being divided into sub-cubes, a more effective method to find out the new bounds of the sub-cube can be achieved with a simpler calculation from the original bounds of the cube. That is, for any cube ***cube***(**w**$^*$, $x_1x_2 \cdots x_m$) with bounds of [lbound$_\text{m}$, ubound$_\text{m}$], the bounds of the new sub-cube ***cube***(**w**$^*$, $x_1x_2 \cdots x_mx_{m+1}$) with $x_{m+1}$ be sorted can be obtained from

$$[lbound_{m+1}, ubound_{m+1}] = \begin{cases}[lbound_m, ubound_m - w_{m+1}], & x_{m+1} = 0, w_{m+1} \geq 0 \\ [lbound_m - w_{m+1}, ubound_m], & x_{m+1} = 0, w_{m+1} < 0 \\ [lbound_m + w_{m+1}, ubound_m], & x_{m+1} = 1, w_{m+1} \geq 0 \\ [lbound_m, ubound_m + w_{m+1}], & x_{m+1} = 1, w_{m+1} < 0\end{cases} \quad (11)$$

**Example 4** According to Example 2, the bound of the sub-cube ***cube***(**w**$^*$,**1**) is [lbound, ubound] = [–0.15, 0.75] When the sub-cube is divided into two sub-cubes with $x_2$, the bound of the sub-cube ***cube***(**w**$^*$,**10**) is

$$[\text{lbound, ubound}] = [-0.15, 0.75 - (0.4)] = [-0.15, 0.35]$$

and the bound of the another sub-cube ***cube***(**w**$^*$,**11**) is

$$[\text{lbound, ubound}] = [-0.15 + (0.4), 0.75] = [0.25, 0.75]$$

The above equations are much simpler, so that the values of the bounds of the sub-cube could be real-time obtained. Through the calculated values of the bounds, some special cubes are defined as following.

**Definition 5** *(positive cube)* If the lower bound of a sub-cube is positive (lbound > 0), it means all the input vectors of the sub-cube would activate the neuron. Such a sub-cube is called ***positive cube***.

**Definition 6** *(negative cube)* If the upper bound of a sub-cube is negative (ubound < 0), it means all the input vectors of the sub-cube would not activate the neuron anymore. Such a sub-cube is called ***negative cube***.

**Definition 7** *(uncertain cube)* If a sub-cube is neither a positive cube nor a negative cube, Such a sub-cube is called ***uncertain cube***. Only some input vectors of the cube would activate the neuron. Hence, it needs to be decomposed again.

## 2.3 Rules of sub-cube

Since a positive cube's lower bound is positive, all $u$(x) in the cube are positive and would make the neuron activated. Oppositely a negative cube's upper bound is negative, thus all inputs in it would not make the neuron activated. Because a positive cube is a set that all input vectors in it would activate the neuron, thus this kind of cube represents as a rule of the neuron. If a cube ***cube***(**w**$^*$, $x_1$, $x_2$, ..., $x_m$) is a positive cube, then the corresponding rule atom of it is

$$\prod_{i=1}^{m} e(x_i) \quad \text{where } e(x_i) = \begin{cases} x_i, & x_i = 1 \\ \overline{x_i}, & x_i = 0 \end{cases} \quad (12)$$

*Example 5* According to Example 4, the bound of ***cube***(**w**$^*$,**11**) is

$$[\text{lbound}, \text{ubound}] = [0.25, 0.75].$$

Because lbound($0.25 \geqq 0$) is positive, the sub-cube is positive cube. Therefore, the neuron has a rule is $x_1\ x_2$.

Similarly, a negative cube also can represent as a rule (negative rule) which wouldn't make the neuron active.

## 2.4 The algorithm of rule extraction

To extract rules from a neuron is method to find out the all sub-sets of input vectors which could make the neuron activated. And the positive cubes are such kinds of sub-sets which all input vectors in them would make neuron activated. Therefore the principle of the bound decomposition tree algorithm (BDT) is to divide the cube of all input vectors of a neuron into sub-cubes repeatedly until all positive cubes had been found. The process of the algorithm can be seen as a hierarchical search.

Algorithm 1. Bound Decomposition Tree (BDT)

```
Input: A neuron's weights and bias
Output: Extract rules of the neuron
Step 1. Set positive cube set, negative cube set, positive rule set, and negative rule set as empty
Set uncertain cube set as cube(w*)
Step 2. Select a cube x form the uncertain cube set
Step 3. Divide the cube x into two sub-cubes: cube m, cube n
Step 4. Compute the bound of cube m and n with Eq. (11)
Step 5. Check the cube m and cube n separately,
    If the cube is positive cube, add it to the positive cube set
    Else if the cube is negative cube, add it to the negative cube set
    Else add it to the uncertain cube set
Step 6. If the uncertain cube set is not empty go to Step 2.
Step 7. Transform each cube in the positive cube set into rules with Eq.(12), and insert them into positive rule set. Similarly, negative cube set also can be transform to negative rules and insert into negative rule set.
Step 8. Then, positive rule set contains the rules which make the neuron active; and negative rule set contains the rules which make the neuron inactive. (end)
```

