Symbolic Representation of Neural Networks

Rudy Setiono and Huan Liu
Department of Information Systems and Computer Science
National University of Singapore
Kent Ridge, Singapore 119260
{rudys,liuh}@iscs.nus.sg
phone: (65) 772-6297/772-6563

Abstract

Although backpropagation neural networks generally predict better than decision trees do for pattern classification problems, they are often regarded as black boxes, i.e., their predictions cannot be explained as those of decision trees. In many applications, more often than not, explicit knowledge is needed by human experts. This work derives symbolic representations from a neural network to make epxlicit each prediction of the network. An algorithm is proposed and implemented to extract symbolic rules from neural networks. Explicitness of the extracted rules is supported by comparing them to the symbolic rules generated by decision trees methods. Empirical study demonstrates that the proposed algorithm generates high quality rules from neural networks comparable with those of decision trees in terms of predictive accuracy, number of rules and average number of conditions for a rule. The symbolic rules from neural networks preserve high predictive accuracy of original networks.

1 Introduction

Researchers[1−3] have compared experimentally the performance of learning algorithms of decision trees and neural networks (NNs). A general picture of these comparisons is that: (1) Backpropagation (an NN learning method) usually requires a great deal more computation; (2) the predictive accuracy of both approaches is roughly the same, with backpropagation often slightly more accurate;[2] and (3) symbolic learning (decision trees induction) can produce interpretable rules while networks of weights are harder to interpret.[3] In effect, a neural network is widely regarded as a black box due to the fact that little is known about how its prediction is made.

Our view is that this is because we are not equipped with proper techniques to know more about how a neural network makes a prediction. If we can extract rules from neural networks as generating rules from decision trees, we can certainly understand better how a prediction is made. In addition, rules are a form of knowledge that can be easily verified by experts, passed on and expanded. Some recent works[4–6] have shown that rules can be extracted from networks. These algorithms are search-based methods that have exponential complexity. Subsets of incoming weights that exceed the bias on a unit are searched. Such sets are then rewritten as rules. To simplify the search process, some assumptions are made. One assumption is that the activation of a unit is either very close to 1 or very close to 0. This can restrict the capability of the network since when the sigmoid transfer function is used as the activation function, the activation of a unit can have any value in the interval (0,1).

In this paper, a novel way to understand a neural network is proposed. Understanding a neural network is achieved by having symbolic rules that represent the network decision process. NeuroRule is an algorithm that extracts these rules from a neural network. It consists of four phases: first, a weight-decay backpropagation network is built so that important connections are reflected by their larger weights; second, the network is pruned such that irrelevant connections and units are removed while its predictive accuracy is still maintained; third, the hidden unit activation values are discretized by clustering; and last, rules are extracted from the network with discretized hidden unit activation values. By drawing parallels with rules generated from decision trees, we show that networks can be interpreted by the rules extracted; the rules in general preserve the accuracy of the networks; and they also explain how a prediction is made.

In the next section, we describe NeuroRule with an emphasis on rule extraction. In Section 3, experiments and datasets are described; after giving a detailed example, a comparison with the rule generation method of decision trees is made in order to draw some parallels of rules from neural networks and from decision trees, and to clearly define what is meant by understandability. A general discussion is provided in Section 4. The conclusion is given in the final section.

2 NeuroRule: an algorithm for rule extraction

A standard three layer feedforward network is the base of the algorithm. Weight decay is implemented while backpropagation is carried out. After the network is pruned, its hidden units activation values are discretized. Rules are extracted by examining the discretized activation values of the hidden units. The algorithm is described in steps below.

2.1 Backpropagation with weight decay

The basic structure of the neural network in this work is a standard three-layer feedforward network, which consists of an input layer, I, a hidden layer, H, and an output layer, O. The number of input units corresponds to the dimensionality of the examples of a classification problem. The number of output units is determined by the number of classes in the data an example can be possibly classified to. The number of hidden units depends on the problem in hand. Two approaches to determine a suitable number of hidden units have been described in the literature. The first approach begins with a minimal number of hidden units, one or two, and more hidden units are added as they are needed to increase the accuracy of the network. The second approach begins with an oversized network and removes redundant connections in the network by pruning. In the process, hidden units that are not connected to any input units or/and output units can be removed as well. We adopt the second approach since we are also interested in removing irrelevant input units. After pruning, irrelevant input units can be identified and deleted from the network.

