Draft

Symbolic Representation of Neural Networks

Rudy Setiono and Huan Liu
Department of Information Systems and Computer Science
National University of Singapore
Kent Ridge, Singapore 0511
{rudys,liuh}@iscs.nus.sg

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 drives a symbolic representation for neural networks to make explicit each prediction of a neural network. An algorithm is proposed and implemented to extract symbolic rules from neural networks. Explicitness of the extracted rules is supported by comparing 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 nerual networks preserve high predictive accuracy of original networks.

An early and shorter version of this paper has been accepted for presentation at IJCAI'95.

1 Introduction

Researchers [2, 3, 13, 17, 21] 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 [13]; and (3) symbolic learning (decision trees induction) can produce interpretable rules while networks of weights are harder to interpret [17]. 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 [5, 15, 19] have shown that rules can be extracted from networks. These algorithms are search-based methods. In order to reduce their exponential complexity in search time, only 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 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 a symbolic representation of it, or extracting rules from it. The proposed algorithm (NNRE - neural network rule extraction) consists of four parts: first, a weight-decay backpropagation network is built so that important connections are reflected by their bigger weights; second, the network is pruned such that insignificant connections are deleted while its predictive accuracy is still maintained; third, the hidden unit activation values are discretized, data are reconstructed according to the discretized activation values; and last, rules are extracted. By drawing parallels with the rules generated from decision trees, we show that networks can be interpreted by the extracted rules; the rules in general preserve the accuracy of the networks; and they also explain how a prediction is made.

In the following section, we describe algorithm NNRE 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 it is meant by understandability. A general discussion is provided in Section 4. The conclusion is given in the the final section.

2 Algorithm for Neural Network 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 then extracted from the data with discretized activation values of the hidden units. The algorithm NNRE 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 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 input units that are irrelevant to the class classification. After pruning, irrelevant input and hidden units can be identified and removed 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

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

where

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

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 [14] is applied to update the weights (w, v) and minimize the following error 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 the penalty function used for weight decay [6, 8]:

$$P(w,v) = \epsilon_1\left(\sum_{m=1}^{h}\sum_{\ell=1}^{n}\frac{\beta(w_\ell^m)^2}{1+\beta(w_\ell^m)^2} + \sum_{m=1}^{h}\sum_{p=1}^{o}\frac{\beta(v_p^m)^2}{1+\beta(v_p^m)^2}\right) + \tag{5}$$
$$\epsilon_2\left(\sum_{m=1}^{h}\sum_{\ell=1}^{n}(w_\ell^m)^2 + \sum_{m=1}^{h}\sum_{p=1}^{o}\left(v_p^m\right)^2\right).$$

Using this penalty function, we have been able to obtain networks with less number of connections than the previously reported for many learning problems. Details of the comparison and the choice of the parameter values (ϵ_1, ϵ_2, and β) are given in [16].

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 (7 and 8). 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, it is therefore imperative that the weights 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. The details of this pruning algorithm and experimental results obtained are described in [16].

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 this condition (6) is satisfied:

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

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

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

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

   then remove w_ℓ^m from the network

4. For each v^m, if
$$\max_p |v_p^m| \leq 4\eta_2, \tag{8}$$
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 $\max_p |v_p^m \times w_\ell^m|$.

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

2.3 Activation values discretization

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

Discretizing Hidden Unit Activation Values by Clustering
For each hidden unit,

1. Let $\epsilon \in (0, 1)$,

2. Start from 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 very small ϵ can guarantee that the network with discretized activation values is as accurate as the original network with continuous activation values. So if its accuracy rate does not drop and there are still many discrete values, clustering can be performed again with a larger ϵ. Otherwise, ϵ should be reduced to a smaller value.

The clustered data used for generating rules from input layer I to hidden layer H is reconstructed from the training data as follows: one data set corresponds to each hidden unit; each cluster of a hidden unit forms a class. That is, if there are m hidden units in the pruned network, there exist m such data sets. The number of classes for each data set is determined by the number of clusters of the corresponding hidden unit. The possible combinations of all the classes of all hidden units are constrained by the pruned network from hidden layer H to output layer O and used to compose the data for generating rules from H to O. The clustered data (total of $m + 1$ data sets) produced by this algorithm does not have any inconsistency in the sense that identical patterns only belong to one cluster.