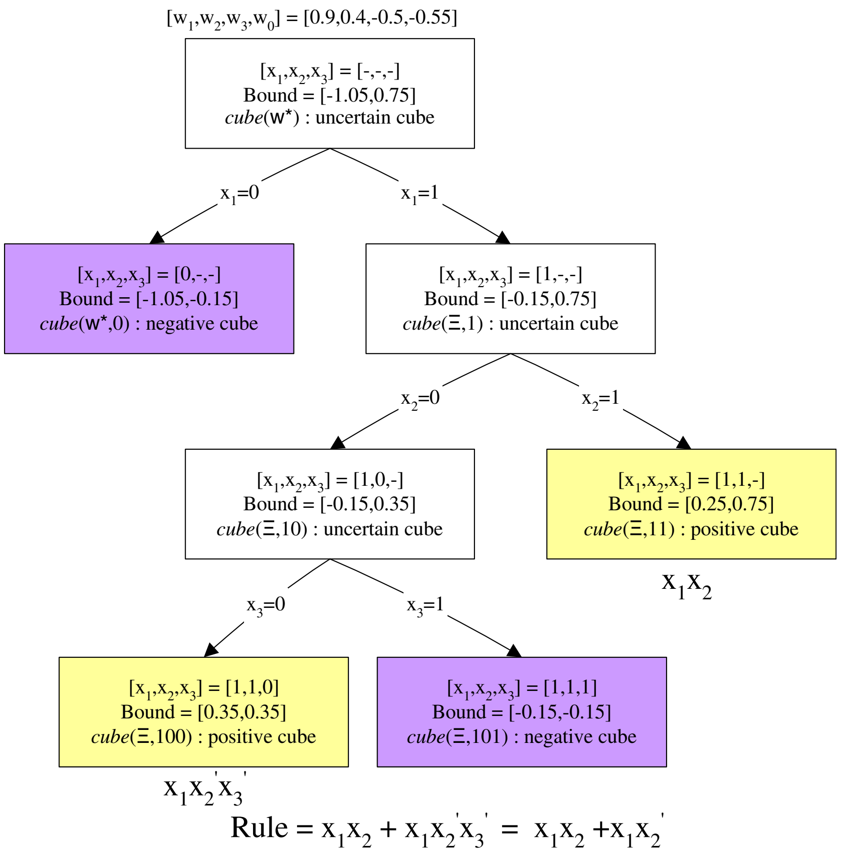

**Fig. 2** The diagram of Example 6

**Example 6** For a neuron with $[w_1, w_2, w_3, w_0] = [0.9, 0.4, –0.5, –0.55]$ as shown in Fig. 2, The bound of the ***cube***($\mathbf{w}^*$) of the neuron is [–0.5 + (–0.55), 0.9 + 0.4 + (–0.55)] = [–1.05, 0.75]. It is an uncertain cube; hence it needs to divide into two smaller sub-cubes with $x_1$.

***cube***($\mathbf{w}^*$, **1**) = [–1.05 + 0.9, 0.75] = [–0.15, 0.75] is still an uncertain cube.
***cube***($\mathbf{w}^*$, **0**) = [–1.05, 0.75 – 0.9] = [–1.05, –0.15] is a negative cube.
***cube***($\mathbf{w}^*$, **1**) is also needs to divide into sub-cubes with $x_2$.
***cube***($\mathbf{w}^*$, **11**) = [–0.15 + (0.4), 0.75] = [0.25, 0.75] is a positive cube.
***cube***($\mathbf{w}^*$, **10**) = [–0.15, 0.75–(0.4)] = [–0.15, 0.35] is still an uncertain cube.
***cube***($\mathbf{w}^*$, **10**) is also needs to divide into sub-cubes with $x_3$.
***cube***($\mathbf{w}^*$, **101**) = [–0.15, 0.35] = [–0.15, –0.15] is a negative cube.
***cube***($\mathbf{w}^*$, **100**) = [–0.15–(–0.5), 0.35] = [0.35, 0.35] is a positive cube

Only ***cube***($\mathbf{w}^*$, **11**) and ***cube***($\mathbf{w}^*$, **100**) are positive cube, so the rules of the neuron are $x_1x_2 + x_1x_2' \, x_3'$.
And the rules can be simplified as $x_1x_2 + x_1x_3'$.

## 3 Rule pruning

In this section, a pruning method would be provided for the BDT algorithm. A neuron's input vectors can be linearly separated into two parts through $u(\mathbf{x})$ ($u(\mathbf{x}) \geq 0$ and $u(\mathbf{x}) < 0$). The input vectors with $u(\mathbf{x}) \geq 0$ would make the neuron active, and others wouldn't. The separation is explicit, but a neuron may not learn perfect and the input vectors which is near the hyper-plane ($u(\mathbf{x}) = 0$) should be fuzzy and uncertain. So, the ideal of the pruning method of BDT algorithm is increasing a threshold Δ that if the input vectors is close enough to the hyper-plan ($–\Delta \leq u(\mathbf{x}) \leq \Delta$), these input vectors will be seen as ''*do not-case*'' state. It means these input vectors could be either active or inactive the neuron, they are unclear. Then they can be classified into the positive cubes or the negative cubes when it needed. With the above concept, it just need to modify the Definitions 5 and 6 as

**Definition 8** *(positive cube with threshold Δ)* A sub-cube is a *positive cube* if its lower bound is large then –Δ.

**Definition 9** *(negative cube with threshold Δ)* A sub-cube is a *negative cube* if its upper bound is small then Δ.

**Example 7** According to Example 4, the threshold Δ is 0.2. The bound of the ***cube***($\mathbf{w}^*$) of the neuron is [–0.5 + (–0.55), 0.9 + 0.4 + (–0.55)] = [–1.05, 0.75]. It is an uncertain cube; thus it needs to divide into two smaller sub-cubes with $x_1$.

$$\boldsymbol{cube}(\mathbf{w}^*, \mathbf{1}) = [-1.05 + 0.9,\ 0.75] = [-0.15,\ 0.75]$$

the lbound is –0.15 > –Δ(–0.2), so it is a positive cube.

$$\begin{aligned}\boldsymbol{cube}(\mathbf{w}^*, \mathbf{0}) &= [-1.05,\ 0.75 - 0.9]\\ &= [-1.05, -0.15] \text{ is a negative cube.}\end{aligned}$$

Then the rule of the neuron is $x_1$ as shown in Fig. 3. □
In the next section, the pruning capability will be shown with Monk's problem.

