Applied Intelligence 12, 15–25 (2000)

FERNN: An Algorithm for Fast Extraction of Rules from Neural Networks

RUDY SETIONO AND WEE KHENG LEOW
School of Computing, National University of Singapore, Lower Kent Ridge Road, Singapore 119260
rudys@comp.nus.edu.sg
leowwk@comp.nus.edu.sg

Abstract. Before symbolic rules are extracted from a trained neural network, the network is usually pruned so as to obtain more concise rules. Typical pruning algorithms require retraining the network which incurs additional cost. This paper presents FERNN, a fast method for extracting rules from trained neural networks without network retraining. Given a fully connected trained feedforward network with a single hidden layer, FERNN first identifies the relevant hidden units by computing their information gains. For each relevant hidden unit, its activation values is divided into two subintervals such that the information gain is maximized. FERNN finds the set of relevant network connections from the input units to this hidden unit by checking the magnitudes of their weights. The connections with large weights are identified as relevant. Finally, FERNN generates rules that distinguish the two subintervals of the hidden activation values in terms of the network inputs. Experimental results show that the size and the predictive accuracy of the tree generated are comparable to those extracted by another method which prunes and retrains the network.

Keywords: rule extraction, penalty function, MofN rule, DNF rule, decision tree

1. Introduction

Neural networks have been successfully applied in a variety of problem domains. In many applications, it is highly desirable to extract symbolic classification rules from these networks. Unlike a collection of network weights, symbolic rules can be easily interpreted and verified by human experts. They can also provide new insights into the application problems and the corresponding data. It is not surprising that in recent years there has been a significant amount of work devoted to the development of algorithms that extract rules from neural networks [1–8].

In order to obtain a concise set of symbolic rules, redundant and irrelevant units and connections of a trained neural network are usually removed by a network pruning algorithm before rules are extracted [5, 6, 9, 10]. This process can be time consuming as most algorithms for neural network pruning such as Optimal Brain Surgeon [11], Hagiwara algorithm [12], Neural Network Pruning with Penalty Function (N2P2F) [13], and the algorithm of Castellano et al. [14] are iterative in nature. They retrain the network after removing some connections or units. The retrained network is then checked to see if any of its remaining units or connections meet the criteria for further removal. More often than not, the amount of computations incurred during retraining is much higher than that needed to train the original fully connected network.

This paper presents a method for extracting symbolic rules from trained feedforward neural networks with a single hidden layer. The method does not require network pruning and hence no network retraining is necessary. By eliminating the need to retrain the network, we can speed up the process of rule extraction considerably and thus make neural networks an attractive tool for generating symbolic classification rules. The well-known Monk problems [15] will be used to illustrate in detail how rules are generated by our new method. They consists of three artificial classification problems that distinguish between monks and non-monks.

The paper is organized as follows. In Section 2 we describe the two main components of our method. In

Section 3 we present the criterion for removing the network connections from the input units to the hidden units. In Section 4 we describe how rules are generated for the Monk2 problem. More illustration of how our method works is presented in Section 5 using the Monk1 and Monk3 problems. In Section 6 we present experimental results on 15 benchmark problems that compare the performance of our new method and another rule extraction algorithm which retrains the network during network pruning. Since no network pruning and retraining is required, the new method executes faster than other algorithms which extract rules from skeletal pruned networks. Nevertheless, our results show that the rules extracted by the proposed method are similar in complexity and accuracy to those generated using an existing algorithm. Finally, in Section 7 we discuss related work and conclude the paper.

2. FERNN: Fast Extraction of Rules from Neural Networks

FERNN consists of the following steps:

1. Train a fully connected network such that an augmented cross-entropy error function is minimized. The usual cross entropy error function is augmented by a penalty function so that relevant and irrelevant network connections can be distinguished by their weights when training terminates.
2. Use information gains to identify the relevant hidden units and build a decision tree that classifies the patterns in terms of the network activation values.
3. For each hidden unit whose activation values are used for node splitting in the decision tree, remove the irrelevant input connections to this hidden unit. The connections are irrelevant if their removal does not affect the decision tree's classification performance.
4. For data sets with discrete attributes, replace each node splitting condition by an equivalent set of symbolic rules.

FERNN consists of two main components. The first is a network training algorithm that minimizes a cross-entropy error function augmented by a penalty function. The minimization of the augmented error function ensures that connections from irrelevant inputs have very small weights. Such connections can be removed without affecting the network's classification accuracy. The second component is a decision tree generating algorithm which generates a tree classifier using the activation values of the network's hidden units.

After a decision tree is generated, we distinguish the relevant network inputs from the irrelevant ones. We have developed a simple criterion for removing the network connections from the input units to the hidden units that does not affect the network's classification accuracy. A group of connections from the input units to a hidden unit can be removed at once if they satisfy this criterion. We expect simpler set of rules to be generated if more connections are removed. Finally, we rewrite all node splitting conditions in the decision tree in terms of the network inputs that are not removed. A node splitting condition can be rewritten as either DNF (disjunctive normal form) rules or MofN rules. DNF rules are expressed as a disjunction of conjunctions while MofN rules are expressed in the following form:

```
if {at least/exactly/at most} M of
  the N conditions (C1,C2,...,CN)
  are satisfied, then .....
```

In the following subsections, we describe the neural network training method and the tree generating algorithm.

