Extracting M-of-N Rules from Trained Neural Networks

Rudy Setiono

School of Computing

National University of Singapore

Kent Ridge, Singapore 119260

Republic of Singapore

Email: rudys@comp.nus.edu.sg

Abstract

An effective algorithm for extracting M-of-N rules from trained feedforward neural networks is proposed. Two components of the algorithm distinguish our method from previously proposed algorithms which extract symbolic rules from neural networks. First, we train a network where each input of the data can only have one of the two possible values, -1 or 1. Second, we apply the hyperbolic tangent function to each connection from the input layer to the hidden layer of the network. By applying this squashing function, the activation values at the hidden units are effectively computed as the hyperbolic tangent (or the sigmoid) of the weighted inputs, where the weights have magnitudes that are equal one. By restricting the inputs and the weights to binary values either -1 or 1, the extraction of M-of-N rules from the networks becomes trivial. We demonstrate the effectiveness of the proposed algorithm on several widely tested datasets. For datasets consisting of thousands of patterns with many attributes, the rules extracted by the algorithm are surprisingly simple and accurate.

I. INTRODUCTION

Feedforward neural networks have been successfully used as a tool for classification in a variety of real-world applications. In some of these applications, it may be desirable to have a set of rules that explains the classification process of a trained network. The classification concept represented as rules is certainly more comprehensible to a human user than a collection of network weights.

There is quite a lot of literature on algorithms that extract rules from trained neural networks available [1]. One of the more recent rule extraction algorithms is NeuroRule [2]. This algorithm extracts symbolic classification rules from a pruned network with a single hidden layer in two steps. First, rules that explain the network outputs are generated in terms of the discretized activation values of the hidden units. Second, rules that explain the discretized hidden unit activation values are generated in terms of the network inputs. When these two sets of rules are merged, a DNF representation of the network classification is obtained. Under DNF representation, the classification concept is expressed as the disjunction of one or more subconcepts.

Another method that extracts DNF rules is the Subset method [3]. It generates rules from a trained network by searching a subset of weights going to a unit that exceeds the bias of that unit. The search is started by having a single connection in the subset. If this connection weight exceeds the unit bias, a rule is generated. Otherwise, the size of the subset is increased by including more network connections.

There are classification problems for which DNF representations are not suitable. Among synthetic problems that have often been used to test neural network and decision tree methods are the N-bit parity problems. The smallest of the N-bit parity problems is the 2-bit parity problem or the XOR problem. The rule set extracted by NeuroRule is as follows:

```
If (input 1 is true) and (input 2 is not true) then odd parity,
else if (input 1 is not true) and (input 2 is true) then odd parity,
else even parity.
```

The above classification can be more succinctly expressed as M-of-N rules. The M-of-N rules are classification rules of the form [3]:

```
If (M of the following N antecedents are true)
    then ......
```

Hence, the XOR classification can be expressed as

```
If (exactly 1 of the 2 inputs is true) then odd parity,
else even parity.
```

The MofN method of Towell and Shavlik [3] is an improvement of the Subset method that is designed to explicitly search for M-of-N rules from knowledge based neural networks. Instead of considering an individual network connection, groups of connections are checked for their contribution to a unit's activation. This is done by clustering the network connections. The weights of the connections in a cluster are then replaced by their average weight. Clusters with small average and a few connections are checked for possible elimination since their removal are not likely to have any effect on the network classification. A rule is formed for each hidden and output unit. This rule consists of a threshold and the weighted antecedents of the remaining connections.

This paper presents MofN3, a new method for extracting M-of-N rules from trained neural networks. The topology of the neural networks is the standard three-layered feedforward networks. Units in the input layer are connected only to units in the hidden layer, while units in the hidden layer are also connected to units in the output layer. Given a hidden unit of a trained neural network with N incoming connections, we show how the value of M can be easily computed. In order to facilitate the process of extracting M-of-N rules, we assume that the attributes of the dataset have binary values -1 or 1.

The outline of the paper is as follows. In Section II, we describe our neural network training and pruning algorithm. In Section III, we explain our M-of-N rule extraction algorithm. The idea behind our rule extraction method is presented in detail in this section. In Section IV, we describe how M-of-N rules can be extracted from pruned networks by giving two illustrative examples. The splice junction determination problem [3] is selected for this purpose. It is a nontrivial real-world problem in molecular biology with thousands of samples and a large number of attributes. The rules extracted by the algorithm are surprisingly simple and their accuracy rates are comparable to those of other neural network rule extraction algorithms that have been reported in the literature. We report our experimental results in in Section V and conclude in Section VI.