## 4 Experiments

This section applies the above methods to several experiments. The first example simply extracts the rule of an

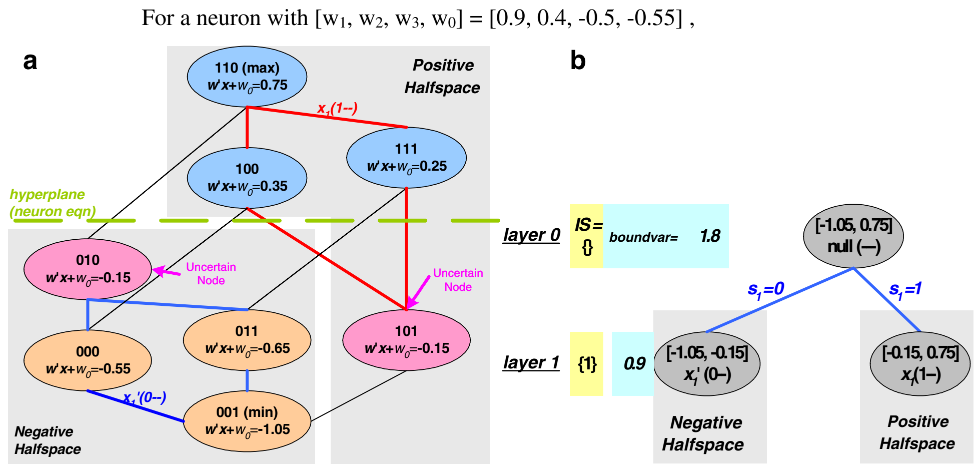

**Fig. 3** **a** the pruned hypercubes, **b** the BDT algorithm with pruning

**Table 1** The conditions for the MONK's problems

| Subproblem | MONK 1 | MONK 2 | MONK 3 |
|---|---|---|---|
| Condition | (head_shape = body_shape) or (jacket_color = red) | Exactly two of the six attributes have their first value | (jacket_color = green and holding = sword) or (jacket_color != blue and body_shape != octagon) |
| Example number | 432 | 432 | 432 |
| Training number | 124 | 169 | 122 |
| Have noise | NO | NO | 5% |

exclusive-OR neural network. The second experiment uses practical data to extract the rules for the Monk's problem. In addition, a comparison with existing algorithms (MofN[4, 5], Garcez [2]), indicates practical engineering applications, as briefly illustrated. And at finally experiment, the BDT algorithm is used in an application of Chinese Medicine Diagnosis.

### 4.1 Exclusive-OR problem

Exclusive-OR (XOR) function is the simplest nonlinear problem for neural networks, which can only be solved by a multi-layer neural network. Here, we use this XOR function as the first trial for our extraction algorithm. The constructed neural network has two inputs ($x_1$ and $x_2$), 2 hidden neurons ($y_1$ and $y_2$) and 1 output ($z_1$). The inputs to this network are adopted from one of the combinations (0, 0), (0, 1), (1, 0) or (1, 1) and the threshold functions of both hidden and output neurons are log-sigmoid.

After the neural network is trained with data of $\overline{XOR}$, the weights (including bias) of hidden node $y_1$ is $\mathbf{w_1} = [w_{11}, w_{12}, w_{10}] = [-11.2272, -8.9857, 15.1622]$, the weights of hidden node $y_2$ is $\mathbf{w_2} = [w_{21}, w_{22}, w_{20}] = [10.9370, 12.4174, -5.7943]$, and the weights of output neuron $z$ is $\mathbf{v} = [v_1, v_2, v_0] = [-28.0095, -32.2535, 44.7402]$. Then our rule-extraction algorithm is applied to the above three neurons in hidden and output layers. The extracted positive rules are $y_1 = R_{p,\mathbf{w1}^*} = x_1' + x_1x_2', y_2 = R_{p,\mathbf{w2}^*} = x_2 + x_1x_2'$, and $z = R_{p,\mathbf{v}^*} = y_2' + y_1'y_2$. Obviously, the output rule can be rewritten as $z = R_{p,\mathbf{v}^*} = y_2' + y_1'y_2 = x_1'x_2' + x_1x_2$, which exactly accomplishes the $\overline{XOR}$ function.

### 4.2 Comparison with MONK's problems

This subsection applies the above rule-extraction algorithm to the well-known MONK's problem [17], which has been widely used as a benchmark for machine learning systems and many rule extraction algorithms. Then with the same problem, BDT will be compared with those from MofN [4, 5] and from Garcez's algorithm [2].

In MONK's problem, experimental data come from robots with six attributes: head_shape ∈ {round, square, octagon}, body_shape ∈ {round, square, octagon}, is_smiling ∈ {yes, no}, holding ∈ {sword, balloon, flag}, jacket_color ∈ {red, yellow, green, blue}, and has_tie ∈ {yes, no}. MONK's problem includes three subproblems, each of which has robots with different conditions, as Table 1 described.

The neural networks for MONK's problem as used for rule extraction are Thrun's trained BPNs [17]. The accuracies of the networks for problem 1, 2 and 3, to training data is 100, 100, and 93.1%, respectively. With the proposed algorithm in this paper, the corresponding rules are extracted with the accuracies 99.5, 100, and 97.2%, as shown in Fig. 4.