Given an n-dimensional example x^i, $i \in \{1,2,...,k\}$ as input, let w_l^m be the weight for the connection from input unit l, $l \in \{1,2,...,n\}$ to hidden unit m, $m \in \{1,2,...,h\}$ and v_p^m be the weight from hidden unit m to output unit p, $p \in \{1,2,...,o\}$, the pth output of the network for example x^i is obtained by computing

$$S_p^i = \sigma(\sum_{m=1}^{h} \alpha^m v_p^m), \quad \text{where} \tag{1}$$

$$\alpha^m = \delta(\sum_{l=1}^{n} x_l^i w_l^m), \quad \delta(x) = (e^x - e^{-x})/(e^x + e^{-x}). \tag{2}$$

The target output for an example x^i that belongs to class C_j is an o-dimensional vector t^i, where $t_p^i = 0$ if $p \neq j$ and $t_j^i = 1,\ j, p = 1, 2, \ldots o$. The backpropagation algorithm is applied to update the weights (w, v) and minimize the following function:

$$\theta(w, v) = F(w, v) + P(w, v), \tag{3}$$

where $F(w, v)$ is the cross entropy function:

$$F(w, v) = -\sum_{i=1}^{k} \sum_{p=1}^{o} \left(t_p^i \log S_p^i + (1 - t_p^i) \log(1 - S_p^i) \right). \tag{4}$$

and $P(w, v)$ is a penalty term used for weight decay:

$$P(w, v) = \epsilon_1 \left(\sum_{m=1}^{h} \sum_{\ell=1}^{n} \frac{\beta (w_\ell^m)^2}{1 + \beta (w_\ell^m)^2} + \sum_{m=1}^{h} \sum_{p=1}^{o} \frac{\beta (v_p^m)^2}{1 + \beta (v_p^m)^2} \right) \tag{5}$$

$$+ \ \epsilon_2 \left(\sum_{m=1}^{h} \sum_{\ell=1}^{n} (w_\ell^m)^2 + \sum_{m=1}^{h} \sum_{p=1}^{o} \left(v_p^m \right)^2 \right).$$

The values of the penalty parameters (ϵ_1, ϵ_2, and β) must be chosen to reflect the relative importance of the accuracy of the network versus its complexity. With larger parameter values, more connections will be removed from the network at the cost of a decrease in the network accuracy. We have found that $\epsilon_1 = 10^{-1}, \epsilon_2 = 10^{-5}$ and $\beta = 10$ work well for a wide range of problems tested. Using these set of parameter values, we have been able to obtain networks with fewer connections than previously reported for many learning problems.

2.2 Network pruning

A network pruning algorithm is briefly described below. This pruning algorithm removes the connections of the network according to the magnitudes of their weights (Equations 7 and 8 below). As our eventual goal is to get a set of simple rules that describe the classification process, it is important that all unnecessary connections be removed. In order to remove as many connections as possible, the weights of the network must be prevented from taking values that are too large. At the same time, weights of irrelevant connections should be encouraged to converge to zero. The penalty function (5) is found to be particularly suitable for these purposes.

Neural network pruning algorithm

1. Let η_1 and η_2 be positive scalars such that $\eta_1 + \eta_2 < 0.5$.

2. Pick a fully connected network. Train this network until a predetermined accuracy rate is achieved and for each correctly classified example the condition

$$\max_p |e_p^i| = \max_p |S_p^i - t_p^i| \leq \eta_1, \tag{6}$$

   where $\eta_1 \in [0, 0.5)$, is satisfied.

3. Let (w, v) be the weights of this network.
   For each w_ℓ^m, if

$$\max_p |v_p^m w_\ell^m| \leq 4\eta_2, \tag{7}$$

   then remove w_ℓ^m from the network

4. For each v_p^m, if

$$|v_p^m| \leq 4\eta_2, \tag{8}$$

   then remove v_p^m from the network

5. If no weight satisfies condition (7) or condition (8), then remove w_ℓ^m with the smallest product $\max_p |v_p^m w_\ell^m|$.

6. Retrain the network. If classification rate of the network falls below an acceptable level, then stop. Otherwise, go to Step 3.

2.3 Clustering of hidden unit activations

When network pruning is completed, the network contains only those salient connections. Nevertheless, rules are not readily extractable because the hidden unit activation values are continuous. The discretization of these values paves the way for rule extraction. The following algorithm discretizes the activation values of a hidden unit (many clustering algorithms can be used for this purpose).

Discretizing hidden unit activation values by clustering

For each hidden unit,

1. Let $\epsilon \in (0, 1)$,
2. Start with activation value α_0 of the first example in the training set,
3. Cluster activation values (α_i) for the remaining examples if $|\alpha_i - \alpha_0| < \epsilon$,
4. Represent this cluster's activation value by the average of the activation values in the cluster;
5. Select next α_0, repeat 3 and 4 for clustering until all activation values are clustered.

When the clustering is done, the network's accuracy is checked to see if it drops or not. A sufficiently small ϵ guarantees that the network with discretized activation values is as accurate as the original network with continuous activation values. Hence, if its accuracy does not drop and there are still many discrete values, clustering can be performed again with a larger ϵ to minimize the number of clusters. Otherwise, ϵ should be reduced to a smaller value.