2.4 Rule Extraction

After network pruning and activation value discretization, rules can be extracted by the following algorithm (**RE**).

Rule Extraction (RE)

1. Generate rules R_{i-h} for each hidden unit with **RG**.
2. Generate rules R_{h-o} for output with **RG**.
3. Substitute R_{h-o} with R_{i-h} to obtain rules R_{i-o}.
4. Resolve the conflicts among the rules R_{i-o}.

Rule Generation (RG)

1. Group the data into D_p in terms of class values, $p \in \{1,2,...,o\}$.
2. 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 a rule R_k; and
   - Remove all the items in D_c that are covered by rule R_k, $k++$.
3. If all D_p have been considered, stop; otherwise go to Step 2.

Rules R_{i-h} for each hidden unit describe the relationships from the input units to the hidden unit; rules R_{h-o} for from the hidden layer H to the output layer O describe the relationships from the hidden unit to the output unit. The need for conflict resolution (Step 4) arises because (1) some combinations are the impossible connections form I to H, to O in the network, and (2) of **RG**'s generalization in rule induction. The rule generation algorithm **RG** produces perfect rules, i.e., the error rate of the rules is no worse than the inconsistency rate δ in the data (refer to [9] for the details); in the context of this work, the clustered data does not have any inconsistency and δ is 0, so its error rate is 0.

2.5 Recursive application of NNRE

However, when there are still too many connections (e.g., more than 7) between a hidden unit and input units, the extracted rules may not be easy to understand. Another three layer feedforward subnetwork may be employed to simplify rule extraction for the hidden unit. This subnetwork is trained in the same ways as is the original network, but in a reduced scale: the number of output units is the number of discrete values of the hidden unit, while the input units are those connected to the hidden unit in the original network. Examples are grouped according to their discretized activation values. Given d discrete activation values $D_1, D_2, \ldots, D_d$, all examples with activation values equal to D_j are given a d-dimensional target value of all zeros except for one 1 in position j. A new hidden layer is introduced for this subnetwork and it is then trained, pruned, and the activation values of its hidden units are discretized for rule extraction. If necessary, another subnetwork is created until the number of connection is small enough or the new subnetwork cannot simplify the connections between the inputs and the hidden unit at the higher level. However, the more subnetworks are introduced, the more layers of rules there will be, since each network has two layers of rules. The chains of rules could be long. The creation of subnetworks is rarely needed. Only if we are not satisfied with rules with many conditions do we consider this choice. For example, as an experiment, subnetworks were created for the Splice-junction problem since some rules had had more than 7 conditions.

3 Experiments and Results

In this section, we describe the datasets and representations used in experiments. A detailed example is given to show how the NNRE algorithm is applied to extracting rules. Summary of the results on all datasets are given with a comparison to those produced by the decision tree induction methods. Understanding a neural network is achieved by having a symbolic representation that is able to explain, based on the rules, how each prediction is made, as in understanding a decision tree by having rules generated from it [12].

3.1 Datasets and Representations

Three datasets used are: 1. Iris – a classic dataset introduced by R.A. Fisher [4]; 2. Breast Cancer – a widely tested real-world dataset for the Wisconsin Breast Cancer diagnosis; and 3. Splice-junction – a dataset used in splice-junction determination originally described by Noordewier et al [11]. The datasets are obtainable from the University of California Irvine data repository for machine learning (via anonymous ftp from ics.uci.edu). Given below is the summary of these datasets, their representations, and how each dataset is used in experiments.

- *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, and sepal-width, petal-length, petal-width. Since each attribute takes a continuous number, the Chi2 algorithm [10], a generalized version of ChiMerge by Kerber [7], is applied to discretize attribute values. The thermometer code [18] 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. (Refer to the details in Tables 6-9 in Appendix.) With 1 input for bias, there are total 39 inputs and three outputs. Examples in odd positions in the original dataset form the training set and the rest are for testing as was done in [5].