For the example of MONK 1, only 2 examples (#185 = [010100010011000 10, 1] and #186) of the rule sets make differences for the resultant network; and in MONK 3, even 12 examples are better. The calculated outputs of hidden nodes ($h_1$, $h_2$, $h_3$) are [0.4367, 0.9999, 0.7249] with hard-limited values [011]. The weights of the output neuron are [9.249339, 8.639715, –9.419991, –3.670920], and then Algorithm 1 gives the extracted rule $h_1h_2 + h_1h_3' + h_2h_3'$. It can be observed that *logsig*(0.4367*9.249339 + 0.9999*8.639715 + 0.7249 *(–9.419991) –3.670920) = 0.8984 > 0.5 and *hardlimit* (0*9.249339 + 1*8.639715 + 1*(–9.419991) –3.670920) = 0. Therefore, when the networks' threshold functions are

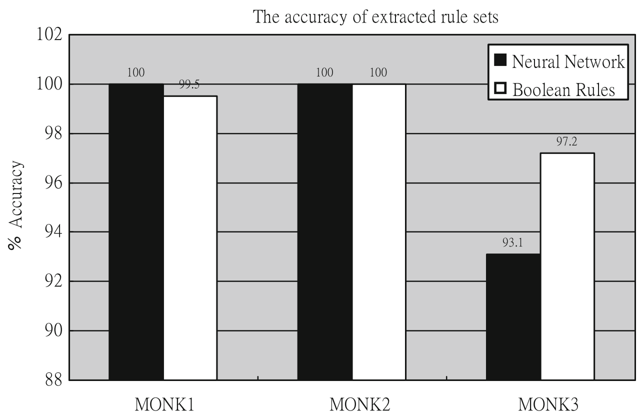

**Fig. 4** The accuracies of the experimental NNs and extracted rules

replaced from log-sigmoid to hard-limited, the accuracies of the extracted rules of the networks for all these three problems are 100%. This is because the output of $h_1$ (0.4367) is too close to the threshold (0.5). This phenomenon corresponds to the unsound and incomplete problem of decompositional extraction algorithm described by Garcez [2].

At next, BDT will be compared with those from MofN [4, 5] and from Garcez's algorithm [2]. MofN adopts special (m of n) rule formats, which makes the extracted rules more concise and the processing speed faster. On the contrary, Garcez's algorithm [2] uses Boolean logic. All these algorithms are implemented by MATLAB and Java together, and the test data comes from 100 execution cycles of MONK's Problem 1 [17]. The experiment environment is Intel Celeron 1.7GHz CPU and Microsoft Windows XP. The test results (rule numbers and rule antecedent numbers) are listed in Table 2. Here, the BDT algorithm is tested by no pruning and by pruning with two thresholds, while MofN is checked by three clusters.

In Table 2, $h_1$–$h_3$ are three hidden nodes in the hidden layer and $o_1$ is the output neuron. ''Rule Number'' is the rule number extracted from a neuron and ''Rule Antecedent Number'' is the number of antecedents of all rules. Obviously, a smaller ''Rule Number'' represents fewer rules and a smaller ''Rule Antecedent Number'' represents more concise rules. Table 2 shows that the pruning makes ''Rule Number'' and ''Rule Antecedent Number'' greatly reduced at a little cost of precision. At the same representation of Boolean logic, ''Rule Number'' from BDT is less than that from others. It can be found that the pruning makes BDT algorithm greatly improved. Thus, a suitable choice of pruning threshold brings a smaller number of extracted rules with more concise forms with finite influence. As for execution speed, BDT performs like MofN, but it has better performance and more concise rule forms then Garcez's algorithm.

## 4.3 The Voting Record

In the sub-section, BDT Algorithm is tested with the voting record [35]. These data have 435 Instances (267 democrats, 168 republicans), and the number of attributes is 16 (all Boolean valued). In the thesis of Schlimmer show that there are about 90–95% accuracy appears to be STAGGER's asymptote [36].

At the beginning, we create a neuron with 16 inputs to be trained with the above voting record. The test result shows that the accuracy of the neuron is 98.62% (six records are missed). As the MONK's problem, we also use BDT, Garcez and MofN to extract rules from the neuron, and the result is show in Table 3.

In Table 3, it shows that BDT algorithm has great accuracy, and it can reduce the extracted rules number and complexity with pruning. When the threshold $\Delta = 0.3$, the prediction rules number is only 11(for Democrats) and 4(for Republicans).

Next, we try to find out the correlation variable between the attributes and classes of the voting records. Then only eight attributes with high correlation variable are the choice inputs of a new neuron. The choice attributes are:

**Table 2** The test results for BDT, MofN, and Garcer algorithms

| | BDT | BDT with pruning ($\Delta = 2$) | BDT with pruning ($\Delta = 4$) | MofN ($k$ mean = 7) | MofN ($k$ mean = 3) | MofN ($k$ mean = 2) | Garcez |
|---|---|---|---|---|---|---|---|
| Rule Format | Boolean Logic | Boolean Logic | Boolean Logic | M-of-N | M-of-N | M-of-N | Boolean Logic |
| Rule Number | | | | | | | |
| $h_1$ | 286 | 74 | 34 | 11 | 4 | 3 (2,315)[a] | 45,369 |
| $h_2$ | 1969 | 84 | 29 | 45 | 9 | 3 (7,956) | 50,276 |
| $h_3$ | 531 | 24 | 11 | 12 | 5 | 3 (780) | 54,168 |
| $o_1$ | 3 | 3 | 3 | 1 | 1 | 1 | 3 |
| Rule Antecedent Number | | | | | | | |
| $h_1$ | 1909 | 360 | 145 | 36 | 8 | 5 | 417,131 |
| $h_2$ | 15692 | 418 | 129 | 166 | 20 | 5 | 454,960 |
| $h_3$ | 3685 | 92 | 36 | 35 | 10 | 3 | 47,7476 |
| $o_1$ | 6 | 6 | 6 | 1 | 1 | 1 | 6 |
| Accuracy | 99.5% | 98.6% | 97.2% | 99.5% | 98.6% | 98.1% | 99.5% |
| Execute Time (Running 1000 times) unit : second | | | | | | | |
| Matlab | 71.0 | 21.5 | 18.4 | 76.7 | 25.16 | 16.6 | 35,094 |
| Java | 26.1 | 1.21 | 0.90 | 3.14 | 1.26 | 0.59 | 969.5 |