2.4 Rule extraction

The data sets used for generating rules from input layer I to hidden layer H are reconstructed from the original training data set as follows: one data set for each hidden unit; these data sets contain only those inputs that are relevant in determining the activation values of the hidden unit. Each cluster in a hidden unit forms a class. That is, if there are m hidden units left in the network after pruning, there will be m such data sets. The number of classes in each data set is solely determined by the number of clusters in the corresponding hidden unit. For each hidden unit, a set of rules that describe the relationship between the inputs and the discretized activation values are extracted.

All possible combinations of the clusters of activation values are examined. Using the weights of the connections from hidden layer H to output layer O, predicted network outputs are determined. Rules are generated to describe these outputs in term of the clustered hidden unit activation values. Complete rules that define the decision process of the network in term of the original inputs are obtained by merging the two sets of rules, the first set are from the input layer to the hidden layer and the second from the hidden layer to the output layer.

When the number of inputs connected to a hidden unit is small, it will be trivial to extract rules that describe how each cluster of activation values is obtained. Similarly, when the total number of clusters in the hidden units is small, it is trivial to obtain rules to describe the network outputs in term of the activation values. For automatic rule generation, we have developed and implemented a general purpose algorithm **RG**. The outline of this rule generation algorithm is as follows.

Rule Generation (RG) algorithm

1. Group the data into D_p groups in terms of class values, $p \in \{1, 2, \ldots \nu\}$.

2. Set k, the number of rules to 0.

3. For each item i in D_c - the data of a certain class value c, do

   - Count the occurrence of each attribute value in D_o - the data of other class values;
   - Order all the attributes of item i according to their value occurrences;
   - Find the minimum number of attributes in the ordered attribute list that uniquely differentiate item i from other items in D_o;
   - Use the chosen attributes (values) to define rule R_k; and
   - Mark all the items in D_c that are covered by rule R_k as having been considered, increment k by 1.

4. If all items in D_p have been considered, stop; otherwise go to Step 3.

The rule generation algorithm **RG** possesses the following features: 1. it produces perfect rules, i.e. the error rate of the rules generated is no worse than the inconsistency rate δ present in the data; in the context of our discussion, identical input samples will always have the same clustered activation values, hence it is always possible to have zero error rate; and 2. the rules for G_j are order independent, (i.e. the accuracy of the rules is not affected by the order in which the rules are fired).

Combining all its components, we give below the outline of NeuroRule, an algorithm to generate symbolic rules from a three-layer feedforward neural network.

NeuroRule: generating symbolic rules from a neural network

1. Train and prune a neural network.
2. Discretize the activation values of the hidden units by clustering.
3. Using the discretized activation values as inputs, apply algorithm **RG** to generate rules that describe network outputs.
4. For each hidden unit: use **RG** to extract rules to describe its activation values in term of the inputs.
5. Generate rules that relate the inputs and the outputs by combining rules generated in Steps 3 and 4.
6. Prune redundant rules generated in Step 5.

---

3 Experiments and Results

In this section, we describe the datasets and their representations used in experiments. A detailed example is given to show how NeuroRule is applied to extract rules. Summary of the results from four datasets are given with a comparison to those produced by the decision tree induction methods. Understanding a neural network is achieved by being able to explain, based on the rules, how each prediction is made in parallel with understanding a decision tree by having rules generated from it.[7]

3.1 Datasets and Representations

Four datasets are used: 1. Iris – a classic dataset for pattern classification algorithms; 2. Breast Cancer – a widely tested real-world dataset for the Wisconsin Breast Cancer diagnosis; 3. Cleveland Heart Disease – a dataset of heart patients; and 4. Splice-junction – a dataset used in splice-junction determination. These datasets are obtainable from the University of California Irvine data repository for machine learning (via anonymous ftp to ics.uci.edu). The summary of these datasets, their representations, and how each dataset is used in experiments are given below.

- *Iris* - the dataset contains 50 examples each of the classes Iris setosa, Iris versicolor, and Iris virginica (species of iris). Each example is described using four numeric attributes ($\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ and $\mathcal{A}_4$): sepal-length, sepal-width, petal-length, and petal-width. Since each attribute takes a continuous value, the ChiMerge algorithm proposed by Kerber[8] was reimplemented to discretize attribute values by dividing the attribute's interval into a finite number of subintervals with the help of χ^2-statistic. The thermometer coding[9] is used to binarize the discretized values; 16, 9, 7, and 6 inputs (discrete values) for $\mathcal{A}_1$,$\mathcal{A}_2$, $\mathcal{A}_3$, and $\mathcal{A}_4$, respectively. For example, the 6 subintervals found for petal-width and their binary representation are as follows

Interval	Coding $I33-I38$	Interval	Coding $I33-I38$
$\mathcal{A}_4 \leq 0.4$	000001	$1.4 < \mathcal{A}_4 \leq 1.6$	001111
$0.4 < \mathcal{A}_4 \leq 1.3$	000011	$1.6 < \mathcal{A}_4 \leq 1.8$	011111
$1.3 < \mathcal{A}_4 \leq 1.4$	000111	$1.8 < \mathcal{A}_4$	111111