- *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 were 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 attribute takes an ordinal integer from 1 to 10 (10 values) except for attribute 6 which has an extra value, 0. Due to the ordinal nature, the thermometer code is used again to code each attribute value. Ten inputs correspond to 10 values of each attribute with all 10 inputs *on* representing value 10, the rightmost input *on* for value 1, and the two rightmost inputs *on* for value 2, etc. In case of value 0, none of the inputs is *on*. With one input for bias, in total there are 91 inputs and 2 outputs for the original network.

- *Splice-junction* - the dataset contains 3175 examples[1] 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 also 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 experiments presented here, 1006 examples were randomly chosen as training data from the 3175 examples following other researchers' way [19], the rest used as testing data.

3.2 A Detailed Example – Iris Data Classification

This example shows in detail how rules are extracted from a pruned network. In the experiment, 100 fully connected neural networks were used as the starting networks. Each of these networks consists of 39 input units, 3 hidden units and 3 output units. These networks were trained with initial weights that had

[1] Another 15 examples in the original dataset contain invalid values so these examples are not included in experiment.

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

Table 1: Average number of connections and accuracy of 100 pruned networks on the training and testing data sets and their standard deviations.

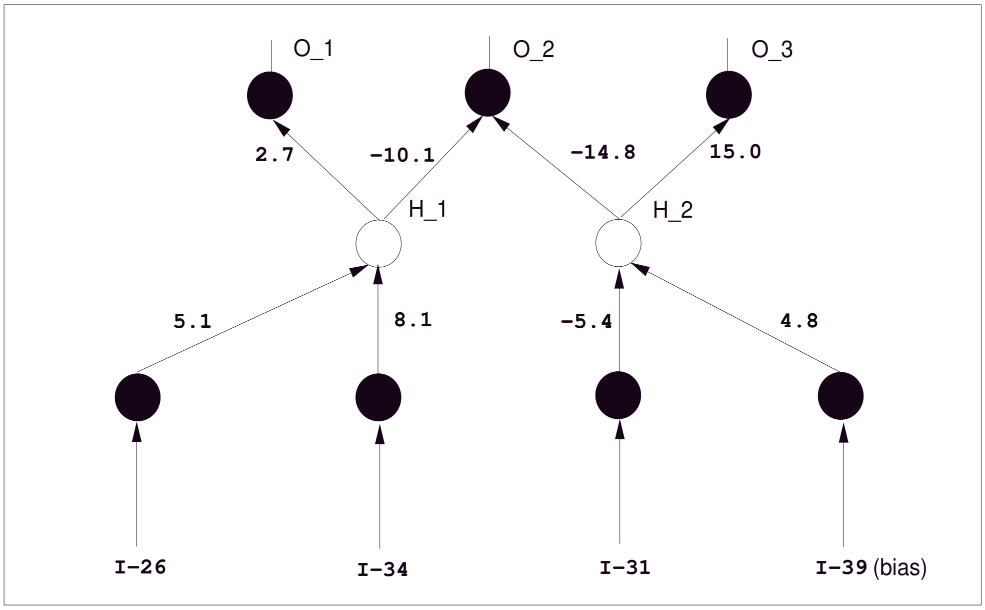

Figure 1: Pruned network with only 8 connections, accuracy on training set = 98.67 %, accuracy on the testing set = 97.33 %. Figure next to a connection indicates the weight of that connection.

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 1 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 are obtained from 100 pruned networks, all of which have accuracy rates on the training data of at least 95 %. In the third column, the figures are obtained from 100 pruned networks with accuracy of at least 97 % on the training data.

One of the smallest pruned networks is depicted in Figure 1. It has only 2 hidden units and a total of 8 connections with 98.67% accuracy on the training set and 97.33% on the testing set. We ran the clustering algorithm of Section 2.3 on this network and found only 2 discrete values are needed at each of the

Hidden unit activations		Predicted output			Classification
H_1	H_2	O_1	O_2	O_3	
0	1	0.5	0	1	*Iris setosa*
0	-0.5	0.5	1	0	*Iris versicolor*
1	1	0.9	0	1	*Iris setosa*
1	-0.5	0.9	0	0	*Iris virginica*

R_{h-o}:
If H_2 = 1 Then *Iris setosa*
If H_1 = 0 and H_2 = -0.5 Then *versicolor*
Else *Iris viginica*

Table 2: (a) Discretized hidden unit activation values and predicted outputs of the network in Figure 1; (b) Rules R_{h-o}.