[a] The number in parenthesis is the rule number of Boolean logic format

**Table 3** The test results of voting record for BDT, MofN, and Garcer algorithms

| | BDT Δ = 0 | BDT Δ = 0.2 | BDT Δ = 0.3 | Garcez | MofN (*k* = 7) | MofN (*k* = 5) | MofN(*k* = 3) |
|---|---|---|---|---|---|---|---|
| Extracted Rule Number | | | | | | | |
| Democrats | 754 | 35 | 11 | 22452 | 6 (924)[a] | 6 (1428)[a] | 4 (3024)[a] |
| Republicans | 722 | 20 | 4 | | | | |
| Accuracy | | | | | | | |
| To data | 97.24% | 94.25% | 91.72% | 97.47% | 93.56% | 95.86% | 95.17% |
| To Neuron | 98.62% | 95.17% | 92.64% | 98.62% | 93.33% | 96.09% | 94.94% |

[a] The number in parenthesis is the rule number of Boolean logic format

| | |
|---|---|
| $x_1$ | = physician-fee-freeze |
| $x_2$ | = adoption-of-the-budget-resolution |
| $x_3$ | = el-salvador-aid |
| $x_4$ | = education-spending |
| $x_5$ | = aid-to-nicaraguan-contras |
| $x_6$ | = mx-missile |
| $x_7$ | = superfund-right-to-sue |
| $x_8$ | = education-spending |

After training, it shows that the accuracy of the neuron is 96.09% (17 records are missed). And the comparison of extracted rules is showed in Table 4.

From the Table 4, it shows that after reduce the inputs number of the neuron with correlation variable; all the extracted rule numbers of BDT, Garcez and MofN are obviously decreasing and the accuracies still keep very well. The extracted rules of BDT(Δ = 0.3) are:

$$\begin{aligned}\text{Democrat rule} &= x_1' x_3 + x_1' * x_5' \\ &= ((\text{physician-fee-freeze is no}) \\ &\quad \text{and}(\text{el-salvador-aid is yes})) \text{ or} \\ &\quad ((\text{physician-fee-freeze is no}) \\ &\quad \text{and}(\text{aid-to-nicaraguan-contras is no}))\end{aligned}$$

$$\begin{aligned}&\text{Republican rule} = x_1 * x_2' \\ &= (\text{physician-fee-freeze is yes}) \text{ and} \\ &\quad (\text{adoption-of-the-budget-resolution is no})\end{aligned}$$

The above rules are very simple, but still have high accuracy (96.55% to neuron, 93.56% to voting data).

## 4.4 The mushroom data

In the subsection, the mushroom data [37] is used to test the BDT algorithm and compare to Garcez's algorithm and MofN. The data is asymptoted to 95% classification accuracy after reviewing 1,000 instances [36]. There are 8,124 instances in the data, and each instance has 22 attributes. The output class of the instances is poisonous or not.

Because the attributes of the mushroom data are multi-value, hence it needs to create a neuron with 124 inputs to learn the knowledge of the data. After training, the accuracy of the neuron is 100%. It seems to be great, but the number of the neuron's inputs is too large. No matter BDT algorithm, Gracez's algorithm or MofN are all needed very long time to extract the rules, and the number of the extracted rules is huge. Therefore, as the method used in voting record, we find out the correlation variable of each inputs to the class. When it reduces the number of training data inputs to be 15, the accuracy of the neuron is 98.42% (128 mistakes in 8,124 instance). Then we try to reduce the number of inputs to be only 8, the accuracy of the neuron is still great (97.74%, 184 mistakes). The eight choice inputs are show as the following:

| | |
|---|---|
| $x_1$: | odor = none |
| $x_2$: | odor = foul |

**Table 4** The test results of voting record for BDT, MofN, and Garcer algorithms with only eight attributes

| | BDT Δ = 0 | BDT Δ = 0.2 | BDT Δ = 0.3 | Garcez | MofN (*k* = 7) | MofN (*k* = 5) | MofN (*k* = 3) |
|---|---|---|---|---|---|---|---|
| Extracted Rule Number | | | | | | | |
| Democrats | 11 | 3 | 2 | 82 | 6(16)[a] | 4(16)[a] | 3(16)[a] |
| Republicans | 11 | 2 | 1 | | | | |
| Accuracy | | | | | | | |
| To data | 95.86% | 94.02% | 93.56% | 95.86% | 94.94% | 95.17% | 95.17% |
| To Neuron | 99.31% | 97.01% | 96.55% | 99.31% | 97.93% | 97.70% | 97.70% |

[a] The number in parenthesis is the rule number of Boolean logic format

$x_3$: stalk-surface-above-ring = silky
$x_4$: stalk-surface-below-ring = silky
$x_5$: ring-type = pendant
$x_6$: gill-size = broad
$x_7$: gill-color = buff
$x_8$: bruises = yes

The extracted results are shown in Table 5.