II. NEURAL NETWORK TRAINING AND PRUNING

We first define our notation:

- P is the number of patterns in the training dataset.
- N is the number of units in the input layer. It is also the dimensionality of the patterns.
- H is the number of units in the hidden layer.
- C is the number of units in the output layer. It is also the number of classes in the dataset, except for binary classification problem when we set $C = 1$.
- $\mathbf{x_p}$ is an N-dimensional input pattern, $p = 1, 2, \ldots, P$, its n-th component is denoted by x_{pn}. The target output for $\mathbf{x_p}$ is a C dimensional vector $\mathbf{t_p}$. The target at output unit c is denoted by t_{pc}, $c = 1, 2, \ldots, C$.
- The weight of the connection from the n-th input unit to the h-th hidden unit is denoted by w_{nh}, while the weight of the connection from the h-th hidden unit to the c-th output unit is denoted by v_{hc}.
- For input pattern $\mathbf{x_p}$, β_{pc} is the output of the network at output unit c:

$$\beta_{pc} = \sigma\left(\sum_{h=1}^{H}(\alpha_{ph}v_{hc}) + \eta_c\right), \tag{1}$$

  where η_c is the bias of output unit c and α_{ph} is hidden unit h's activation value of pattern $\mathbf{x_p}$:

$$\alpha_{ph} = \tanh\left(K_0, \left(\sum_{n=1}^{N} x_{pn} \times \tanh(K_1, w_{nh})\right) + \tau_h\right) \tag{2}$$

The function $\sigma(x)$ is the sigmoid function $\sigma(x) = 1/(1+e^{-x})$ and τ_h is the bias of hidden unit h.

A crucial factor underlying our rule extraction algorithm is the application of the hyperbolic tangent function

$$\tanh(K,x) = (e^{Kx} - e^{-Kx})/(e^{Kx} + e^{-Kx}), \quad K > 0 \tag{3}$$

to each connection from an input unit to a hidden unit in (2). The activation of unit h in the hidden layer is normally computed as some function f of the total weighted inputs going into this unit:

$$\alpha_{ph} = f(\sum_{n=1}^{N} w_{nh} x_{pn} + \tau_h)$$

where f is usually the sigmoid or the hyperbolic tangent function. If K_1 in (2) is large, applying the hyperbolic function to all connections to the hidden unit implies that the activation value is effectively computed as the hyperbolic tangent of the weighted inputs, where the weights have magnitudes near one. Connections with small magnitudes are removed from the network by pruning. The process of extracting M-of-N rules from a trained network is greatly simplified by restricting both the inputs and the weights to have magnitudes that are equal one.

The error function that we minimize during network training is the following function:

$$F(w,v) = -\sum_{p=1}^{P}\sum_{c=1}^{C}(t_{pc}\log\beta_{pc} + (1-t_{pc})\log(1-\beta_{pc})) + P(w,v). \tag{4}$$

It is the cross-entropy error measure augmented by the penalty term

$$P(w,v) = \epsilon \sum_{h=1}^{H}\left(\sum_{n=1}^{N}\frac{\tanh(K_1,w_{nh})^2}{1+\tanh(K_1,w_{nh})^2} + \sum_{c=1}^{C} v_{hc}^2\right) \tag{5}$$

where ϵ is a positive penalty parameter. The cross-entropy error function has been shown to improve the convergence of network training compared to the standard least-squares error function [4].

The weights of the connections from the input layer to the hidden layer are penalized so that smaller weights decay more rapidly than larger ones [5]. The weights of the connections from the hidden layer to the output layer are given a quadratic penalty to prevent these weights from getting too large as the pruning algorithm removes redundant connections based on the magnitudes of their weights. Connections with sufficiently small weights can be removed without affecting the classification accuracy of the network. After these connections are removed, the network is retrained. The process is repeated by checking if there are any connections in the retrained network that meet the criteria for removal. The criteria for removing network connections and other details of the pruning algorithm can be found in our earlier work [6].

The iterative process of removing connections and retraining the network can be very time consuming, especially for problems that have many attributes and/or require a large number of hidden units. It is therefore imperative that we have an algorithm that trains faster than the standard backpropagation method. In our implementation, a version of the quasi-Newton algorithm, the BFGS method [7, 8] is used to minimize the function (4). This method has been shown to be very effective for neural network training [9, 10]. At each iteration of the BFGS method, a positive definite matrix which approximates the inverse of the Hessian of this function is computed. A descent direction is then obtained by multiplying this matrix by the negative of the gradient of the function. Using a line search algorithm, a suitable step size is computed to ensure a decrease in the error function value when we move along the computed descent direction.

III. EXTRACTING M-of-N RULES

The outline of our algorithm is as follows.

Algorithm MofN3 (**M-of-N** rules from **N**eural **N**etwork)

1. Train and prune a network.

2. Cluster the hidden unit activation values of the pruned network.

3. Generate classification rules in terms of the clustered activation values.

4. Replace the conditions of the rules generated in Step 3 by M-of-N conditions.

The hidden units have their activation values in the interval $[-1, 1]$. Before rules are extracted, we cluster these activation values in order to have the network classifications determined by the combinations of a small number of clusters. Discretization algorithms that are normally used to discretize input data can be used for clustering the activations of the hidden units, among these algorithms are ChiMerge [11] and Chi2 [12]. It can be expected that few clusters of hidden unit activation values will result in a more compact set of rules.

Clustering the hidden unit activation values in Step 2 of the algorithm MofN3 is equivalent to dividing the interval $[-1, 1]$ into disjoint subintervals. The method that we use for clustering is Chi2. It first sorts the activation values at the first hidden unit in increasing order. Initial subintervals are formed by placing each unique value in its own subinterval. The χ^2 value of adjacent subintervals are computed and the adjacent subintervals with the lowest χ^2 value are merged. Chi2 continues merging the subintervals as long as the resulting merged values still contain sufficient information to distinguish activation values of patterns that belong to different classes. When there are no more subintervals in the first hidden unit that can be merged, Chi2 repeats the process with the activation values of the next hidden unit. Thus when Chi2 terminates, a hidden unit's activation range is divided into ℓ disjoint subintervals $S_1 = [-1, U_1), S_2 = [L_2, U_2), \ldots, S_\ell = [L_\ell, 1]$, where $U_j = L_{j+1}$ and $L_j < U_j$ for all j.