I_31	I_39	H_2
0	1	1
1	1	-0.5

R_{i-h2}:
If I_31 = 0 Then H_2 = 1
Else H_2 = -0.5

Table 3: (a) From input values to hidden unit activation values; (b) Rules R_{i-h2}.

two hidden units to maintain the same level of accuracy on the training data. At hidden unit 1, 48 of 75 training examples have activation values equal to 0 and the remaining 27 have activation values equal to 1. At hidden unit 2, the activation value of 25 examples is 1 and the activation value of the remaining 50 examples is -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 2. From this table, it is clear that an example will be classified as *Iris setosa* as long as its activation value at the second hidden unit is equal to 1. Otherwise, the example is classified as *Iris versicolor* provided that its first hidden unit activation value H_1 = 0. The default class will then be *Iris virginica*. Rules $R_h h - o$ in Table 2 summarize these findings.

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 not difficult 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). It is summarized in Table 3.

The activation value of the first hidden unit, H_1, depends only on I_26

I_26	I_34	H_1
0	0	0
0	1	1
1	0	1
1	1	1

R_{i-h1}:
If I_26 = 0 and I_34 = 0
Then H_1 = 0
Else H_1 = 1

Table 4: (a) From input values to hidden unit activation values; (b) Rules R_{i-h1}.

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. Therefore, an example with I_31 = 1, I_26 = I_34 = 0 will be classified as *Iris versicolor*. It is summarized in Table 4.

The clustering algorithm produces three data sets: one set as in Table 2, and two sets for 2 hidden units as in Tables 3 and 4. **RE** then extracts rules from them. Referring to the coding scheme for the input data in Appendix, 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:

NN Rules[2]
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 generates more concise rules than the tree itself) is included here:

DT Rules
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 neural networks and decision trees (NN rules vs. DT rules). We hope to establish that obtaining NN rules is equivalent to having a symbolic representation for neural networks. Its *understandability* is partly defined as being explicable 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

[2]The rules are fired in a top-down and left-to-right fashion.

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

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

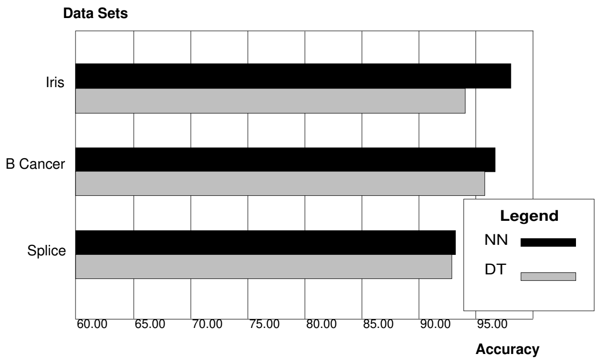

Figure 2: Comparison: predictive accuracy for the three datasets.

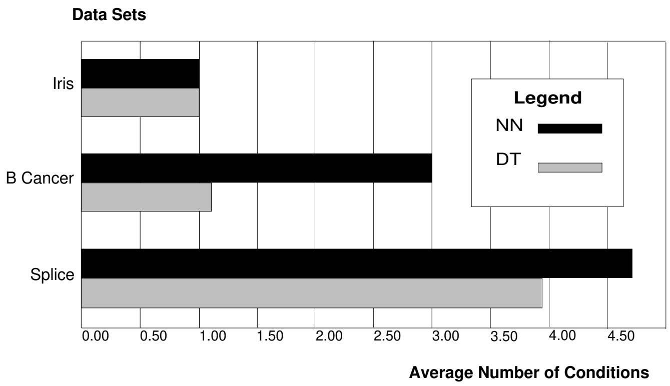

Figure 3: Comparison: average number of a rule's conditions for the three datasets.