The extracted rules of BDT algorithm with Δ = 0.2 are:

$$\text{Poisonous rule} = x_1'x_2 + x_2x_3x_6' + x_1'x_3x_6' + x_2x_3x_8' + x_1'x_3x_8' + x_2x_6'x_8' + x_1'x_6'x_8'$$

= ((odor is foul)) or ((odor is foul)and(stalk-surface-above-ring is silky)and(gill-size is narrow))or
((odor is not none) and(stalk-surface-above-ring is silky) and(gill-size is narrow))or
((odor is foul)and(stalk-surface-above-ring is silky)and(bruises is no))or
((odor is not none)and(stalk-surface-above-ring is silky)and(bruises is no))or
((odor is foul)and(gill-size is narrow)and(bruises is no))or
((odor is not none)and(gill-size is narrow)and(bruises is no))

$$\text{Non-poisonous rule} = x_1x_2'x_6 + x_1x_2'x_3' + x_2'x_3'x_6x_8$$

= ((odor is none)and(gill-size is broad))or
((odor is none)and(stalk-surface-above-ring is not silky))or
((odor is not foul)and(stalk-surface-above-ring is not silky)and)or(bruises is yes)

The above rules are also simple and have high accuracy (95.37%).

## 4.5 Application to chinese medicine diagnosis

At this point, BDT is applied to a real problem. In the field of Chinese medicine, an expert system is built by BDT for diagnosing coronary artery disease (CAD) [20]. This expert system is a hybrid-intelligence system. This means its major framework is a traditional expert system with knowledge base, but the knowledge acquisition process is implemented with neural network and rule extraction.

The system is built in 4 steps: (1) Data collection: A questionnaire about CAD is designed with 72 items, including 53 Chinese symptoms, 10 clinical diagnostic records and 9 syndromes determined by doctors [19]. And this questionnaire is used to collect 47 case data from the Far Eastern Memorial Hospital from Novemeber 2003 to April 2004. (2) Input selection: by correlation analysis, 30 out of 63 inputs (Chinese symptoms and clinical diagnostic records) are selected for the neuron inputs, as Table 6 a listed. (3) Neural network training: Build a 30 × 9 single-layer perceptron and execute a 5,000-epoch delta learning rule. (4) Knowledge base with rule extraction: BDT algorithm can extract rules from the above neural network, as Table 6 b shows. These rules can be stored into a knowledge base of expert system.

The extracted rules are proposed to the original doctor for his discussion. The rules of heart qi vacuity, heart yin vacuity and kidney yang vacuity have connection with the clinical diagnostic records. For example, heart qi vacuity relates to 3VD. The major reason is that coronary thrombosis causes anoxemic heart muscles and the qi and blood of the patients with anoxemic heart muscles are vacuous.

**Table 5** The test results of mushroom data for BDT, MofN, and Garcer algorithms with only eight attributes

| | BDT Δ = 0 | BDT Δ = 0.1 | BDT Δ = 0.2 | Garcez | MofN (*k* = 7) | MofN (*k* = 5) | MofN (*k* = 3) |
|---|---|---|---|---|---|---|---|
| Extracted Rule Number | | | | | | | |
| Poisonous | 18 | 13 | 7 | 106 | 8(24)[a] | 8(25)[a] | 4(28)[a] |
| Not poisonous | 22 | 7 | 3 | | | | |
| Accuracy | | | | | | | |
| To data | 97.74% | 95.77% | 95.27% | 97.74% | 97.74% | 97.74% | 97.74% |
| To Neuron | 100% | 97.74% | 95.37% | 100% | 100% | 100% | 100% |

[a] The number in parenthesis is the rule number of Boolean logic format

**Table 6** (a) Network inputs; (b) syndrome outputs and their corresponding outputs

| (a) Inputs (all are Chinese symptoms except $x_{10}$ and $x_{12}$) | | (b) Syndrome outputs and the corresponding extracted rules (the numbers are the counts of data sets with values = 1's) | |
|---|---|---|---|
| | | Syndrome | Rule |
| $x_1$ | Oppressed in the chest | Qi stagnation (20) | $x_1$ (17), $x'_1x_2$ (3) |
| $x_2$ | Heavy-headedness | | |
| $x_3$ | Purplish lip | Blood stasis (29) | $x_3$ (28), $x'_3x_4$ (1) |
| $x_4$ | Pain at fixed position | | |
| $x_5$ | Dry tongue fur | Thick phlegm (3) | $x_5x_6$ (1), $x'_5x_6x_7$ (2) |
| $x_6$ | Phlegm | | |
| $x_7$ | Thick tongue fur | Qi stasis due to cold (5) | $x_8$ (5) |
| $x_8$ | Cold | | |
| $x_9$ | Panting | Heart qi vacuity (19) | $x_9x_{10}$ (11), $x_9x'_{10}x_{11}$ (6), |
| $x_{10}$ | 3VD | | $x_9x'_{10}x'_{11}x_{12}$ (2) |
| $x_{11}$ | Sleep disorders | Heart blood vacuity (1) | $x_{13}$ (1) |
| $x_{12}$ | LCX | | |
| $x_{13}$ | Bloodless complexion | Heart yin vacuity (20) | $x_{14}\ x'_{15}x'_{16}$(5), $x'_{14}x'_{15}x'_{16}x_{17}x_{18}$(1), |
| $x_{14}$ | Few tongue fur | | $x_{14}\ x'_{15}x_{16}x_{17}\ x'_{18}$(1), |
| $x_{15}$ | frequent urination | | $x_{14}\ x_{15}x_{16}\ x_{17}\ x_{18}\ x_{19}$ (1), |
| $x_{16}$ | Unstable Angina | | $x_{14}\ x_{15}\ x'_{16}x_{17}\ x'_{18}x_{19}$ (1), |
| $x_{17}$ | Fissured tongue | | $x'_{14}x'_{15}x'_{16}x_{17}\ x'_{18}x_{19}$ (5), |
| $x_{18}$ | Lazy speech | | $x_{11}\ x_{14}\ x_{15}\ x'_{16}x'_{17}x'_{18}x_{19}$ (1), |
| $x_{19}$ | Hypertension | | $x_{11}\ x'_{14}x'_{15}x'_{16}x'_{17}{}'x'_{18}x_{19}$ (2), |
| $x_{20}$ | Afraid of hot | | $x_{11}\ x'_{14}x'_{15}\ x'_{16}\ x_{17}\ x'_{18}\ x'_{19}$ (1), |
| $x_{21}$ | Oppressed in the chest | | $x_{11}\ x'_{14}\ x'_{15}\ x'_{16}\ x'_{17}\ x_{18}\ x'_{19}$ (1), |
| $x_{22}$ | Afraid of cold | | $x'_{11}\ x_{14}\ x'_{15}\ x_{16}\ x'_{17}\ x_{18}\ x'_{19}\ x'_{20}$ (1) |
| $x_{23}$ | Oliguria | Heart yang vacuity (4) | $x_{14}\ x'_{15}x'_{21}\ x_{22}\ x_{23}\ x'_{24}\ x_{25}\ x'_{26}\ x'_{27}$ (1), |
| $x_{24}$ | Loss of bladder control | | $x'_{14}\ x'_{15}\ x_{21}\ x_{22}\ x'_{23}\ x'_{24}x_{25}\ x_{26}\ x'_{27}$ (1), |
| $x_{25}$ | Felling of inertia on weak waist and knee | | $x_{14}\ x_{15}\ x_{21}\ x_{22}\ x'_{23}\ x'_{24}x'_{25}\ x_{26}x_{27}\ x'_{28}$ (1), |
| $x_{26}$ | Puffiness of legs | | $x'_8\ x_{14}x_{15}\ x_{21}\ x'_{22}\ x_{23}\ x_{24}\ x'_{25}x_{26}\ x_{27}\ x'_{28}x'_{29}$ (1) |
| $x_{27}$ | Disordered pulse | Kidney yang vacuity (8) | $x_{26}\ x'_{27}\ x_{29}$ (1), $x'_5x'_{26}\ x'_{27}\ x_{29}$ (3), $x'_5x'_{26}\ x_{27}x_{29}x_{30}$ (1), $x'_5\ x_{18}x'_{26}\ x'_{27}x'_{29}\ x_{30}$ (1), |
| $x_{28}$ | Cold of hands& legs | | |
| $x_{29}$ | Weak speech | | $x_5\ x_{18}\ x_{26}\ x_{27}\ x_{29}\ x'_{30}$ (1), |
| $x_{30}$ | Diabetes | | $x'_5\ x'_{10}\ x'_{18}\ x'_{26}\ x'_{27}x_{29}\ x'_{30}$ (1) |