2.1. *Neural Network Training*

Given an input pattern p, $p = 1, 2, \ldots, P$, the network's output unit value S_{ip} and hidden unit activation value H_{jp} are computed as follows:

$$S_{ip} = \sigma\left(\sum_{j=1}^{J} v_{ij} H_{jp}\right) \tag{1}$$

$$H_{jp} = \sigma(\mathbf{w}_j \mathbf{x}_p) = \sigma\left(\sum_{k=1}^{K} w_{jk} x_{kp}\right) \tag{2}$$

where $x_{kp} \in [0, 1]$ is the value of input unit k given pattern p, w_{jk} is the weight of the connection from input unit k to hidden unit j, v_{ij} is the weight of the connection from hidden unit j to output unit i, and $\sigma(\xi)$ is the sigmoid function $1/(1 + e^{-\xi})$. J and K are the number of hidden units and input units, respectively. Each pattern $\mathbf{x}_p$ belongs to one of the C possible classes $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_C$. The target value for pattern p at output unit i is denoted by t_{ip}. For binary classification problem, one output unit with binary encoding is used. For classification problems with $C > 2$ classes, C output units are used in the network. If pattern p belongs to class c, then $t_{cp} = 1$ and $t_{ip} = 0, \forall i \neq c$.

The network is trained to minimize the augmented cross-entropy error function

$$\theta(w, v) = F(w, v) - \sum_{i=1}^{C}\sum_{p=1}^{P}[t_{ip} \log S_{ip} + (1 - t_{ip}) \times \log(1 - S_{ip})]. \quad (3)$$

$P(w, v)$ is a penalty term

$$F(w, v) = \epsilon_1 \sum_{j=1}^{J}\left(\sum_{i=1}^{C}\frac{\beta v_{ij}^2}{1+\beta v_{ij}^2} + \sum_{k=1}^{K}\frac{\beta w_{jk}^2}{1+\beta w_{jk}^2}\right) + \epsilon_2 \sum_{j=1}^{J}\left(\sum_{i=1}^{C} v_{ij}^2 + \sum_{k=1}^{K} w_{jk}^2\right) \quad (4)$$

where ϵ_1, ϵ_2, and β are positive parameters. The cross-entropy error function has been shown to improve the convergence of network training over the standard least-squares error function [16], while the penalty function $F(w, v)$ is added to encourage weight decay [17]. Each network connection that has nonzero weight incurs a cost. By minimizing the augmented error function we expect those connections that are not useful for classifying the patterns to have small weights. Our experience with the augmented error function (3) shows that it is very effective in producing networks where relevant and irrelevant network connections can be distinguished by the magnitudes of their weights [13].

A version of the quasi-Newton algorithm, the BFGS method [18, 19], has been developed to minimize the error function (3). Compared to the standard backpropagation method, this method has been shown to converge much faster [20, 21].

2.2. An Illustrative Example

A network with 10 hidden units is trained to solve the Monk2 problem [15], which is an artificial problem of identifying the monks. The training data set consists of 169 patterns, each described by 6 discrete attributes as shown in Table 1.

The attribute values are coded in binary 0 or 1 format. Hence, the network requires 17 input units plus an additional input with a constant value of 1 for the hidden unit bias. An input pattern is classified as a monk if exactly two of its six attributes have their first values, i.e., if exactly two of $\{x_3, x_6, x_8, x_{11}, x_{15}, x_{17}\}$ equal 1. We trained a network to correctly classify all samples in the training data set and illustrate how the classification rule can be recovered from this network.

Table 1. The attributes of the Monk2 problem.

Attribute	Meaning	Possible values	Input units
A_1	head_shape	round, square, octagon	x_3, x_2, x_1
A_2	body_shape	round, square, octagon	x_6, x_5, x_4
A_3	is_smiling	yes, no	x_8, x_7
A_4	holding	sword, balloon, flag	x_{11}, x_{10}, x_9
A_5	jacket_color	red, yellow, green, blue	$x_{15}, x_{14}, x_{13}, x_{12}$
A_6	has_tie	yes, no	x_{17}, x_{16}

2.3. Identifying the Relevant Hidden Units

Once a neural network has been trained, its relevant hidden units are identified using information gain method. For this purpose, the C4.5 algorithm is employed. Given a data set D, the decision tree is generated recursively as follows:

1. If D contains one or more examples, all belonging to a single class C_c, stop.
2. If D contains no example, the most frequent class at the parent of this node is chosen as the class, stop. Or
3. If D contains examples belonging to a mixture of classes, information gain is used as a heuristic to split D into partitions (branches) based on the values of a single feature.

The decision tree is built using the hidden unit activations of training patterns that have been correctly classified by the neural network along with the patterns' class labels. These activation values are continuous in the interval $[0, 1]$ since each activation value has been computed as the sigmoid of the weighted inputs (2). Since the hidden activation of only correctly classified patterns are used to generate the decision tree, the number of patterns in D is usually less than the total number of patterns P in the training data set. Let $\bar{P}$ be the number of correctly classified patterns. For hidden unit j, its activation values H_{jp} in response to patterns p, $p = 1, 2, \ldots, \bar{P}$, are first sorted in increasing order. The values are then split into two groups $D_1 \equiv \{H_{j1}, \ldots, H_{jq}\}$ and $D_2 \equiv \{H_{j,q+1}, \ldots, H_{j\bar{P}}\}$ where $H_{jq} < H_{j,q+1}$, and the information gained by this split is computed. This information gain is computed for all possible splitting of the activation values (i.e., for q ranging from 1 to $\bar{P}$), and the maximum gain is taken as the information gain of hidden unit j.