After clustering, the transformed dataset is usually much simpler than the original dataset. By simpler we mean that the dimensionality of the data has been reduced from N to $\overline{H}$, where $\overline{H}$ is the number of hidden units left after pruning. By clustering the activation values, the number of possible unique patterns is also reduced. Let $h_1, h_2, \ldots, h_{\overline{H}}$ be the number of subintervals found by Chi2 for hidden unit $1, 2 \ldots \overline{H}$, respectively, then the number of unique patterns is bounded by $h_1 \times h_2 \times \ldots \times h_{\overline{H}}$.

In Step 3 of MofN3, we apply X2R [13] to generate the classification rules. X2R is a fast rule generator that is particularly suitable for moderate sized datasets with discrete attribute values. It consists of three steps. In the first step, it generates rules with fewest conditions and marks the patterns covered by each rule until all patterns are covered by at least one rule. In the second step, rules are clustered in terms of their class labels. Finally, redundant rules in a cluster are pruned in the last step.

The rules generated by X2R from the clusters of hidden unit activations are DNF rules where each rule condition involves an activation value α and is of the following form

$$L \leq \alpha < U \tag{6}$$

for some values of L and U such that $-1 < L < U < 1$. (Two exceptions are when $\alpha \in S_1$, where we have $\alpha < U_1$ and when $\alpha \in S_\ell$, where we have $\alpha \geq L_\ell$). Before explaining how this condition can be re-expressed as an M-of-N rule involving the original input attributes of the data in Step 4 of MofN3, we state our 2 assumptions regarding the input data and the weights of the network:

Assumption 1. All the attributes of the data are binary valued, -1 or 1.

Assumption 2. After the network is pruned, each remaining weight w for a connection from an input unit to a hidden unit is sufficiently large, that is, $|\tanh(K_1, w)| \approx 1$.

Assumption 1 can always be satisfied by recoding of the data. To check if a pruned network satisfies Assumption 2, we replace all the negative connections by -1 and all the positive weights by 1. If the classification accuracy of the network is not affected, then this assumption is satisfied. Otherwise, DNF rules instead of M-of-N rules can be extracted using NeuroRule [2].

The activation α has been computed as (2)

$$\alpha = \tanh\left(K_0, \left(\sum_n x_n \times \tanh(K_1, w_n)\right) + \tau\right)$$

(we drop the pattern index and the hidden unit index for simplicity). Before we analyze what set of input values produces an activation value that satisfies inequality (6) for some values of (L, U), let us define

- $\overline{L} = \lceil \tanh^{-1}(K_0, L) - \tau \rceil$ and $\overline{U} = \lfloor \tanh^{-1}(K_0, U) - \tau \rfloor$

  ($\overline{L}$ and $\overline{U}$ are well-defined since $\tanh(K, x)$ is a one to one function).

- $\overline{N}$: the number of input connections connected to the hidden unit.

- $\overline{x}$: not x, hence by Assumption 1, $\overline{x} = -x$.

- $\mathcal{W}_+, \mathcal{W}_-$: the set of connections with positive weight and negative weight, respectively.

- $\mathcal{I}_+, \mathcal{I}_-$: the set of inputs with positive connections and negative connections, respectively.

We have the following

$$L \ \leq \ \alpha < \ U$$

$$\Updownarrow$$

$$\tanh^{-1}(K_0, L) \ \leq \ \sum_n x_n \times \tanh(K_1, w_n) + \tau \ < \ \tanh^{-1}(K_0, U)$$

$$\Updownarrow$$

$$\tanh^{-1}(K_0, L) - \tau \ \leq \ \sum_{w_n \in \mathcal{W}_+} x_n \times \tanh(K_1, w_n) \ + \sum_{w_n \in \mathcal{W}_-} x_n \times \tanh(K_1, w_n) \ < \ \tanh^{-1}(K_0, U) - \tau$$

$$\Updownarrow$$

$$\tanh^{-1}(K_0, L) - \tau \ \leq \ \sum_{w_n \in \mathcal{W}_+} x_n - \sum_{w_n \in \mathcal{W}_-} x_n \ < \ \tanh^{-1}(K_0, U) - \tau \tag{7}$$

$$\Updownarrow$$

$$\overline{L} \ \leq \ \sum_{w_n \in \mathcal{W}_+} x_n + \sum_{w_n \in \mathcal{W}_-} \overline{x}_n \ \leq \ \overline{U} \tag{8}$$

Inequality (7) follows from Assumption 2, while inequality (8) follows from the definition of $\overline{x}, \overline{L}$ and $\overline{U}$, and Assumption 1.