Such rule (heart qi vacuity —$x_9x_{10}$) connects eastern and western medical practicer.

## 5 Conclusions

Based on the analysis of inputs and weights, this paper proposes an algorithm to extract the positive and negative Boolean rules of a neuron at the same time, then extends the rules for a neural network. From the hyperplane function of the neuron, the positive and negative sub-cubes in positive and negative halfspaces and their upper and lower bounds can be found. The corresponding bounds of decomposed sub-cubes have some specific relationships. Therefore, starting from the overall inputs cube and we can construct a BDT (bound decomposition tree) through sorted neuron weight, thus the positive, negative, and even uncertain, Boolean rules of the rules can be obtained. These rules can be further pruned to get simpler forms.

This algorithm is verified by several data. Firstly, an exclusive-OR neural network is introduced to check the results of extracted rules. The second example uses MONK's problems to test BDT algorithm and also compares to Garcez's algorithm and MofN. It shows that BDT (including pruning) performs like MofN [4, 5], but compared to Garcez's algorithm [2], BDT has better performance and more concise rule forms.

There are four important major criteria (*accuracy, fidelity, consistency* and *comprehensibility*) of rule quality provided by Andrew et al. [12]. In the experiment results of MONK's problem, voting records and mushroom data, all show that the ***accuracy*** of the BDT algorithm is great. With the pruning option, BDT algorithm can effectively reduce the size and complexity of the extracted rules and keep excellent accuracy. In the experiment of voting records and mushroom data, the size and of rules extracted by BDT algorithm with pruning is very little, and the number of antecedents of per rule are also few. Thus the ***comprehensibility*** of extracted rules is also excellent. Because BDT algorithm analyzes the weights of neural network directly and uses the decomposition method to check the bound value of difference input value sets, then find the fit input combinations to extract the rules of neural network. Hence the rules should mimic the behavior of the neural network and the ***fidelity*** of extracted rules is good. Finally, the ***consistency*** of the extracted rules is also well.

In the paper of Andrew et al. [12], they also provided five primary classification criteria (*rule format*, *quality*, *translucency*, *complexity* and *portability*) of rule extraction algorithm.

| | |
|---|---|
| ***Rule Format***: | Boolean logic. |
| ***Quality***: | Extracted rules show high fidelity, accuracy and simple. |
| ***Translucency***: | Decompositional. |
| ***Complexity***: | Exponential. |
| ***Portability***: | Generally applicable to feed-forward networks. |

Although the complexity of BDT algorithm is exponential, but in the experiment results of MONK's problem, the calculate time of BDT can be substantially reduce with the pruning. With appropriate threshold (Δ), the accuracy of extracted rules with pruning is still excellent. Besides that, the size and complexity of extracted rules also can be reduced.

Finally in Sect. 4.5, BDT is applied to a real expert system of Chinese medical diagnosis to show its practicability. In the future, the BDT algorithm will be improved to extract rules from continuous value input neural networks.