Table 1: The subintervals found by ChiMerge algorithm and their binary coding for $\mathcal{A}_4$ – petal width

With 1 input for hidden unit bias, there are a total of 39 inputs, $I1-I39$, and 3 outputs in the network. Examples in odd positions in the original dataset form the training set and the rest are for testing as was done by Fu.[4]

- *Breast Cancer* - the dataset consists of 699 examples, of which 458 examples are classified as benign, and 241 are malignant. 50% examples of each class are randomly selected (i.e., 229 benign and 121 malignant examples) for training, the rest for testing in the experiments. Each example is described by 9 attributes, each of which takes an ordinal integer value from 1 to 10. Due to the ordinal nature, the thermometer coding is used again to code the attribute values. Ten inputs correspond to 10 possible values of each attribute with all 10 inputs *on* representing value 10, only the rightmost input *on* for value 1, and the two rightmost inputs *on* for value 2, etc. With one input for bias, in total there are 91 inputs and 2 outputs in the original network.

- *Cleveland Heart Disease.* The dataset consists of 297 examples described by 13 attributes. The classification problem is to distinguish patients with heart disease from patients with no heart disease. Five of the attributes are continuous and the remaining eight discrete. ChiMerge algorithm is also applied on this dataset to discretize continuous attributes. This set is divided randomly into a training set consisting of 198 examples and a testing set consisting of the remaining 97 examples. The number of input units in the network is 21 and the number of output units is 2.

- *Splice-junction.* The dataset consists of 3175 examples, approximately 25 % are exon/intron boundaries (EI), 25% are intron/exon boundaries (IE), and remaining 50% are neither (N). Each example consists of a 60-nucleotide-long DNA sequence categorized with EI, IE or N. Each of these 60 attributes takes one of the four values: G, T, C or A that are coded as 1000, 0100, 0010, and 0001, respectively. Three class values (EI, IE, N) are similarly coded as 100, 010, and 001, respectively. With one input for bias, there are total 241 inputs and three outputs for the original network. For the results presented here, 1006 of the 3175 examples are chosen as training data. The testing set consists of all 3175 examples.

3.2 A Detailed Example – Iris Data Classification

This example shows in detail how rules are extracted by NeuroRule. In the experiment, 100 fully connected neural networks were used as the starting networks. Each of these networks

Min. network acc.	**95 %**	**97 %**
No. connections	9.26 (2.33)	10.66 (3.06)
Acc. on train data	97.20 (1.45) %	99.03 (0.84) %
Acc. on test data	92.32 (2.49) %	94.55 (2.03) %

Table 2: Average number of connections and accuracy of 100 pruned networks on the training and testing data sets of the Iris problem and their standard deviations.

consisted of 39 input units, 3 hidden units and 3 output units. These networks were trained with initial weights that had been randomly generated in the interval $[-1, 1]$. Each of the trained networks was pruned until its accuracy on the training data dropped below 95%. The weights and topology of networks with the smallest number of connections and an accuracy rate of more than 97% were saved for possible rule extraction. The results of these experiments are summarized in Table 2 in which we list the average number of connections in the pruned networks and their average accuracy rates on the training data and the testing data. Statistics in the second column of this table were obtained from 100 pruned networks, all of which had accuracy rates on the training data of at least 95 %. In the third column, the figures were obtained from 100 pruned networks with accuracy of at least 97 % on the training data.

After Step 1 of NeuroRule, a pruned network with only 2 hidden units and a total of 8 connections depicted in Figure 1 was obtained. Its accuracy rates are 98.67% on the training set and 97.33% on the testing set.

The clustering algorithm in Step 2 of NeuroRule found only 2 clusters of activation values at each of the two hidden units of the pruned network. At hidden unit 1, 49 of 75 training examples had activation values equal to 0 and the remaining 26 had activation values equal to 1. At hidden unit 2, the activation value of 25 examples was 1 and the activation value of the remaining 50 examples was -0.5. Since we have two activation values at each of the two hidden units, four different outcomes at the output units are possible. These possible outcomes are listed in Table 3.

In Step 3, RG generates the following rules that classify an example according to the