Suppose that each pattern in the data set D belongs to one of the C classes, and n_c is the number of patterns

in class $\mathcal{C}_c$, the expected information for classification is

$$I(D) = -\sum_{c=1}^{C} \frac{n_c}{N} \log_2 \frac{n_c}{N}$$

where the number of patterns in the set D is $N = \sum_{c=1}^{C} n_c$. For the two subsets of D, the expected information is similarly computed:

$$I(D_1) = -\sum_{c=1}^{C} \frac{n_{c1}}{N_1} \log_2 \frac{n_{c1}}{N_1}$$

$$I(D_2) = -\sum_{c=1}^{C} \frac{n_{c2}}{N_2} \log_2 \frac{n_{c2}}{N_2}$$

where n_{cj} is the number of samples in D_j, $j = 1, 2$ that belong to class $\mathcal{C}_c$ and $N_j = \sum_{i=1}^{c} n_{cj}$. The information gained by splitting the data set D into D_1 and D_2 is

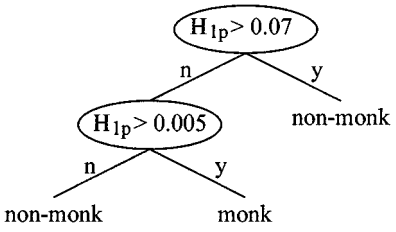
$$\text{Gain}(H_{jq}) = I(D) - [I(D_1) + I(D_2)]$$

and the normalized gain is

$$\text{NGain}(H_{jq}) = \text{Gain}(H_{jq}) \Bigg/ \left[-\sum_{j=1}^{2} (N_j/N) \log_2 (N_j/N) \right]$$

The root node of the decision tree contains a test condition which involves the hidden unit whose activation values give the highest normalized gain. The complete decision tree is generated by applying the same procedure to the subsets of the data at the two branches of a decision node. Once the decision tree has been constructed, the identification of the relevant hidden units is trivial. The hidden units whose activations are used in one or more nodes of the decision tree are the relevant units.

For the network that has been trained to solve the Monk2 problem, applying C4.5 on the hidden unit activation values gives the following decision tree:



This tree indicates that the monks and non-monks in the data set can be distinguished by the activation values of hidden unit 1 alone. The unit's activation values are split into two groups at the threshold value of 0.07. Sixty-four patterns produce activation values greater than 0.07. The values that are smaller than or equal to 0.07 are further split into two subgroups at the threshold value of 0.005. Forty-one patterns produce activation values less than or equal to 0.005 and the remaining 64 greater than 0.005.

3. Identifying the Relevant Input Connections

Because the network has been trained by minimizing an error function that has been augmented by a penalty term, network connections from irrelevant inputs can be expected to have small weights. For each hidden unit j, one or more of its connection weights w_{jk} from the input units may be sufficiently small that they can be removed without affecting the overall classification accuracy. The criterion for removing these irrelevant connections is given below.

Proposition 1. *Let the splitting condition for a node in the decision tree be $H_{jp} \leq H_{jt}$ for some t. Define $L \equiv H_{jt}$, $U \equiv H_{j,t+1}$ (the smallest activation value that is larger than L), $D_L \equiv \{\mathbf{x}_p \mid H_{jp} = \sigma(\mathbf{w}_j \mathbf{x}_p) \leq L\}$, and $D_U \equiv \{\mathbf{x}_p \mid H_{jp} = \sigma(\mathbf{w}_j \mathbf{x}_p) \geq U\}$. Let S be the set of input units whose connections to hidden unit j satisfy the following condition*:

$$\sum_{k \in S} |w_{jk}| < 2(U - L) \tag{5}$$

and S' be the complement of S. Then, by changing the splitting condition to

$$H_{jp} \leq (L + U)/2, \tag{6}$$

the connections from the units in S to hidden unit j can be removed without changing the membership of D_L and D_U.

Proof: For notational convenience, let us denote

$$\Psi = \sum_{k \in S'} w_{jk} x_{kp} \quad \psi = \sum_{k \in S} w_{jk} x_{kp}$$

Consider the following 2 cases:

Case 1. $\mathbf{x}_p \in D_L$, i.e., for this sample $H_{jp} = \sigma(\mathbf{w}_j \mathbf{x}_p) = \sigma(\Psi + \psi) \leq L$.

By Taylor's theorem,

$$\sigma(\Psi) = \sigma(\Psi + \psi) - \psi\sigma'(\xi)$$

where ξ is a point lying between Ψ and $\Psi + \psi$. The derivative of the sigmoid function is always positive and less than or equal to $1/4$. Consider these two cases:

1. $\psi \leq 0$: $\sigma(\Psi) \leq L - \frac{1}{4}\psi \leq L + \frac{1}{4}\sum_{k\in S}|w_{jk}| < (L+U)/2$
2. $\psi > 0$: $\sigma(\Psi) \leq \sigma(\Psi + \psi) \leq L < (L+U)/2$

Case 2. $\mathbf{x}_p \in D_U$, i.e. for this sample $H_{jp} = \sigma(\mathbf{w}_j\mathbf{x}_p) = \sigma(\Psi + \psi) \geq U$. Consider the following cases:

1. $\psi \leq 0$: $\sigma(\Psi) \geq \sigma(\Psi + \psi) \geq U > (L+U)/2$
2. $\psi > 0$: $\sigma(\Psi) \geq U - \frac{1}{4}\psi \geq U - \frac{1}{4}\sum_{k\in S}|w_{jk}| > (L+U)/2$