Let $\mathcal{C}$ be the disjunction

$$\left(\bigvee_{I_n \in \mathcal{I}_+} (I_n = 1) \right) \bigvee \left(\bigvee_{I_n \in \mathcal{I}_-} (I_n = -1) \right)$$

The rule that we are extracting is: if M of the $\overline{N}$ conditions of $\mathcal{C}$ are satisfied, then $L \leq \alpha < U$. Let $\overline{M}$ be the number of conditions of $\mathcal{C}$ that are **not** satisfied. It follows from (8), that we must have

$$\overline{L} \leq M - \overline{M} \leq \overline{U} \tag{9}$$

Since $M + \overline{M} = \overline{N}$, if we let $M = \lfloor \frac{1}{2}(\overline{N} + \overline{U}) \rfloor$ and $\overline{M} = \overline{N} - M$, we have a pair of values $(M, \overline{M})$ that satisfies condition (9). All other pairs of $(M, \overline{M})$ that satisfy this condition can be found by simply decreasing M by one (and increasing $\overline{M}$ by one) as long as $M - \overline{M} \geq \overline{L}, M \geq 0$. Alternatively, we may let $M = \lceil \frac{1}{2}(\overline{N} + \overline{L}) \rceil$ and find all other pairs of $(M, \overline{M})$ by increasing M by one as long as $M - \overline{M} \leq \overline{U}$ and $M \leq \overline{N}$.

IV. ILLUSTRATIVE EXAMPLES

The splice junction determination is selected for testing our M-of-N rule extraction algorithm. This problem has been used by Towell and Shavlik [3] to test their algorithms which extract symbolic rules from knowledge based neural networks. Each sample in the dataset is described by a 60-nucleotide-long DNA sequence. A nucleotide may assume one of the 4 possible values: G (Guanine), T (Thymine), C (Cytosine), or A (Adenine). These values are binary coded as {1,-1,-1,-1}, {-1,1,-1,-1}, {-1,-1,1,-1} and {-1,-1,-1,1}, respectively. By convention, the positions of the nucleotides in a sequence are numbered from -30 to -1 and from 1 to 30. The learning task is to recognize the type of boundary at the center of the sequence. This boundary can be an intron/exon (IE) boundary, exon/intron boundary (EI), or neither (N). The target output for each pattern in class IE was $\{1, 0, 0\}$, in class EI $\{0, 1, 0\}$, and in class N $\{0, 0, 1\}$.

We trained and pruned the neural networks using 1006 training samples and tested the accuracy of the pruned networks on a dataset of 3175 test samples. Each starting network had 8 hidden units. Aiming to achieve a comparable accuracy rate to that reported in [3], we terminated network pruning when removal of an additional connection caused the accuracy of the network on the training dataset to drop below 92 %. The same set of values for the parameters in the penalty function (5) was used to obtain all the results presented in this and the next sections. They were $\epsilon = 1$ and $K_1 = 1$. The value of K_0 (2) was also set to 1.

Networks trained with different initial random weights are very likely to result in different pruned networks. We illustrate how 2 different sets of rules are extracted from 2 networks that have been trained using the same set of training samples. As may be expected, the network with fewer connections results in a simpler set of rules than a network with more connections. The rule set extracted from the latter, however, achieves higher predictive accuracy.

Example 1.

The first pruned network is depicted in Figure 1. In this figure, we label the remaining 8 inputs $I1, I2, \ldots, I8$ for convenience. The relationships between these inputs and the original nucleotide sequence are as follows:

$I1 = 1$ iff nucleotide @-3 = T (this means that the nucleotide at position -3 is Thymine).

$I2 = 1$ iff nucleotide @-3 = C.

$I3 = 1$ iff nucleotide @-2 = A.

$I4 = 1$ iff nucleotide @-1 = G.

$I5 = 1$ iff nucleotide @+1 = G.

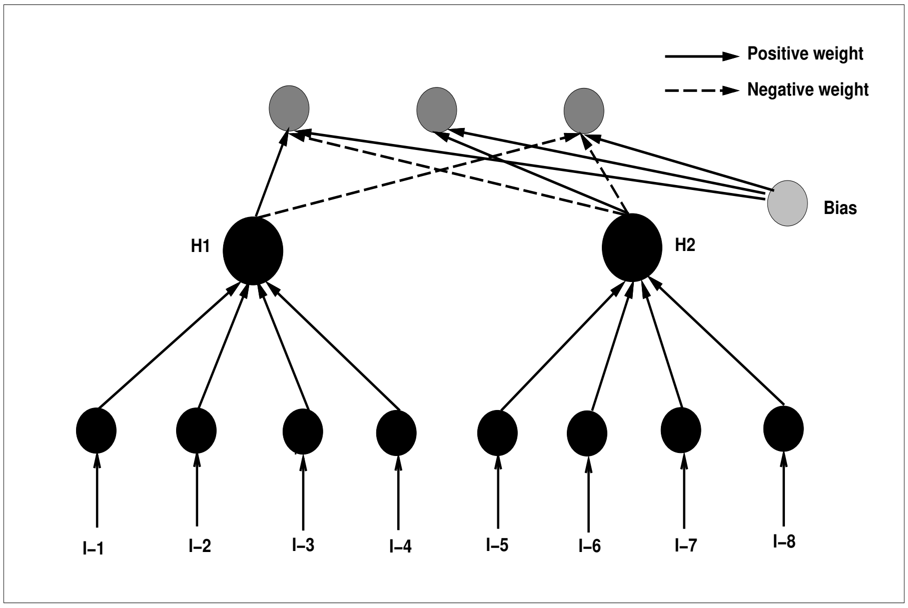

Fig. 1: A pruned network for the splice junction determination problem. The accuracy rates on the training and test datasets are 92.84 % and 92.94 %, respectively.