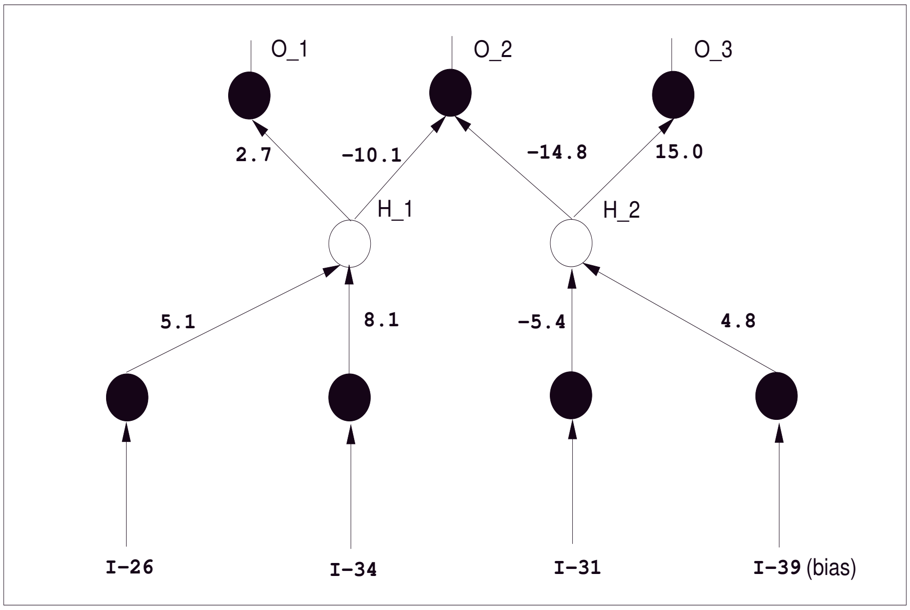

Figure 1: Pruned network with only 8 connections for the Iris problem, accuracy on training set = 98.67 %, accuracy on the testing set = 97.33 %. Figure next to a connection indicates the weight of that connection.

Activations		Pred. output			Class
H_1	H_2	O_1	O_2	O_3	
0	1	0.5	0	1	*setosa*
0	-0.5	0.5	1	0	*versicolor*
1	1	0.9	0	1	*setosa*
1	-0.5	0.9	0	0	*virginica*

Table 3: Discretized hidden unit activation values and predicted outputs of the network in Figure 1.

Hidden unit 1:

I_{26}	I_{34}	Value
0	0	0
0	1	1
1	0	1
1	1	1

If $I_{26} = I_{34} = 0$, then H_1 = 0

Else H_1 = 1

Hidden unit 2:

I_{31}	I_{39}	Value
0	1	1
1	1	-0.5

If $I_{31} = 0$, then H_2 = 1

Else H_2 = -0.5

Table 4: Discretized activation values of the hidden units and the rules that define them in term of network inputs.

discretized activation values of the hidden units:

If $H_2 = 1$, then *Iris setosa*

Else if $H_1 = 0$ and $H_2 = -0.5$, then *Iris versicolor*

Else *Iris virginica*

Step 4 of NeuroRule generates rules that determine the activation values of the hidden units in term of the network inputs. As seen in Figure 1, only two inputs, I_31 and I_39, determine the activation values of the second hidden unit, H_2. However, since I_39 is 1 for all the training data, H_2 is effectively determined by I_31. Since the weights of the arcs connecting input units 31 and 39 to the second hidden unit are -5.4 and 4.8 respectively, it is easy to conclude that if I_31 = 0, then H_2 is 1, otherwise, H_2 is -0.52. This implies that an example will be classified as *Iris setosa* only if I_31 is 0 (hence H_2 is 1).

The activation value of the first hidden unit, H_1, depends only on I_26 and I_34. The weights of the arcs connecting input units 26 and 34 to the first hidden unit are 5.1 and 8.1, respectively, hence H_1 is 0 if and only if I_26 = I_34 = 0. Other input combinations will yield value 1 for H_1. Hence, an example with I_31 = 1, I_26 = I_34 = 0 will be classified as *Iris*

	NN Rules		DT Rules	
	Training	Testing	Training	Testing
setosa	25/25	25/25	25/25	25/25
versicolor	24/25	23/25	24/25	24/25
virginica	25/25	25/25	24/25	22/25
Overall	74/75	73/75	73/75	71/75
	98.67%	97.33%	97.33%	94.67%

Table 5: Accuracy of NN rules and DT rules on the training and testing data.

versicolor. The summary of how each cluster of activation values at the two hidden units are obtained in term of possible input values and the rules generated by **RG** are summarized in Table 4.

Combining the two sets of rules obtained in Steps 3 and 4, NeuroRule obtained a set rules that classify iris species in term of the binarized network inputs. With the thermometer coding scheme used for the input, a complete set of rules can be easily obtained in terms of the original attributes of the iris data set. The accuracy of this rule set is summarized in Table 5.

<u>NN Rules</u>

Rule 1: If Petal-length $\leq$ 1.9 then *Iris setosa*

Rule 2: If Petal-length $\leq$ 4.9 and Petal-width $\leq$ 1.6 then *Iris versicolor*

Default Rule: *Iris virginica.*

For reference, the rule set (DT Rules) generated by C4.5rules (based on a decision tree method but generate more concise rules than the tree itself) is included here:

<u>DT Rules</u>

Rule 1: If Petal-length $\leq$ 1.9 then *Iris setosa*

Rule 2: If Petal-length $>$ 1.9 and Petal-width $\leq$ 1.6 then *Iris versicolor*

Rule 3: If Petal-width $>$ 1.6 then *Iris virginica*

Default Rule: *Iris setosa.*

3.3 Comparisons

In this section, parallels are drawn between rules extracted from both neural networks and decision trees (NN rules vs. DT rules). Obtaining NN rules is equivalent to having a symbolic representations for neural networks. *Understanding* neural networks is partly defined as being able to interpret in the sense that a prediction can be explained in terms of inputs (or attribute values). Choosing to compare NN rules with DT rules is due to the fact that DT rules are considered best understandable among the available choices. A rule in discussion consists of two parts: the if-part is made of a conjunction of conditions, and the then-part specifies a class value. The conditions of a rule are in forms of "A_i *op* V_j", where $op \in \{=, <, \geq\}$, i.e., attribute A_i takes value that is equal to, less than or greater than or equal to V_j. When a rule is fired, a prediction is given that the example under consideration belongs to class C_k. By examining the fired rule, we can explain how the prediction is attained.