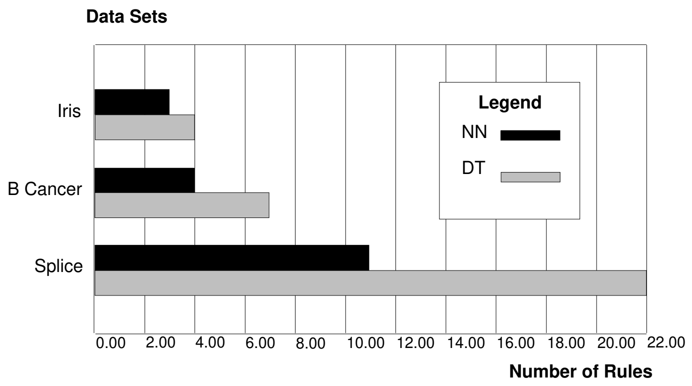

Figure 4: Comparison: number of rules for the three datasets.

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=V_j", i.e., attribute A_i takes value 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, it can be explained how the prediction is attained. If necessary, the intermediate process can also be explicitly explained by rules R_{i-h} and R_{h-o}.

C4.5 and C4.5rules [12] were run on the above three datasets to generate DT rules. Briefly, C4.5 generates a decision tree which C4.5rules generalizes to rules. Since researchers [1, 17, 20] observed that mapping many-valued variables to two-valued variables results in decision trees with higher classification accuracy[3], 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.

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 higher than that of DT for 2 of the 3 problems tested, however, the total number of NN rules is less than DT rules for all the 3 problems. These observations are consistent with the nature of each learning algorithm, i.e., parallel vs. sequential.

In the detailed example (Section 3.2), the NN and DT rules are shown for the Iris data. We show here the NN and DT rules for the breast cancer data [4]. Let A_1 be clump thickness, A_2 uniformity of cell size, A_3 uniformity of cell

[3]It's true indeed for the three datasets in our experiments.

[4]We leave out the rules for the Splice data due to its large size. They can be obtained from the authors.

shape, A_4 marginal adhesion, A_5 single epithelial cell size, A_6 bare nuclei, A_7 bland chromatin, A_8 normal nucleoli, and A_9 mitosis, we have:

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

Rule 1: If $A_1 < 7$ and $A_2 < 8$ and $A_3 < 3$ and $A_8 < 9$ then *benign*
Rule 2: If $A_1 < 7$ and $A_2 < 8$ and $A_6 < 3$ and $A_8 < 9$ then *benign*
Rule 3: If $A_2 < 8$ and $A_3 < 3$ and $A_6 < 3$ and $A_8 < 9$ then *benign*
Default Rule: *malignant.*

<u>**DT Rules**</u>

Rule 1: If $A_1 < 7$ and $A_2 < 3$ then *benign*
Rule 2: If $A_1 < 7$ and $A_4 < 2$ then *benign*
Rule 3: If $A_2 \geq 5$ then *malignant*
Rule 4: If $A_6 \geq 9$ then *malignant*
Rule 5: If $A_1 \geq 7$ then *malignant*
Rule 6: If $A_4 \geq 4$ 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 emphasize the use of parallel features: each rule consists of as many as four attributes; and DT rules focus on individual features: most rules have one attribute. What is particularly interesting here is that there are fewer NN rules than DT rules, although NN rules have more conditions/attributes (hence more specific) and DT rules have fewer conditions/attributes (hence more general)[5]. The possible explanation is that NN rules really explore the coordinated effects of the attributes.

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.

4 Discussion

Discussed here are other issues of interests:

- *The training time.* As was mentioned in many papers [2, 3, 13, 17, 21] doing comparison between backpropagation and decision tree learning, 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 [13], the two approaches are expected to coexist. When time is really scarce,

[5]This runs counter to our intuition. Normally, fewer general rules are needed to cover a data set than specific rules for the same data, other things (e.g., accuracy) being equal.

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. [19]).

- *Average performance of NN rules.* Because of neural networks' nondeterministic nature, it is not uncommon that hundreds runs of networks are needed with different initial weights. As was shown in Table 1, the average performance for 100 pruned networks is very impressive (94.55%). This displays the robustness of the presented 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 [19], 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 [19, 5] 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. In the example of Breast Cancer Data, the two rule sets are not alike. Which set is more helpful in explaining the cause of cancer or makes more sense will have to rely on the judgment of field experts. The symbolic representation of a neural network, however, opens an avenue in search of new knowledge and makes the comparison between the three parties (NN, DT and human) possible.