Therefore, it can be concluded that after changing the node splitting condition to (6), the input-to-hidden unit connections whose weights satisfy (5) can be removed without affecting the membership of the sets D_L and D_U. □

Proposition 2. *Let the splitting condition for a node in the decision tree be $H_{jp} > H_{jt}$ for some t. Define $L \equiv H_{jt}$, $U \equiv H_{j,t+1}$ (the smallest activation value that is larger than L), $D_L \equiv \{\mathbf{x}_p \mid H_{jp} = \sigma(\mathbf{w}_j\mathbf{x}_p) \leq L\}$, and $D_U \equiv \{\mathbf{x}_p \mid H_{jp} = \sigma(\mathbf{w}_j\mathbf{x}_p) \geq U\}$. Let S be the set of input units whose connections to hidden unit j satisfy the following condition*:

$$\sum_{k\in S}|w_{jk}| \leq 2(U - L) \tag{7}$$

and S' be the complement of S. Then, by changing the splitting condition to

$$H_{jp} > (L+U)/2\,, \tag{8}$$

the connections from the units in S to hidden unit j can be removed without changing the membership of D_L and D_U.

Proof: The proof is similar to that of Proposition 1 and is omitted.

In the decision tree we obtained for the Monk2 problem, there are 2 tests involving H_1. The values of L are 0.07 and 0.005 and the corresponding values of U are 0.79 and 0.04, respectively.

- Test 1. The new threshold value is $(0.07+0.79)/2 = 0.43$. The weights of the connections from the input units to the hidden unit are sorted in increasing order of their absolute values. The first 11 weights sum up to 0.036 which is smaller than the threshold for weight removal (7) of $2(0.79 - 0.07) = 1.44$. Therefore, these weights can be removed.
- Test 2. The new threshold value is $(0.005 + 0.04)/2 = 0.023$. The same 11 weights as in Test 1 total less than $2(0.04 - 0.005) = 0.07$ and are removed as they satisfy condition (5).

After removing redundant connections, the node splitting condition in the decision tree can be written as follows (the pattern subscript p is not shown):

If $\sigma(4.6x_3 + 4.2x_6 - 4.5x_7 + 4.4x_{11} + 4.6x_{15} - 4.6x_{16} - 2.6) > 0.43$, then Class 0 (not monk),
else if $\sigma(4.6x_3 + 4.2x_6 - 4.5x_7 + 4.4x_{11} + 4.6x_{15} - 4.6x_{16} - 2.6) <= 0.023$, then Class 0 (not monk),
else if $\sigma(4.6x_3 + 4.2x_6 - 4.5x_7 + 4.4x_{11} + 4.6x_{15} - 4.6x_{16} - 2.6) > 0.023$, then Class 1 (monk). □

4. Rule Generation

By computing the inverse of the sigmoid function $\sigma^{-1}((L+U)/2)$ for all node splitting conditions in a decision tree, we obtain conditions that are linear combinations of the input attributes of the data. For a data set with continuous attributes, such *oblique* decision rules are appropriate for classifying the patterns. For a data set with discrete attributes, however, it may be desirable to go one step further and extract an equivalent set of symbolic rules. This is done by replacing each node splitting condition by an equivalent set of DNF or MofN rules.

MofN rule can be more concise than DNF rules and it is more general. The conjunction $x_1 \wedge x_2 \cdots \wedge x_K$ is equivalent to *exactly K of* $\{x_1, x_2, \ldots, x_K\}$, while the disjunction $x_1 \vee x_2 \cdots \vee x_K$ is equivalent to *at least* 1 *of*

$\{x_1, x_2, \ldots, x_K\}$. FERNN always attempts to extract an MofN rule first before resorting to generating a DNF rule. MofN rule can be generated by applying one of the following two propositions.

Proposition 3. *Let the node splitting condition be*

$$w_1x_1 + w_2x_2 + \cdots + w_Kx_K \leq U. \tag{9}$$

Define $\lfloor U \rfloor$ *to be the largest integer that is less than or equal to* U *and define* $F = U - \lfloor U \rfloor$ *and suppose that the following assumptions hold*:

1. *The weights* w_i *are positive and can be expressed as* $w_i = 1 + f_i$ *with* $f_i \in [0, 1)$, *i.e.* $w_i \in [1, 2)$.
2. *The possible values for* x_i *are either* 0 *or* 1.
3. *The sum of the* $\lfloor U \rfloor$ *largest* $f_i, i = 1, 2, \ldots, K$, *is less than or equal to* F.

We can then replace condition (9) *by the equivalent rule*

if at most $\lfloor U \rfloor$ *of* $\{x_1, x_2, \ldots x_K\}$ *equal* 1.

Proof:

1. Suppose the three assumptions hold and that at most $\lfloor U \rfloor$ of $x_1, x_2, \ldots x_K$ equal 1, then

$$\sum_{i=1}^{K} w_i x_i = \sum_{i=1}^{K} (1 + f_i)x_i \leq \lfloor U \rfloor + F = U$$

2. If the three assumptions hold and condition 9 is satisfied by $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_K$, we claim that at most $\lfloor U \rfloor$ of these K inputs have value of 1. Suppose $\lfloor U \rfloor + \mu$, $(\mu \geq 1)$ of the inputs have value of 1, then

$$\begin{aligned}\sum_{i=1}^{K} w_i \hat{x}_i &= \sum_{i=1}^{K} (1 + f_i)\hat{x}_i \\ &= \lfloor U \rfloor + \mu + \sum_{i=1}^{K} f_i \hat{x}_i \\ &\geq \lfloor U \rfloor + \mu > \lfloor U \rfloor + F = U\end{aligned}$$

which is a contradiction. □

Proposition 4. *Let the node splitting condition be*

$$w_1x_1 + w_2x_2 + \cdots w_Kx_K > L. \tag{10}$$

Define $\lceil L \rceil$ *to be the smallest integer that is larger than or equal to* L *and define* $F = \lceil L \rceil - L$ *and suppose that the following conditions hold*:

1. *The weights* w_i *are positive and can be expressed as* $w_i = 1 + f_i$ *with* $f_i \in [0, 1)$, *i.e.* $w_i \in [1, 2)$.
2. *The possible values for* x_i *are either* 0 *or* 1.
3. *The sum of* $\lceil L \rceil$ *largest* $f_i, i = 1, 2, \ldots, K$ *is less than or equal to* $1 - F$.

We can then replace condition (10) *by the equivalent rule*

if at least $\lceil L \rceil$ *of* $\{x_1, x_2, \ldots x_K\}$ *equal* 1.

Proof: The proof is similar to that of Proposition 3 and is omitted. □

The relevant weights of a trained network can have any real values. Before we test if they satisfy the conditions in Propositions 3 and 4, the following steps can be performed to improve the chances of satisfying them:

- remove negative weights. This may be possible by replacing the corresponding inputs x_i's with their complements. For example, suppose the attribute $\mathcal{A}$ has 2 discrete values $\{a_1, a_2\}$ and 2 binary inputs (x_1, x_2) have been used to represent them: $\mathcal{A} = a_1 \Leftrightarrow (x_1, x_2) = (1, 0)$ and $\mathcal{A} = a_2 \Leftrightarrow (x_1, x_2) = (0, 1)$. If w_1 is negative, then we can replace x_1 by its complement which is x_2:

$$w_1x_1 = w_1(1 - x_2) = -w_1x_2 + w_1$$

- divide all the weights by the smallest w_i.

For the Monk2 example, an MofN rule can be generated as follows. A sample pattern is classified as a monk if and only if the condition

$$\begin{aligned}0.023 < \sigma(&4.6x_3 + 4.2x_6 - 4.5x_7 + 4.4x_{11} + 4.6x_{15} \\ &- 4.6x_{16} - 2.6) \leq 0.43\end{aligned}$$

is satisfied. The weights of x_7 and x_{16} are negative. Since both attributes have only 2 possible values 0 or 1, the negative weights can be removed by substituting x_7 and x_{16} by their complements: $x_7 = 1 - x_8$ and $x_{16} = 1 - x_{17}$. These substitutions and the computations of $\sigma^{-1}(0.023) = -3.75$ and $\sigma^{-1}(0.43) = -0.28$ lead to a simplified rule:

$$\begin{aligned}1.9 < 1.1x_3 + x_6 + 1.1x_8 + x_{11} + 1.1x_{15} \\ + 1.1x_{17} \leq 2.7\end{aligned} \tag{11}$$

The above condition can be transformed into the MofN rule:

> if 2 of the 6 inputs $\{x_3, x_6, x_8, x_{11}, x_{15}, x_{17}\}$ are 1 then monk,

by applying Propositions 3 and 4.

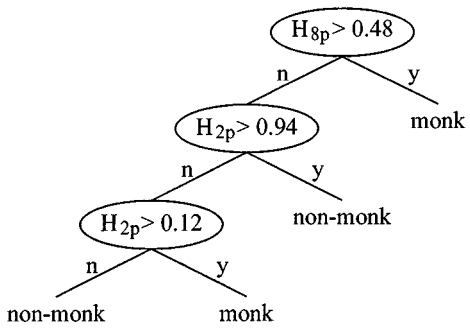
When it is not possible to express a node splitting condition as an MofN rule, FERNN generates a DNF rule. A general purpose algorithm X2R [22] is used to automate the DNF rule generation process. X2R takes as input a set of discrete patterns and their corresponding target outputs and produces a set of classification rules with 100% accuracy if there are no identical patterns with different class labels. The next section contains illustrations of how such rules are extracted.

5. Experiments with the Monk1 and Monk3 Problems

In addition to the Monk2 problem, Thrun et al. [15] also created two other classification problems with the same input attributes. These are the Monk1 and Monk3 problems. The input attributes for these problems are the same as those for Monk2 problem listed in Table 1. In this section, we illustrate how FERNN extracts classification rules for these problems.

5.1. The Monk1 Problem

A pattern is classified as a monk in this problem if (head_shape = body_shape) or if (jacket_color = red). FERNN extracts rules from a network that has been trained to correctly classify all its 216 input patterns. The original network had 10 units in the hidden layer. The following C4.5 tree that was generated indicates that hidden units 2 and 8 are relevant:



- There are 3 node splits and the relevant values of L_1, L_2, and L_3 are 0.48, 0.94, and 0.12. The corresponding values of U_1, U_2, and U_3 are 0.94, 1.00, and 0.90, respectively. The new threshold values for the splits are 0.71, 0.97, and 0.51, respectively.
- For hidden unit 8, we remove connections whose absolute weights total less than $2(U-L) = 2(0.94-0.48) = 0.92$. As a result, only x_{15} remains. The test $H_8 > 0.48$ is replaced by $\sigma(2.9x_{15}) > (0.48 + 0.94)/2 = 0.71$, or equivalently $x_{15} > 0.30$. This test is satisfied only when $x_{15} = 1$, that is, when jacket_color = red
- For hidden unit 2, after removing the irrelevant weights, we have a pattern classified as a monk if

$$0.51 < \sigma(2.2x_1 - 2.1x_2 + 6.8x_3 + 4.8x_5 - 4.3x_6) \leq 0.97.$$

  After computing the values of $\sigma^{-1}(0.51)$ and $\sigma^{-1}(0.97)$ and letting $x_1 = 1 - x_2 - x_3$, the above condition can be simplified to

$$-0.5 < -x_2 + x_3 + 1.1x_5 - x_6 \leq 0.3.$$

  The values of (x_2, x_3, x_5, x_6) that satisfy this condition can be given in terms of an MofN rule if the negative weights are removed by substituting $x_2 = 1 - \bar{x}_2$ and $x_6 = 1 - \bar{x}_6$ to obtain

$$1.5 < \bar{x}_2 + x_3 + 1.1x_5 + \bar{x}_6 \leq 2.3.$$

  Hence, by Propositions 3 and 4, a pattern is a monk if exactly 2 of $\{\bar{x}_2, x_3, x_5, \bar{x}_6\}$ are 1. The three sets of values of $\{x_2, x_3, x_5, x_6\}$ that satisfy this condition are listed in Table 2.
- We conclude that a pattern is classified as a monk if (jacket_color = red) or (head_shape = body_shape)

5.2. The Monk3 Problem

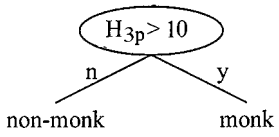
A pattern is classified as monk in this problem if (jacket_color = green and holding = sword) or (jacket_color ≠ blue and body_shape ≠ octagon). FERNN generates a tree with a total of 3 nodes, the root node and 2 child nodes. This indicates that the patterns in the data set of this problem are linearly separable, i.e., there exists a hyperplane such that all the monks are on one side of it and the non-monks on the other side. For problems with linearly separable patterns, the use of C4.5 to identify the relevant hidden

Table 2. The values of $\{x_2, x_3, x_5, x_6\}$ that satisfy the node splitting condition of hidden unit 2 for a pattern to be classified as monk and their equivalent attribute values in the original data set.

(x_2, x_3, x_5, x_6)	$(\bar{x}_2, x_3, x_5, \bar{x}_6)$	Meaning
(0,0,0,0)	(1,0,0,1)	head_shape = octagon, body_shape = octagon
(1,0,1,0)	(0,0,1,1)	head_shape = square, body_shape = square
(0,1,0,1)	(1,1,0,0)	head_shape = round, body_shape = round

units quickly reduces the number of hidden units J from the initial value of 10 to 1.

An example of a decision tree that is generated is as follows:



FERNN removes the irrelevant connections to the hidden unit 3 of the network and obtains

If $\sigma(-5.2x_4 + 2.8x_{11} - 5.2x_{12} + 2.6x_{13}) < 0.34$,
then Class 0 (not monk),
else if $\sigma(-5.2x_4 + 2.8x_{11} - 5.2x_{12} + 2.6x_{13}) > 0.34$,
then Class 1 (monk).

Twelve[1] different combinations of $\{x_4, x_{11}, x_{12}, x_{13}\}$ are possible and they are all represented in the training data set. X2R takes as input these 12 different combinations along with the corresponding class labels and outputs the following table where the symbol "×" is used to indicate "don't care" value.

Rule	x_4	x_{11}	x_{12}	x_{13}	monk?
0	0	×	0	×	yes
1	×	×	1	×	no
2	×	1	0	1	yes
3	1	×	×	0	no
4	1	0	×	×	no

Collecting the rules for monk, FERNN obtains

Rule0: If $x_4 = x_{12} = 0$, then monk.
Rule2: If $x_{11} = x_{13} = 1$ and $x_{12} = 0$, then monk.

In terms of the original attributes, the equivalent rules are

Rule0: If (body_shape $\neq$ octagon) and (jacket_color $\neq$ blue) then monk.
Rule2: If (holding = sword) and (jacket_color = green) then monk.

6. Experimental Results

The effectiveness of FERNN has been tested on 15 problems listed in Table 3. The data sets were obtained from the UCI repository [23] or generated according to the function definitions given by Vilalta et al. [24]. Each data set was randomly divided into three subsets: the training set (40%), the cross validation set (10%) and the test set (50%). For all experiments, the initial number of hidden units in the network was 10. The parameters of the penalty function (3) were fixed at $\epsilon_1 = 1$, $\epsilon_2 = 0.01$, and $\beta = 5$. These values were obtained after empirical studies were performed.

Table 3 compares the tree size (i.e., the number of nodes) and the predictive accuracy of the decision trees generated by FERNN with those generated by C4.5 and the combined algorithm N2P2F+C4.5. For the combined algorithm, N2P2F [13] stopped pruning when the accuracy of the network on the cross validation set dropped by more than 1% from its best value. C4.5 was then used to generate decision trees using the activation values of the pruned networks. FERNN and C4.5 do not require the cross validation set, hence, 50% of the data was used for each training session. The tree size and predictive accuracy for FERNN and N2P2F+C4.5 are averaged over 20 test runs.

The decision trees extracted by N2P2F+C4.5 and FERNN are smaller than those generated by C4.5. This is expected since the decision nodes of the trees generated by N2P2F+C4.5 and FERNN usually involve several original attributes of the data. In contrast, the decision nodes of the trees generated by C4.5 involve only individual attributes. Using multiple attributes in the decision nodes may result in simpler rules, such as the MofN rules for the Monk2 and MAJ12 problems. Multi-attribute decision nodes may also improve the

Table 3. Comparison of various algorithms. P = number of samples, K = number of attributes, a = predictive accuracies, and N = tree size.