C4.5 and C4.5rules[7] were run on the four datasets to generate DT rules. Briefly, C4.5 generates a decision tree which C4.5rules generalizes to rules. Since researchers[6,10] observed that mapping many-valued variables to two-valued variables results in decision trees with higher classification accuracy, the same binary coded data for neural networks were used for C4.5 and C4.5rules.

Being explicable is only one aspect of understandability. A rule with many conditions is harder to understand than a rule with fewer conditions. Too many rules also hinder humans understanding of the data under examination. In addition to understandability, rules without generalization (i.e., high accuracy on testing data) are not much of use. Hence, the comparison is performed along three dimensions: 1. predictive accuracy; 2. average number of conditions of a rule; and 3. number of rules (see Figures 2-4).

In the previous subsection, the NN and DT rules are shown for the Iris data. We give here the rules for the breast cancer data. Let us label the ten attributes of the data by $\mathcal{A}_1, \mathcal{A}_2, \ldots \mathcal{A}_{10}$, the rules are:

<u>**NN Rules**</u>

Rule 1: If $\mathcal{A}_1 < 7$ and $\mathcal{A}_2 < 8$ and $\mathcal{A}_3 < 3$ and $\mathcal{A}_8 < 9$, then *benign*

Rule 2: If $\mathcal{A}_1 < 7$ and $\mathcal{A}_2 < 8$ and $\mathcal{A}_6 < 3$ and $\mathcal{A}_8 < 9$, then *benign*

Rule 3: If $\mathcal{A}_2 < 8$ and $\mathcal{A}_3 < 3$ and $\mathcal{A}_6 < 3$ and $\mathcal{A}_8 < 9$, then *benign*

Default Rule: *malignant*

DT Rules

Rule 1: If $\mathcal{A}_1 < 7$ and $\mathcal{A}_2 < 8$ and $\mathcal{A}_2 < 3$, then *benign*

Rule 2: If $\mathcal{A}_1 < 7$ and $\mathcal{A}_2 < 2$, then *benign*

Rule 3: If $\mathcal{A}_2 \geq 5$, then *malignant*

Rule 4: If $\mathcal{A}_6 \geq 9$, then *malignant*

Rule 5: If $\mathcal{A}_1 \geq 7$, then *malignant*

Rule 6: If $\mathcal{A}_4 \geq 4$, then *malignant*

Default Rule: *benign*

As seen above, the two sets of rules are not much alike, although both achieve high predictive accuracy rates. NN rules emphasis the use of parallel features: each rule consists of as many as four attributes; while DT rules focus on individual feature; most of its rules involve only a single attribute value. What is particularly interesting here is that there are fewer specific NN rules having more conditions/attributes than general DT rules having fewer conditions/attributes. This runs counter to our intuition. Normally, we would expect fewer general rules to cover a dataset than specific rules for the same set of data. A possible explanation is that neural networks really explore the coordinated effects of the attributes on the outcome of their predictions.

The reasoning behind the comparisons is that if NN rules are comparable with DT rules, since the latter are admittedly interpretable, so should the former. Now that each prediction can be explained in light of some rule, and those rules have direct links to the neural network, it can be concluded that the network's behavior can be understood via those rules. In other words, a symbolic representation is obtained for the network.