- *Undeterministic vs. deterministic.* By varying the seeds for neural network training, different networks are constructed, so are various NN rules. The large representation space allows the possible extraction of many equally good NN rules. Relatively speaking, DT rules are deterministic due to the nature of tree construction. If there is a need for exploring different rule sets, rules extraction based on neural networks such as NNRE would be an ideal choice.

5 Conclusion

Neural networks have been considered as black boxes. In this paper, we propose to symbolically represent a backpropagation neural network by extracting rules from it so that a prediction made can be explained. We describe the NNRE 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. Through discretization of hidden unit activation values, rules extracted 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 of backpropagation should no longer be black boxes.

References

[1] J. Cheng, U.M. Fayyad, K.B. Irani, and Z Qian. Improved decision trees: A generalized version of id3. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 100–106. Morgan Kaufman, 1988.

[2] T.G. Dietterich, H. Hild, and G. Bakiri. A comparative study of id3 and backpropagation for english text-to-speech mapping. In *Machine Learning: Proceedings of the Seventh International Conference*. University of Texas, Austin, Texas, 1990.

[3] D.H. Fisher and K.B. McKusick. An empirical comparison of id3 and back-propagation. In *Proceedings of 11th International Joint Conference on Artificial Intelligence*, pages 788–793, 1989.

[4] R.A. Fisher. The use of multiple measurements in taxonomic problems. *Ann. Eugenics*, 7(2):179–188, 1936.

[5] L. Fu. *Neural Networks in Computer Intelligence*. McGraw-Hill, 1994.

[6] B. Hassibi and D.G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Neural Information Processing Systems*, 5:164–171, 1993.

[7] R. Kerber. Chimerge: Discretization of numeric attributes. In *AAAI-92, Proceedings Ninth National Conference on Artificial Intelligence*, pages 123–128. AAAI Press/The MIT Press, 1992.

[8] Y. Le Cun, J.S. Denker, and S.A. Solla. Opptimal brain damage. *Neural Information Processing Systems*, 2:598–605, 1990.

[9] H. Liu. Generating perfect rules. Technical report, Department of Info Sys and Comp Sci, National University of Singapore, February 1995.

[10] H. Liu and R. Setiono. Chi2: Discretization of ordinal variables and feature selection. Technical report, Department of Info Sys and Comp Sci, National University of Singapore, January 1995.

[11] M.O. Noordewieer, G.G. Towell, and J.W. Shavlik. Training knowledge-based neural networks to recognize genes in dna sequences. In *Advances in neural information processing systems*, volume 3, pages 530–536. Morgan Kaufmann, 1991.

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

[13] 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*, volume 1, pages 445–456. A Bradford Book, The MIT Press, 1994.

[14] D.E. Rumelhart, McClelland J.L., and the PDP Rearch Group. *Parallel Distributed Processing*, volume 1. Cambridge, Mass. The MIT Press, 1986.

[15] K. Saito and R Nakano. Medical diagnostic expert system based on pdp model. In *Proceedings of IEEE International Conference on Neural Networks*, volume 1, pages 255–262. IEEE, 1988.

[16] R. Setiono. A penalty function approach for pruning feedforward neural networks. *Technical Report, DISCS, National University of Singapore*, 1995.

[17] J.W. Shavlik, R.J. Mooney, and G.G. Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6(2):111–143, 1991.

[18] M. Smith. *Neural networks for Statistical Modeling*. Van Nostrand Reinhold, 1993.

[19] G.G. Towell and J.W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101, 1993.

[20] P.E. Utgoff and C.E. Brodley. An incremental method for finding multivariate splits for decision trees. In *Machine Learning: Proceedings of the Seventh International Conference*, pages 58–65. University of Texas, Austin, Texas, 1990.