## References

1. Tickle AB, Andrews R, Golea M, Diederich J (1968) The truth will come to light: directions and challenges in extracting the knowledge embedded within trained artificial neural networks. IEEE Trans Neural Netw 9(6):1057–1068
2. d'Avila Garcez AS, Broda K, Gabbay DM (2001) Symbolic knowledge extraction from trained neural networks: a sound approach. Artif Intell 125:155–207
3. Pop E, Hayward R, Diedrich A (1994) RULENEG: extracting rules from a trained neural network using stepwise negation, Technical Report, (Neurocomputing Research Centre, Queensland University of Technology, 1994)
4. Towell GG, Shavlik JW (1993) The extraction of refined rules from knowledge-based neural networks. Mach Learn 13:71–101
5. Towell GG, Shavlik JW (1994) Knowledge-based artificial neural networks. Artif Intell 70:119–165
6. Tsukimoto H (2000) Extracting rules from trained neural networks. IEEE Trans Neural Netw 11(2):377–389
7. Freeman JA, Skapura DM (1992) Neural networks. Addison-Wesley, Reading
8. Fu LM (1994) Rule generation from neural networks. IEEE Trans Syst Man Cybern 28(8):1114–1124
9. Pratt LY, Mostow J, Kamm CA (1991) Direct transfer of learned information among neural networks. In: Proceedings of the 9th National Conference on Artificial Intelligence. Anaheim, pp 584–589
10. Craven MW, Shavlik JW (1994) Using sampling and queries to extract rules from trained neural networks. In: Proceedings of the 11th International Conference on Machine Learning. New Brunswick, pp 37–45
11. Jackson P (1999) Introduction to expert systems. Addison-Wesley, Reading
12. Andrews R, Diederich J, Tickle AB (1995) Survey and critique of techniques for extracting rules from trained artificial neural networks. Knowl Base Syst 8(6):373–389
13. Andrews R, Geva S (1995) Inserting and extracting knowledge from constrained error back propagation networks. In: Proceedings of the 6th Australian Conference on Neural Networks. Sydney
14. Andrews R, Geva S (2002) Rule extraction from local cluster neural nets. Neuro Comput 47:1–20
15. Setiono R (1997) A penalty function approach for pruning feedforward neural networks. Neural Comput 9(1):185–204
16. Setiono R (1997) Extracting rules from neural networks by pruning and hidden-unit splitting. Neural Comput 9(1):205–225
17. Thrun SB, et al. (1991) The MONK's problems: a performance comparison of different learning algorithms, Technical Report CMU-CS-91-197. Carnegie Mellon University
18. Thrun SB (1994) Extracting provably correct rules from artificial neural networks, Technical Report IAI-TR-93-5. Institut für Informatik III, University of Bonn, Germany
19. Naturopathic association (1999) Naturopathy of coronary artery disease. Chinese edn. Ye-He Publishing
20. Chen JC, Liu TS, Weng CS, Heh JS (2005) An expert system of coronary artery disease in Chinese and western medicine. In: Proceedings of the 6th Asian–Pacific conference on medical and biological engineering. Tsukuba
21. Pham DT, Salem Z (2004) A new technique for rule pruning in machine learning, In: International conference on information and communication technologies: from theory to applications. Damascus, pp 437–438
22. Chang M, Chen JC, Chang JW, Heh JS (2006) Advanced process control expert system of CVD membrane thickness based on neural network. Material Science Forum, pp 505–507, 313–318
23. Krishnan R, Sivakumar G, Bhattacharya P (1999) A search technique for rule extraction from trained neural networks. Pattern Recognit Lett 20:273–280
24. Taha IA, Ghosh J (1999) Symbolic interpretation of artificial neural networks. IEEE Trans Knowl Data Eng 11(3):448–463
25. Chen JC, Heh JS, Chang M (2006) Designing A Decompositional Rule Extraction Algorithm for Neural Networks. Lect Notes Comput Sci 3971:1305–1311
26. Chang M, Chen JC, Heh J (2006) The control of membrane thickness in Pecvd process utilizing a rule extraction technique of neural networks. Lect Notes Comput Sci 3973:1091–1098
27. Chen JC, Chang M, Chu KK (2006) Representing the abnormal Web access for advertisement counting based on the rule-extraction mechanism of neural networks. In: Proceedings of the 8th International conference on artificial intelligence and soft computing. Zakopane

28. Takagi T, Sugeno M (1985) Fuzzy identification of system and its applications to modeling and control. IEEE Trans Syst Man Cybern 15(0):116–132
29. Roger Jang JS, Sun CT (1993) Functional equivalence between radial basis function networks and fuzzy inference systems. IEEE Trans Neural Netw 4:156–158
30. Leng G, McGinnity TM, Prasad G (2005) An approach for on-line extraction of fuzzy rules using a self-organising fuzzy neural network. Fuzzy Sets Syst 150:211–243
31. McGarry KJ, Wermter S, MacIntyre J (2001) Knowledge extraction from local function networks. In: Proceedings of the 17th international joint conference on artificial intelligence. Seattle
32. Ultsch A, Halmans G, Mantyk R (1991) CONKAT: a connectionist knowledge acquisition tool. In: Proceedings of the 24th Hawaii IEEE International Conference on System Sciences, Hawaii
33. Castellano G, Fanelli AM, Mencar C (2005) A neuro-fuzzy network to generate human-understandable knowledge from data. Cogn Syst Res 3:125–144
34. Malone J, McGarry KJ, Bowerman C, Wermter S (2006) Rule extraction from Kohonen neural networks. Neural Comput Appl J 15(1):9–17
35. Jeff Schlimmer (1984) United States congressional voting records database, Congressional Quarterly Almanac, 98th Congress, 2nd session 1984, vol XL. Congressional Quarterly Inc., Washington, DC
36. Jeff Schlimmer (1987) Concept acquisition through representational adjustment, Doctoral dissertation. Department of Information and Computer Science, University of California, Irvine
37. Jeff Schlimmer (1981) Mushroom records drawn from The Audubon Society field guide to north American mushrooms, In: Alfred A. Knopf (ed). G. H. Lincoff (Pres), New York