On Mapping Decision Trees and Neural Networks

Rudy Setiono and Wee Kheng Leow

School of Computing, National University of Singapore

Lower Kent Ridge Road, Singapore 119260, Singapore

phone: (65) 874-6297, fax: (65) 779-4580, email: {rudys, leowwk}@comp.nus.edu.sg

Abstract

There exist several methods for transforming decision trees to neural networks. These methods typically construct the networks by directly mapping decision nodes or rules to the neural units. As a result, the networks constructed are often larger than necessary. This paper describes a pruning-based method for mapping decision trees to neural networks, which can compress the network by removing unimportant and redundant units and connections. In addition, equivalent decision trees extracted from the pruned networks are simpler than those induced by well-known algorithms such as ID3 and C4.5.

Keywords: decision trees, neural networks, pruning.

1 Introduction

Decision trees have been widely used for nonparametric pattern classification tasks which involve several pattern classes and a large number of features. Given an input pattern, the tree classifier performs the comparisons stipulated in each decision node of the tree, and then branches to either the left or the right subtree based on the comparison results. Since the comparisons are made between the pattern's feature values and specific constant thresholds, the classification result is very sensitive to noise in feature measurements.

To minimise the sensitivity of tree classifiers to noise, several researchers have explored the use of soft decisions based on probabilities associated to the outcomes of the comparisons in the decision nodes [1, 5, 6]. Another approach is to transform decision trees into equivalent neural networks [2, 3, 7, 8, 11]. Ivanova and

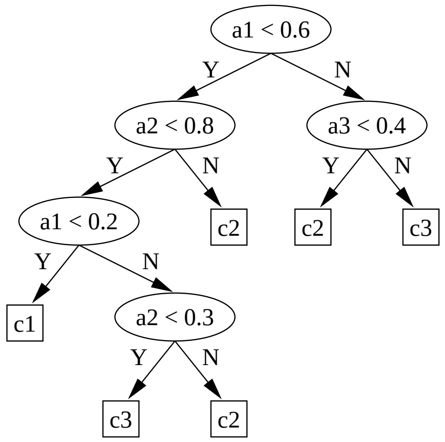

Figure 1: A sample decision tree induced by ID3. It has a depth of 4 and contains 5 decision nodes.

Kubat's Tree-Based Neural Network (TBNN) method [2] is particularly interesting because it constructs a neural network based on discrete binary decision trees and yet permits the network to be further trained on continuous-valued patterns.

Our study of the TBNN method reveals that the networks that it constructs tend to be more complex than necessary. In this paper, a new method of generating neural networks is described. This method, called *Pruning-Based Neural Network* (PBNN), is based on the pruning of networks trained on binary patterns generated from decision trees. The PBNN method not only generates simpler networks, but also simplifies the equivalent decision trees in the process of pruning the networks.

In the remainder of this paper, the TBNN method will first be briefly described in Section 2. Our PBNN method will be described in Section 3. Examples illustrating the strength of our method are presented in Sections 3 and 4. Finally, concluding remarks will be given in Section 5.

2 Tree-Based Neural Network

An example helps to illustrate how Ivanova and Kubat's TBNN method [2] constructs a neural network from a decision tree. Consider, for example, the decision tree shown in Fig. 1, which is induced from continuous-valued patterns using the ID3 algorithm [4]. The tree contains three attributes $a1$, $a2$, and $a3$, and three classes $c1$, $c2$, and $c3$. The attributes values can be divided into intervals (Fig. 2) based on the threshold values given in the decision nodes of the tree. Each interval is assigned a Boolean variable from a to h.

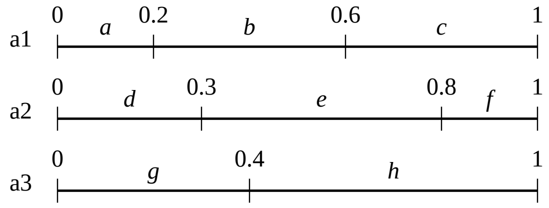

Figure 2: Attribute intervals deduced from the decision tree in Fig. 1. Each interval is assigned a label from a to h.

The Boolean variables take the value 1 if the attribute values fall inside the corresponding intervals, and 0 otherwise. Given the intervals, the decision tree can be re-written into the following rules in disjunctive normal form (DNF):

$$
\begin{aligned}
&(a \wedge \neg f) \Rightarrow c1 \\
&(b \wedge e) \vee (\neg c \wedge f) \vee (c \wedge g) \Rightarrow c2 \\
&(b \wedge d) \vee (c \wedge h) \Rightarrow c3
\end{aligned}
\tag{1}
$$

Now, the TBNN is constructed as follows: Assign an input unit to each Boolean variable a to h. Use a hidden unit to implement each conjunctive term in the rules. To each hidden unit, insert connections from the input units which represent Boolean variables in the conjunctive term: use positive connections for non-negated variables and negative connections for negated variables. Assign an output unit to each class $c1$ to $c3$. These output units implement the disjunctions between conjunctive terms. To each output unit, insert a positive connection from the hidden units which represent the conjunctive terms in the corresponding rule. The actual values of the connection weights and the units' biases are selected such that the hidden units implement logical AND and the output units implement logical OR. The connections between other units carry small weights. Figure 3 shows the TBNN constructed in this manner.