$I6 = 1$ iff nucleotide @+2 = T.

$I7 = 1$ iff nucleotide @+3 = A.

$I8 = 1$ iff nucleotide @+5 = G.

The accuracy of the pruned network is 92.84 %. It correctly classifies 934 of the 1006 training patterns. When the activation values of the 934 correctly patterns were clustered by Chi2, the results were as follows. In hidden unit 1, two subintervals were found: $[-1, 0.96)$ and $[0.96, 1]$. Similarly at the hidden unit 2, the subintervals found were $[-1, 0.96)$ and $[0.96, 1]$. We use the notation $\alpha_i = 1$ or 2 for $i = 1$ and 2 to denote an activation value at hidden unit i falling in the first or second subinterval. Since there are only 2 subintervals each at the two hidden units, there are at most 4 unique discrete representations of the training samples. They are as follows:

$$(\alpha_1, \alpha_2) = \begin{cases} (2,1) & \text{Class IE} \\ (1,2) & \text{Class EI} \\ (2,2) & \text{Class EI} \\ (1,1) & \text{Class N} \end{cases}$$

We can classify these four samples by the following simple rule:

If $\alpha_2 = 2$, then EI,

else if $\alpha_1 = 2$, then IE,

else N.

In order to replace the conditions of the above rule by M-of-N conditions, we must first compute $\tanh^{-1}(K_0, 0.96) = 1.95$. Hence, $\overline{L} = \lceil 1.95 \rceil = 2$. We have $M = \lceil \frac{1}{2}(\overline{N} + \overline{L}) \rceil = \lceil (\frac{1}{2}(4+2) \rceil) = 3$ and $\overline{M} = 4 - 3 = 1$. A second pair of values, $(M, \overline{M}) = (4, 0)$ also satisfies the conditions $M + \overline{M} = 4, M - \overline{M} \geq 2$. We conclude that $\alpha_2 = 2$ if and only if 3 or 4 of the inputs I5, I6, I7, I8 are equal to 1.

Coincidentally, the first hidden unit of the network also has 4 input connections, all of which have positive weights. The subintervals found by Chi2 are also identical as those of the first hidden unit. By similar derivation, we obtain: if 3 or 4 of the inputs I1, I2, I3, I4 are equal to 1, then $\alpha_1 = 2$. Substituting these M-of-N rules into the rules that classify the patterns, we obtain the following rules:

```
If 3 of (@+1=G, @+2=T, @+3=A, @+5=G), then EI,
else if 3 of (@-3=T, @-3=C, @-2=A, @-1=G), then IE,
else N.
```

(In the above rule conditions, 3 of (.....) is true if *at least* 3 of the conditions in the brackets are satisfied).

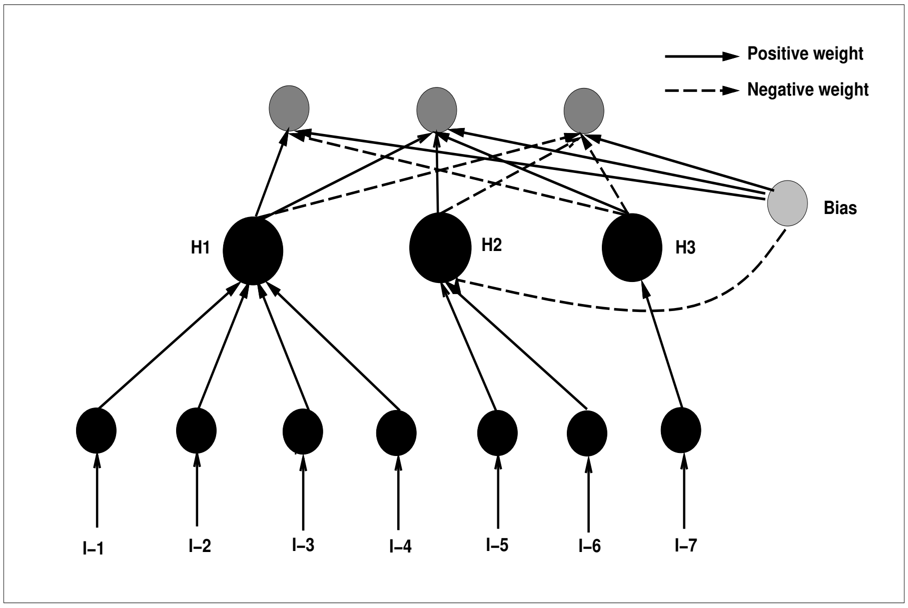

Fig. 2: A pruned network for the splice junction determination problem. The accuracy rates on the training and test datasets are 93.94 % and 94.20 %, respectively.

The accuracy rates of the extracted rule on the training and test datasets are the same as those of the pruned network. Of the 3175 patterns in the test dataset, 2951 are correctly classified giving a predictive accuracy rate of 92.94 %.

Example 2.

The second pruned network is depicted in Figure 2. Its accuracy rates on the training and test datasets are 93.94 % and 94.20 %, respectively. After clustering its activation values by Chi2, the following subintervals were found:

Hidden unit 1: $[-1, 0), [0, 0.96), [0.96, 1]$.