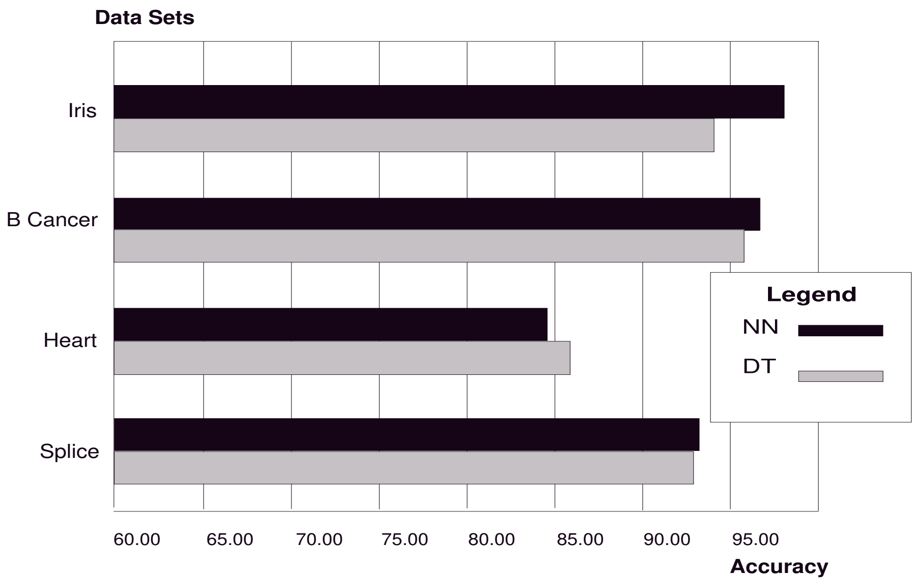

Figure 2: Comparison: predictive accuracy for the four datasets.

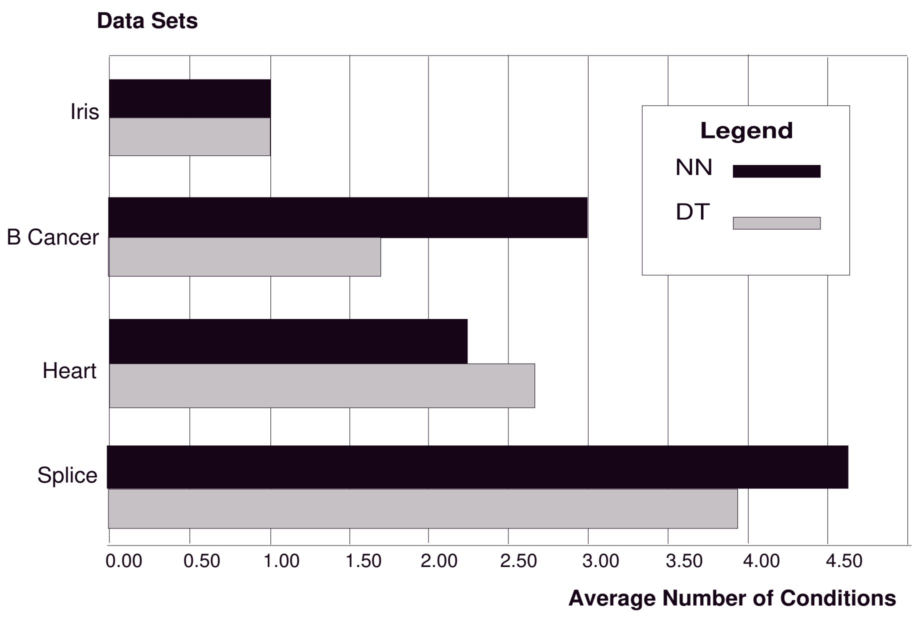

Figure 3: Comparison: average number of a rule's conditions for the four datasets.

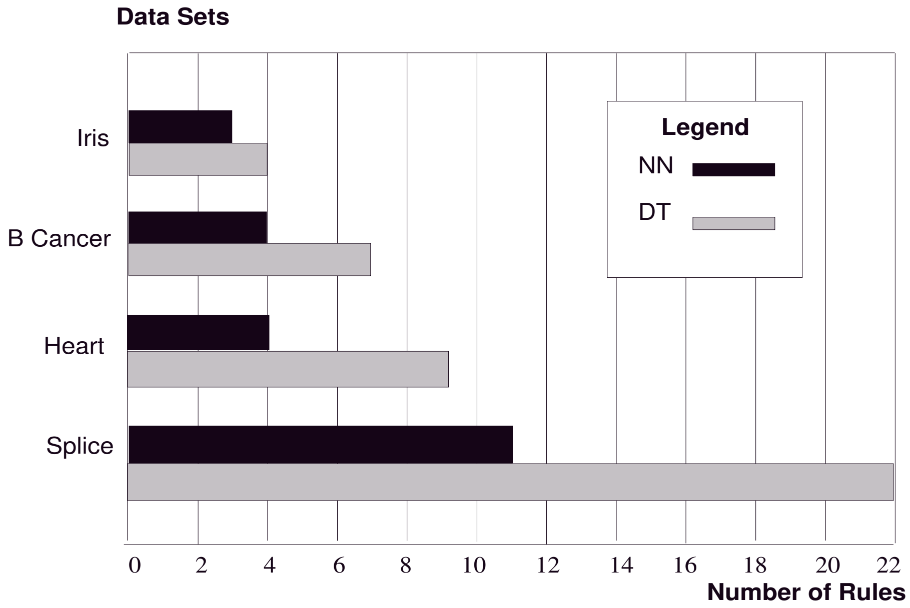

Figure 4: Comparison: number of rules for the four datasets.