[21] S.M. Weiss and I. Kapouleas. An empirical comparison of pattern recognition, neural nets, and machine learning classifcation methods. In *Proceedings of 11th International Joint Conference on Artificial Intelligence*, pages 781–787, 1989.

Appendix

The Chi2 algorithm [10] is employed to discretize attribute values. A 0.30 significance level is determined by Chi2 that corresponds to a low χ^2 threshold value of 0.70. That means when other significance levels, such as 0.20 or 0.10 are chosen, some examples that belong to different classes are discretized into identical examples. This conflict in class labeling hinders a neural network from getting a high accuracy.

Using 75 examples in one of the training set, the attributes of the data are discretized by the ChiMerge algorithm as follows. The sepal length, which ranges from 4.3 cm to 7.9 cm is divided into 16 subintervals. The subintervals and the coding scheme for these subintervals are depicted in Table 1. Tables 2 to 4 show the results on sepal width, sepal length and petal width, respectively.

Interval	Coding $I1 - I16$	Interval	Coding $I1 - I16$
$\mathcal{A}_1 \leq 4.8$	00000 00000 000001	$5.7 < \mathcal{A}_1 \leq 5.8$	00000 00111 111111
$4.8 < \mathcal{A}_1 \leq 4.9$	00000 00000 000011	$5.8 < \mathcal{A}_1 \leq 6.0$	00000 01111 111111
$4.9 < \mathcal{A}_1 \leq 5.1$	00000 00000 000111	$6.0 < \mathcal{A}_1 \leq 6.5$	00000 11111 111111
$5.1 < \mathcal{A}_1 \leq 5.3$	00000 00000 001111	$6.5 < \mathcal{A}_1 \leq 6.6$	00001 11111 111111
$5.3 < \mathcal{A}_1 \leq 5.4$	00000 00000 011111	$6.6 < \mathcal{A}_1 \leq 6.7$	00011 11111 111111
$5.4 < \mathcal{A}_1 \leq 5.5$	00000 00000 111111	$6.7 < \mathcal{A}_1 \leq 6.9$	00111 11111 111111
$5.5 < \mathcal{A}_1 \leq 5.6$	00000 00001 111111	$6.9 < \mathcal{A}_1 \leq 7.0$	01111 11111 111111
$5.6 < \mathcal{A}_1 \leq 5.7$	00000 00011 111111	$7.0 < \mathcal{A}_1$	11111 11111 111111

Table 6: The subintervals created by the ChiMerge algorithm and their binary coding for $\mathcal{A}_1$ – sepal length

Interval	Coding $I17-I25$	Interval	Coding $I17-I25$
$\mathcal{A}_2 \leq 2.4$	0000 00001	$3.0 < \mathcal{A}_2 \leq 3.2$	0001 11111
$2.4 < \mathcal{A}_2 \leq 2.7$	0000 00011	$3.2 < \mathcal{A}_2 \leq 3.3$	0011 11111
$2.7 < \mathcal{A}_2 \leq 2.8$	0000 00111	$3.3 < \mathcal{A}_2 \leq 3.4$	0111 11111
$2.8 < \mathcal{A}_2 \leq 2.9$	0000 01111	$3.4 < \mathcal{A}_2$	1111 11111
$2.9 < \mathcal{A}_2 \leq 3.0$	0000 11111		

Table 7: The subintervals created by the ChiMerge algorithm and their binary coding for $\mathcal{A}_2$ – sepal width

Interval	Coding $I26-I32$	Interval	Coding $I26-I32$
$\mathcal{A}_3 \leq 1.9$	0000001	$4.7 < \mathcal{A}_3 \leq 4.8$	0011111
$1.9 < \mathcal{A}_3 \leq 4.4$	0000011	$4.8 < \mathcal{A}_3 \leq 4.9$	0111111
$4.4 < \mathcal{A}_3 \leq 4.5$	0000111	$4.9 < \mathcal{A}_3$	1111111
$4.5 < \mathcal{A}_3 \leq 4.7$	0001111		

Table 8: The subintervals created by the ChiMerge algorithm and their binary coding for $\mathcal{A}_3$ – petal length

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 9: The subintervals created by the ChiMerge algorithm and their binary coding for $\mathcal{A}_4$ – petal width