Hidden unit 2: $[-1.0.76), [0.76, 1]$

Hidden unit 3: $[-1.0.76), [0.76, 1]$

The three clusters at hidden unit 1 and two clusters each at hidden units 2 and 3 produce 12 unique hidden representations of the training samples. The classification rule in terms of the clustered activation values is as follows:

If $\alpha_1 = 3$ and $\alpha_3 = 1$, then IE,

else if $\alpha_1 = 3$ and $\alpha_2 = 1$, then IE,

else if $\alpha_2 = 2$ and $\alpha_3 = 2$, then EI,

else if $\alpha_1 = 2$ and $\alpha_2 = 2$, then EI,

else N.

For hidden unit 2, $\overline{N} = 2$. We compute $\tanh^{-1}(K_0, 0.76) = 0.996$. Hence if, $I5 + I6 - 1 < 0.996$, then $\alpha_2 = 1$, else $\alpha_2 = 2$. We have $\overline{U} = \lfloor 0.996 + 1 \rfloor = 1$ and $M = \lfloor \frac{1}{2}(\overline{N} + \overline{U}) \rfloor = \lfloor \frac{1}{2}(2+1) \rfloor = 1, \overline{M} = \overline{N} - 1 = 1$. It follows that if $(M, \overline{M}) = (1,1)$ or $(M, \overline{M}) = (0,2)$ then $\alpha_2 = 1$; otherwise if $(M, \overline{M}) = (2,0)$, then $\alpha_2 = 2$.

Hidden unit 3 is connected to only 1 input with positive connection, hence if $I7 = -1$, then $\alpha_3 = 1$, otherwise $\alpha_3 = 2$. The condition in terms of the original attributes is if @5=H, then $\alpha_3 = 1$; else if @5=G, then $\alpha_3 = 2$ (H denotes A or C or T).

Upon extracting the M-of-N rules for hidden unit 1 and substituting the conditions of the rule by the original attributes of the data, we obtain

```
If 3 of (@-3=T, @-3=C, @-2=A, @-1=G) and @+5=H, then IE,
else if 3 of (@-3=T, @-3=C, @-2=A, @-1=G) and not more than 1 of (@+1=G, @+2=T),  then IE,
else if 2 of (@+1=G, @+2=T) and @+5=G, then EI,
else if exactly 2 of (@-3=T, @-3=C, @-2=A, @-1=G) and 2 of (@+1=G, @+2=T), then EI,
else N.
```

The accuracy rates of this rule on both the training and test datasets are the same as the accuracy rates of pruned network from which they are extracted.

V. EXPERIMENTAL RESULTS

All datasets used in the experiments are publicly available via anonymous ftp from ics.uci.edu [14]. These dataset are:

1. The splice junction dataset. The characteristics of the patterns in this dataset have been described in the previous section. The dataset that consists of 1006 patterns was used.

2. The 3 monks datasets [15]. Each pattern of the monks problems is described by six attributes: head-shape (round, square, or octagon), body-shape (round, square, or octagon), is-smiling (yes or no), holding (sword, balloon, or flag), jacket-color (red, yellow, green, or blue), and has-tie (yes or no). The classification tasks are to distinguish monks from non monks:

   - Monks1: (head-shape = body-shape) or (jacket-color = red).
   - Monks2: Exactly two of the six attributes have their first value.
   - Monks3: (jacket-color is green and holding a sword) or (jacket-color is not blue and body-shape is not octagon).

   Each input attribute value is coded as -1 or 1. Hence, 17 input units are required. The total number of patterns in each of the three datasets is 432.

3. The mushroom classification dataset [16]. The dataset consists of 8124 samples, each of which is described by 22 nominal attributes that are the characteristics of species of mushroom. The classification task is to predict whether a mushroom is edible or poisonous. The attributes of the data were converted to 126 binary inputs before training. In order to reduce computation time, only 2000 randomly selected samples were used.

4. The Wisconsin breast cancer classification dataset [17]. Each of the 699 patterns in the

TABLE I: The initial network topology (input, hidden and output units) and the average user time required for training and pruning. Figures in parentheses are standard deviations.

Data	Network topology	Time (seconds)
Monks1	$17 \times 8 \times 1$	2.74 (0.35)
Monks2	$17 \times 8 \times 1$	2.48 (0.56)
Monks3	$17 \times 8 \times 1$	1.93 (0.09)
S-junction	$240 \times 8 \times 3$	227.04 (4.37)
Mushroom	$127 \times 8 \times 1$	141.35 (15.43)
B-cancer	$90 \times 8 \times 1$	16.29 (3.52)

dataset is described by 9 attributes. The nine measurements taken from fine needle aspirates from human breast tissues correspond to cytological characteristics of a benign or of a malignant sample. These are $\mathcal{A}_1$. clump thickness, $\mathcal{A}_2$. uniformity of cell size, $\mathcal{A}_3$. uniformity of cell shape, $\mathcal{A}_4$. marginal adhesion, $\mathcal{A}_5$. single epithelial cell size, $\mathcal{A}_6$. bare nuclei, $\mathcal{A}_7$. bland chromatin, $\mathcal{A}_8$. normal nucleoli, and $\mathcal{A}_9$. mitosis. Each of these nine attributes of the fine needle aspirates was graded 1 to 10 at the time of sample collection, with 1 being the closest to benign and 10 the most anaplastic (more detailed description of these attributes can be found in [17]). Since the attributes are integer-valued ranging from 1 to 10, we created 10 input units for each attribute. The attribute values are represented by the following encoding scheme:

$$\mathcal{A}_i = k \Leftrightarrow \begin{cases} I_{10\times(i-1)+j} = 1 & \forall \quad j = 1, 2, \ldots, k \\ I_{10\times(i-1)+j} = -1 & \forall \quad j = k+1, k+2, \ldots, 10 \end{cases}$$

The classification task is to distinguish between benign and malignant samples.