			C4.5		N2P2F+C4.5		FERNN	
Dataset	P	K	a	N	a	N	a	N
Monk1	432	17	100.00	15	99.98	10.20	99.89	10.50
Monk2	432	17	75.46	45	100.00	5.00	99.77	5.10
Monk3	432	17	100.00	9	99.98	3.20	99.88	3.00
CNF12a	4096	12	100.00	41	100.00	7.40	100.00	7.70
CNF12b	4096	12	96.97	87	99.96	30.70	99.94	26.60
DNF12a	4096	12	100.00	33	100.00	11.40	99.99	10.20
DNF12b	4096	12	100.00	43	99.62	19.60	99.99	12.20
MAJ12a	4096	12	91.85	63	99.97	3.00	99.99	3.00
MAJ12b	4096	12	81.49	195	100.00	3.00	100.00	3.00
MUX12	4096	12	100.00	147	99.93	25.50	99.70	28.20
Australian	690	15	83.19	36	83.48	7.10	83.93	10.30
BCancer	699	10	93.70	13	94.14	5.90	95.10	6.80
HeartD	297	13	73.65	32	82.30	10.60	82.70	11.30
Pima	768	8	71.09	75	72.64	17.80	72.75	17.30
Sonar	208	60	79.81	13	81.83	8.80	83.51	9.50

accuracy of the tree because samples from real world problems may be better separated by oblique hyperplanes. This is the case with the heart disease data set (HeartD in Table 3) where significant improvement is achieved by the neural network methods over C4.5.

There is no significant difference in the accuracy and size of the decision trees generated by FERNN and N2P2F+C4.5. The test results suggest that FERNN can save substantial amount of retraining time by using C4.5 to identify the relevant hidden units of the unpruned networks without sacrificing the predictive accuracy and increasing the size of the decision trees.

7. Discussion and Conclusion

The paper by Andrews et al. [1] includes a comprehensive list of papers that discuss methods for extracting rules from neural networks. We compare FERNN with some of the works listed along the following 3 dimensions:

1. *The expressive power of the extracted rules.* Some methods search for specific type of rules. For example, MofN algorithm [8] and GDS algorithm [9] extract MofN rules. BRAINNE [25], RX [10] and NeuroRule [5] generate DNF rules. RX and NeuroRule require that the continuous attributes of a data set be discretized before neural networks are trained. NeuroLinear [6] extracts oblique decision rules for data set with continuous attributes. The present work is more general. For data sets with continuous attributes, it extracts oblique decision rules. For data sets with discrete attributes, the process of rule extraction is continued with possible extraction of either MofN rules or DNF rules. We provide the conditions for which an MofN rule can be extracted. When these conditions are not met, X2R is applied to extract DNF rules.
2. *The quality of the extracted rules in terms of their accuracy, fidelity, and comprehensibility*. It is hard to make a direct comparison between FERNN and other methods as published works include results obtained from only a small number of data sets. Any good neural network rule extraction algorithm, however, can be expected to extract rules with similar classification and predictive accuracies as the networks themselves. Our experimental results indicate that FERNN has a high degree of fidelity, that is, the rules that it extracts achieved the same training accuracy and similar test accuracy as the networks. We also compare FERNN's performance with that of a method that extracts rules from pruned networks (N2P2F+C4.5) and find that there is no significant

difference in the predictive accuracy and the size of the decision trees generated by both methods.

3. *The extent to which the underlying neural networks incorporate specialized training methods*. Like most other methods, FERNN requires that the networks be trained to minimize an augmented penalty function. A penalty or weight decay term is added to the error function to encourage some weights to go to zero. Network connections with weights that are close to zero are irrelevant and can be removed without jeopardizing the network's accuracy. In FERNN, however, the choice of penalty function is very crucial since weights are set to zero based on their magnitudes and no network retraining is performed after the connections are removed. Other than this careful selection of the penalty function and its parameter values, FERNN does not require specialized network training algorithm. This is in contrast to a method that requires the weights to be of certain magnitudes [9], methods that require that all the units in the networks be binary valued [3, 8], and a method that requires the output units to be replicated in the input layer [25].

To conclude, in this paper we have presented FERNN, a fast algorithm for extracting symbolic classification rules from neural networks. Our experimental results show that even though the algorithm does not perform network retraining after identifying the relevant hidden units and connections, the decision trees that it generates are comparable in terms of predictive accuracy and tree size to those generated by another method which requires network pruning and retraining. The algorithm employs C4.5 to generate a decision tree using the hidden unit activations as inputs. For a data set with discrete attributes, the node splitting conditions in the tree can be replaced by their equivalent symbolic rules after irrelevant input connections are removed. Simple criteria for identifying such connections are given. The criteria ensure that the removal of these connections will not affect the classification accuracy of the tree.

Acknowledgments

This research is supported in part by NUS Academic Research Grant RP950656. The authors wish to thank their colleague, Dr. Huan Liu for making his X2R code available to them and an anonymous reviewer for the valuable suggestions and comments given.