4 Discussion

The comparisons made in Figures 2-4 indicate that NN rules are comparable with, if not better than, DT rules in terms of our understanding measures. The average number of conditions in NN rules is larger than that of DT for 2 of the 4 problems tested, however, the total number of NN rules is less than DT rules for all the 4 problems. These observations are consistent with the nature of each learning algorithm, i.e., parallel vs. sequential. Other issues of interests are:

- *The training time.* It takes much longer time to train a neural network than to learn a decision tree. This is also true for NN rules and DT rules extraction. Due to the existence of sequential and parallel data types, and decision trees and neural networks are best suited to one type only,[2] the two approaches are expected to coexist. When time is really scarce, the decision tree approach should be taken. Otherwise, it is worthwhile trying both because of backpropagation's other advantages (generalizing better on a smaller dataset, predicting better in general, etc. [6])

- *Average performance of NN rules.* Because of neural networks' nondeterministic nature, it is not uncommon that many runs of networks are needed with different initial weights. As was shown in Table 2, the average performance for 100 pruned networks is very impressive (94.55%). This displays the robustness of the pruning algorithm.

- *Accuracy of neural networks and NN rules.* There is a trade-off between the accuracy of the the rules extracted from the network and the complexity of the rules. A network can be further pruned and simpler rules obtained at the cost of sacrificing its accuracy. A notable feature of our rule extraction algorithm is that while it allows us to extract rules with the same accuracy level as that of the pruned network, it is also possible to simplify the rules by considering a smaller number of hidden unit activation values.

- *Understanding the weights of connections.* Unlike M-of-N rules,[6] NN rules here reflect precisely how the network works. NN rules given here are actually the merge of the two sets: 1. from the input layer to the hidden layer; and 2. from the hidden layer to the output layer. NN rules cover all the possible combinations of the connections with various input values and discrete activation values of hidden units. This is a significant improvement over search-based methods[4,6] where all possible input combinations are searched for subsets that will exceed the bias on a unit. To reduce the cost of searching, they normally limit the number of antecedents in extracted rules. Our algorithm imposes no such limit.

- *Consistency between NN and DT rules.* Consistency checking is not an easy task. In general, the possible rule space is very large since the training data is only a sample of the world under consideration. It is not surprising that there exist many equally good rule sets. Using the binary code for the Iris data, for example, the possible size of the rule space is 2^{38}, but there are only 75 examples for training. However, for simple problem like the Iris problem, the rules extracted by NN and the rules generated by DT are remarkably similar.

5 Conclusion

Neural networks have been considered black boxes. In this paper, we propose to understand a network by rules extracted from it. We describe NeuroRule, an algorithm that can extract rules from a *standard* feedforward neural network. Network training and pruning is done via the simple and widely-used backpropagation method. No restriction is imposed on the activation values of the hidden units or output units. Extracted rules are a one-to-one mapping of the network. They are compact and comprehensible, and do not involve any weight values. The accuracy of the rules from a pruned network is as high as the accuracy of the network. Experiments show that NN rules and DT rules are quite comparable. Since DT rules are regarded as explicit and understandable, we conclude that NN rules are likewise. With the rules extracted by the method introduced here, neural networks should no longer be regarded as black boxes.

References

1. T.G. Dietterich, H. Hild, and G. Bakiri, "A Comparative Study of ID3 and Backpropagation for English Text-to-speech Mapping," *Proc. of the Seventh Int'l Conf. on Machine Learning.* University of Texas, Austin, Texas, 1990, pp. 24–31.

2. J.R. Quinlan, "Comparing Connectionist and Symbolic Learning Methods," In S.J. Hanson, G.A. Drastall, and R.L. Rivest, editors, *Computational Learning Therory and Natural Learning Systems*, A Bradford Book, The MIT Press, 1994, pp. 445–456. 1.

3. J.W. Shavlik, R.J. Mooney, and G.G. Towell, "Symbolic and Neural Learning Algorithms: An Experimental Comparison," *Machine Learning*, Vol. 6, No. 2, 1991, pp. 111–143.

4. L. Fu, *Neural Networks in Computer Intelligence.* McGraw-Hill, New York, 1994.

5. K. Saito and R. Nakano, "Medical Diagnostic Expert System Based on PDP Model," In *Proc. of IEEE Int'l Conf. on Neural Networks*, IEEE Press, 1988, pp. 255–262. 1.

6. G.G. Towell and J.W. Shavlik, "Extracting Refined Rules from Knowledge-based Neural Networks," *Machine Learning*, Vol. 13, No. 1, 1993, pp. 71–1013.

7. J.R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

8. R. Kerber, "Chimerge: Discretization of Numeric Attributes," *AAAI-92, Proc. Ninth National Conf. on Artificial Intelligence*, AAAI Press/The MIT Press, 1992, pp. 123–128.

9. M. Smith, *Neural networks for Statistical Modeling.* Van Nostrand Reinhold, 1993.

10. J. Cheng, U.M. Fayyad, K.B. Irani, and Z Qian, "Improved Decision Trees: A Generalized Version of ID3," *Proc. of the Fifth Int'l Conf. on Machine Learning*, Morgan Kaufman, 1988, pp.100–106.