TABLE II: The predictive accuracy and complexity of the pruned networks.

Data	Accuracy (%)	# connections	# hidden units	% unit weight	NUW
Monks1	100.00 (0.00)	13.30 (0.75)	3.00 (0.00)	100.00 (0.00)	30
Monks2	100.00 (0.00)	18.93 (0.25)	2.00 (0.00)	100.00 (0.00)	30
Monks3	100.00 (0.00)	8.80 (1.00)	1.97 (0.41)	95.00 (15.26)	27
S-junction	92.28 (2.75)	18.00 (2.36)	2.50 (0.57)	93.52 (10.60)	20
Mushroom	98.12 (3.21)	6.47 (0.68)	1.17 (0.38)	100.00 (0.00)	30
B-cancer	93.99 (4.81)	6.60 (1.43)	2.13 (0.63)	96.07 (12.46)	27

Three repetitions of ten-fold cross-validation [18] experimental procedure were conducted. In a ten-fold cross-validation experimental setting, the dataset is divided into ten partitions of equal size. A network is then trained using the patterns in nine partitions and its predictive accuracy is evaluated on the remaining partition. This is repeated ten times so that each partition is used as the test set once. Each starting network had 8 hidden units. The initial weights were assigned randomly and uniformly in the interval $[-1, 1]$. The topology of the initial networks and the average user time required for training and pruning one network are shown in Table I. The training and pruning algorithms were implemented using FORTRAN 77 and run on a Sun Enterprise Ultra 450.

In order to check if Assumption 2 is satisfied by a pruned network, we do the following: (1) find a connection weight from an input unit to a hidden unit that is not -1 or 1 with the largest magnitude, replace this weight by -1 or 1 depending on whether it is negative or positive, (2) if the accuracy of the network on the training data does not drop, repeat from Step 1, otherwise restore the weight to its original value, (3) count how many weights have value -1 or 1.

TABLE III: The predictive accuracy, complexity and fidelity of the extracted rules.

Data	# rules	Rule acc. (%)	# MofN/rule	# antecedents/rule	Fidelity
Monks1	4.37 (0.49)	100.00 (0.00)	2.02 (0.06)	6.64 (0.76)	100.00 (0.00)
Monks2	2.00 (0.00)	100.00 (0.00)	1.00 (0.00)	7.53 (1.01)	100.00 (0.00)
Monks3	2.00 (0.00)	100.00 (0.00)	1.97 (0.41)	5.83 (0.65)	100.00 (0.00)
S-junction	4.87 (2.03)	92.41 (2.75)	1.92 (0.46)	6.83 (1.38)	99.87 (0.57)
Mushroom	2.00 (0.00)	98.12 (3.21)	1.17 (0.38)	4.30 (0.60)	100.00 (0.00)
B-cancer	2.90 (0.76)	94.04 (4.74)	1.69 (0.40)	2.97 (1.17)	99.95 (0.26)

The results of the experiments are tabulated in Tables II and III. In Table II, we show the statistics from the pruned networks: the predictive accuracy, the number of connections, the number of hidden units, and the percentage of connections from the input layer to the hidden layer whose weights had been replaced by -1 or 1 (unit weight). Each value is an average from 30 networks with the corresponding standard deviation in parentheses. We also show the number of networks (out of 30) with all of their input-layer to hidden-layer connection weights replaced by Unit Weights (NUW – Networks with Unit Weights).

From Table II, we observe that the numbers of unpruned connections and hidden units are small compared to those in the original networks. For the Monks1, Monks2 and the mushroom problems, all unpruned connections from the input units to the hidden units can be replaced by unit weights. For the Monks3 problem, 3 networks still have some non-unit weights. All these networks have only 1 hidden unit, indicating that the patterns are linearly separable. Although 93.52 % of the input-hidden layer connections of the splice junction networks can be replaced by -1 or 1, ten (33.33 %) of the 30 networks still have some non-unit weights.

In Table III, we show the number of extracted rules, the accuracy of the rules, the average M-of-N conditions and number of antecedents per rule. The number of rules includes the default

TABLE IV: Comparison between MofN3 and other rule generating methods on the splice junction problem.

Method	Accuracy (%)	# rules	# antecedents/rule
MofN3	92.41	4.87	6.83
MofN	92.8	20	5.5
Subset	86.1	> 40	5.9

rule. For example, the rules in Example 1 in the previous section consists of 3 rules with a total of 2 M-of-N conditions and 8 antecedents. The fidelity of the rules are shown in the last column. A 100% fidelity indicates that the neural networks and the rules extracted from them always give identical predictions. The figures in this table indicate that few rules with few M-of-N conditions can achieve high accuracy rates. The high fidelity rates also show that the extracted rules mimic very well the classification of the networks from which they were extracted.