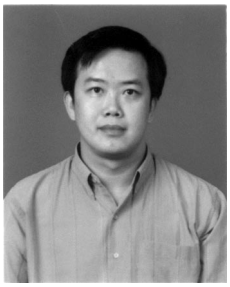
Note

1. Two for x_4, two for x_{11} and three for (x_{12}, x_{13}).

References

1. R. Andrews, J. Diederich, and A. Tickle, "Survey and critique of techniques for extracting rules from trained artificial neural networks," *Knowledge Based Systems*, vol. 8, no. 6, pp. 373–389.
2. G.A. Carpenter and A.-H. Tan, "Rule extraction: From neural architecture to symbolic representation," *Connection Science*, vol. 7, no. 1, pp. 3–28, 1995.
3. L.M. Fu, "Rule learning by searching on adapted nets," in *Proc. of the 9th National Conference on Artificial Intelligence*, AAAI Press/MIT Press: Menlo Park, CA, 1991, pp. 590–595.
4. S. Gallant, "Connectionist expert systems," *Communications of the ACM*, vol. 4, pp. 152–169, 1988.
5. R. Setiono and H. Liu, "Symbolic representation of neural networks," *IEEE Computer*, vol. 29, no. 3, pp. 71–77, 1996.
6. R. Setiono and H. Liu, "NeuroLinear: From neural networks to oblique decision rules," *Neurocomputing*, vol. 17, pp. 1–24, 1997.
7. S.B. Thrun, "Extracting rules from artificial neural networks with distributed representations," in *Advances in Neural Information Processing* 7, in edited by G. Tesauro, D. Touretzky and T. Leen, Morgan Kaufmann, 1995.
8. G.G. Towell and J.W. Shavlik, "Extracting refined rules from knowledge-based neural networks," *Machine Learning*, vol. 13, no. 1, pp. 71–101, 1993.
9. R. Blassig, "GDS: Gradient descent generation of symbolic rules," in *Advances in Neural Info. Proc. Systems 6*, Morgan Kaufmann: San Mateo, CA, 1994, pp. 1093–1100.
10. R. Setiono, "Extracting rules from neural networks by pruning and hidden-unit splitting," *Neural Computation*, vol. 9, no. 1, pp. 205–225, 1997.
11. B. Hassibi and D.G. Stork, "Second order derivatives for network pruning: optimal brain surgeon," in *Advances in Neural Information Processing Systems* 5, San Mateo, Morgan Kaufmann: CA, 1993, pp. 164–171.
12. M. Hagiwara, "A simple and effective method for removal of hidden units and weights," *Neurocomputing*, vol. 6, pp. 207–218, 1994.
13. R. Setiono, "A penalty function approach for pruning feedforward neural networks," *Neural Computation*, vol. 9, no. 1, pp. 185–204, 1997.
14. G. Castellano, A.M. Fanelli, and M. Pelilo, "An iterative pruning algorithm for feedforward neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 519–531, 1997.
15. S.B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dẑeroski, S.E. Fahlman, D. Fisher, R. Hamann, K. Kaufmann, S. Keller, I. Kononenko, J. Kreuziger, R.S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich, H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang, "The MONK's problems—a performance comparison of different learning algorithm," Preprint CMU-CS-91-197, Carnegie Mellon University, Pittsburgh, PA, 1991.
16. A. van Ooyen, A. and B. Nienhuis, "Improving the convergence of the backpropagation algorithm," *Neural Networks*, vol. 5, no. 3, pp. 465–471, 1992.

17. J. Hertz, A. Krogh, and R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison Wesley: Redwood City, CA, 1991.
18. R. Battiti, "First- and second-order methods for learning: Between steepest descent and Newton's method," *Neural Computation*, vol. 4, pp. 141–166, 1992.
19. J.E. Dennis, Jr. and R.B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice Hall: Englewood Cliffs, NJ, 1983.
20. R. Setiono, "A neural network construction algorithm which maximizes the likelihood function," *Connection Science*, vol. 7, no. 2, pp. 147–166, 1995.
21. R.L. Watrous, "Learning algorithms for connectionist networks: Applied gradient methods for nonlinear optimization," in *Proc. IEEE 1st Int. Conf. Neural Networks*, San Diego, CA, 1987, pp. 619–627.
22. H. Liu and S.T. Tan, "X2R: A fast rule generator," in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, IEEE Press: New York, 1995, pp. 1631–1635.
23. C. Merz and P. Murphy, "UCI repository of machine learning databases," `http://www.ics.uci.edu/~mlearn/MLRepository.html`, Dept. of Info. and Comp. Sci., University of California: Irvine, CA, 1996.
24. R. Vilalta, G. Blix, and L. Rendell, "Global data analysis and the fragmentation problem in decision tree induction," *Machine Learning: ECML-97*, edited by M. van Someren and G. Widmer, Springer-Verlag, 1997, pp. 312–326.
25. S. Sestito and T. Dillon, *Automated Knowledge Acquisition*, Prentice Hall: Sydney, Australia, 1994.
26. J.R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann: San Mateo, CA, 1993.



Rudy Setiono received the B.S. degree in Computer Science from Eastern Michigan University in 1984, the M.S. and Ph.D. degrees in Computer Science from the University of Wisconsin-Madison in 1986 and 1990 respectively. Since August 1990, he has been with the School of Computing at the National University of Singapore where he is currently an Associate Professor. His research interests include unconstrained optimization, neural network construction and pruning, and rule extraction from neural networks.

Wee Kheng Leow received the B.Sc. and M.Sc. degrees in Computer Science from the National University of Singapore in 1985 and 1989, respectively. He received the Ph.D. degree in Computer Science from the University of Texas at Austin in1994. He is now an Assistant Professor at the School of Computing, National University of Singapore and a member of INNS and IEEE. His research interests include neural modeling and application of neural networks to problem solving.