Table IV compares the results of MofN3 on the splice junction problem to those of MofN and Subset methods as reported in [3]. For this problem the average MofN3 accuracy rate is 92.41 %, which is 0.4 % lower than the accuracy obtained by MofN method. However, the average MofN rule set is four times as large as an average MofN3 set. The number of antecedents per MofN rule is 5.5, while the corresponding number of MofN3 rule is 6.83. From the size of the rule sets and the average number of antecedents per rule, we can expect MofN3 rules to be easier to comprehend than MofN rules.

VI. CONCLUSION

Rules that distinguish patterns from different classes in a dataset may be more concisely expressed in terms of M-of-N rules rather than DNF rules. This paper describes the MofN3 algorithm for extracting M-of-N rules from trained neural networks. The process of extracting M-of-N rules by

MofN3 is greatly simplified by making the assumption that both the input data and the input-to-hidden-unit connection weights of the neural network (after applying the hyperbolic squashing function) are binary valued -1 or 1. The assumption that the input data are binary valued is not restrictive. The values of discrete input attributes can be easily converted to binary. Those of continuous attributes can be discretized into clusters and the clusters can then be represented as binary input. By minimizing the network error function which is augmented by a penalty term, we are able to obtain pruned networks where the assumption that the network weights are binary valued is usually satisfied. When the network pruning terminates, the magnitudes of the remaining connection weights between the input and hidden layers are not equal to 1. They are, however, near one. We check if they are sufficiently close to 1 by simply replacing the negative weights by -1 and the positive weights by 1. If the accuracy of the network remains unchanged, we proceed with the rule extraction.

The MofN3 algorithm for extracting M-of-N concepts from pruned networks that we have presented here has been shown to work well on both real-world and artificial datasets. Desirable properties of the extracted M-of-N rules include accuracy, simplicity and fidelity. The accuracy rates of extracted rules on the 3 artificial problems are always 100 %, while the accuracy rates on the 3 real-world datasets are comparable to those reported in the literature. The simplicity of the extracted rules is reflected in the small number of rules and the relatively small number of MofN conditions per rule. Rule simplicity is important as simple rules are clearly easier to understand than complex ones. Finally, the very high fidelity of the extracted rules shows that the rules are able to mimic the network from which they are extracted in their predictions. We can say that we are able to explain the trained network's output in terms of rules that are more comprehensible to humans than a collection of network weights.

REFERENCES

[1] R. Andrews, J. Diederich and A.B. Tickle, "Survey and critique of techniques for extracting rules from trained artificial neural networks," *Knowledge Based Systems*, vol. 8, no. 6, pp. 373–389, 1995.

[2] R. Setiono and H. Liu, "Symbolic representation of neural networks," *IEEE Computer*, pp. 71–77, March 1996.

[3] 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.

[4] A. van Ooyen, A. and B. Nienhuis, B. (1992) "Improving the convergence of the backpropagation algorithm," *Neural Networks,* vol. 5, no. 3, pp. 465-471, 1992.

[5] J. Hertz, A. Krogh, and R.G. Palmer, *Introduction to the theory of neural computation.* Redwood City, CA: Addison Wesley, 1991.

[6] R. Setiono, "A penalty function approach for pruning feedforward neural networks," *Neural Computation*, vol. 9, no. 1, pp. 185–204, 1997.

[7] J.E. Dennis, Jr. and R.B. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations.* Englewood Cliffs, NJ: Prentice Hall, 1983.

[8] R. Battiti, "First- and second-order methods for learning: Between steepest descent and Newton's method," *Neural Computation,* vol. 4, pp. 141–166, 1992.

[9] 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.

[10] R. Setiono, "A neural network construction algorithm which maximizes the likelihood function," *Connection Science,* vol. 7, no. 2, pp. 147–166, 1995.

[11] R. Kerber, "ChiMerge: Discretization of numeric attributes," in *Proc. of the 9th National Conf. on AI*, AAAI Press/The MIT Press, 1992, pp. 123–128.

[12] H. Liu and R. Setiono, "Chi2: Feature selection and discretization of numeric attributes," in *Proc. of the 7th IEEE International Conf. on Tools with Artificial Intelligence*, 1995, pp. 388–391.

[13] H. Liu and S.T. Tan, "X2R: A fast rule generator," in *Proc. of IEEE International Conf. on Systems, Man and Cybernetics*, New York: IEEE Press, 1995, 1631–1635.

[14] C.J. Merz and P.M. Murphy, *UCI repository of machine learning databases [http://www.ics.uci.edu/mlearn/MLRepository.html]*, Department of Information and Computer Science, University of California, Irvine, 1996.

[15] S.B. Thrun, et al., "The MONK's problems - a performance comparison of different learning algorithm," Preprint CMU-CS-91-197, Carnegie Mellon University, Pittsburgh, PA., 1991.

[16] J.S. Schlimmer, "Concept acquisition through representational adjustment," PhD Dissertation, Dept. of Information and Computer Science, University of California, Davis, 1987.

[17] W.H. Wolberg and O.L. Mangasarian, "Multisurface method of pattern separation for medical diagnosis applied to breast cytology," *Proceedings of the National Academy of Sciences*, vol. 87, pp. 9193–9196, 1990.

[18] S.M. Weiss and C.A. Kulikowski, *Computer systems that learn.* San Mateo, CA: Morgan Kaufmann, 1990.