In this network, there are as many input units as there are intervals, and the number of hidden units equals the number of conjunctive terms in the rules. However, close examination of the rules (Eq. 1) reveals that there are redundancies in the Boolean variables, e.g., $c = \neg a \wedge \neg b$. By removing the redundancies, it is possible to simplify the rules and the neural network. In the next section, it will be shown that our Pruning-Based Neural Network method not only produces simple network, but also automatically simplifies the equivalent decision tree.

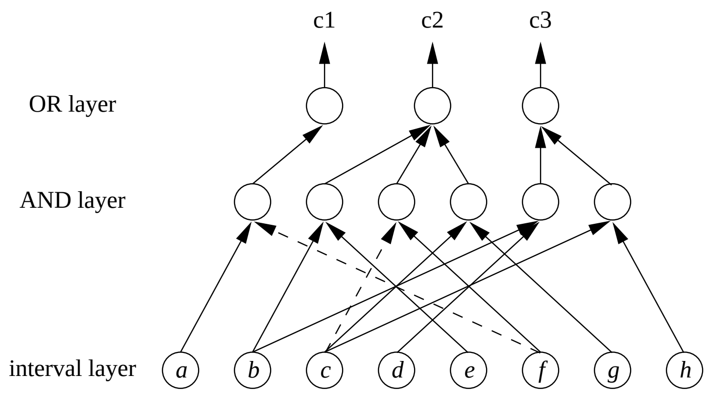

Figure 3: The neural network constructed from the rules given in Eq. 1. Solid lines represent positive connections and dashed lines denote negative connections. The connections between other units carry small weights and are not shown in the figure.

3 Pruning-Based Neural Network

Our Pruning-Based Neural Network (PBNN) method constructs a neural network from a decision tree in the following steps:

1. Derive attribute intervals from the decision tree.

2. Derive decision rules from the decision tree and the attribute intervals.

3. Generate binary training patterns from the decision rules and the attribute intervals.

4. Train a neural network on these binary patterns.

5. Prune the trained network.

PBNN generates attribute intervals and decision rules from decision tree (first two steps) in a similar way as TBNN. However, it uses a different method to generate the neural network from the decision rules. Instead of mapping variables and rules directly to a neural network, PBNN generates binary training patterns from decision rules and uses these patterns to train a network (last three steps).

To understand how the training patterns are generated, let us examine the example illustrated in Section 2. Eight attribute intervals (Fig. 2) are derived from the decision tree. Each interval is assigned a Boolean variable from a to h. These Boolean variables give rise to binary training patterns of 8 inputs and

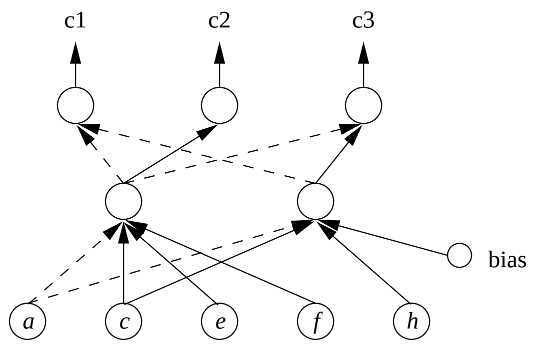

Figure 4: A pruned neural network obtained by training on all 18 possible binary patterns. A negative connection weight is indicated by a dashed line, while a positive weight by a solid line.

3 target outputs for the three classes $c1$, $c2$, and $c3$. The variables a, b, and c correspond to three non-overlapping intervals of attribute $a1$. Therefore, in any binary pattern, exactly one of these three variables takes the value 1 while the others take the value 0. That is, there are only three possible combinations for these variables. Similarly, variables d, e, and f of attribute $a2$ contribute 3 combinations, and variables g and h of $a3$ contribute 2 combinations to the binary patterns. In total, $3 \times 3 \times 2 = 18$ binary patterns can be generated from the decision rules and the attribute intervals. The target output values of the patterns are assigned according to the decision rules. Patterns that satisfy the conditions for memberships in class $c1$, $c2$, or $c3$ are given target values of $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$, respectively.

In the training stage, a fully-connected multi-layer perceptron with one hidden layer is used. The number of neural units required is quite arbitrary as long as the network has enough units. The TBNN method provides an upper bound on the number of units required in each layer. In our experiment, the network started with 8 input units, 6 hidden units, and 3 output units, as specified by TBNN. The network was trained using a quasi-Newton method [10]. After training the network, irrelevant and redundant network connections were removed by applying the Neural Network Pruning with Penalty Function (N2P2F) algorithm, which we proposed earlier and has been shown to be very effective [9]. The resulting network is shown in Fig. 4. It has only 2 hidden units and 7 connections from the input units to the hidden units. All connections from inputs b, d, and g are no longer present in the network. They were redundant because their values can be determined by those of the remaining inputs (as explained above).

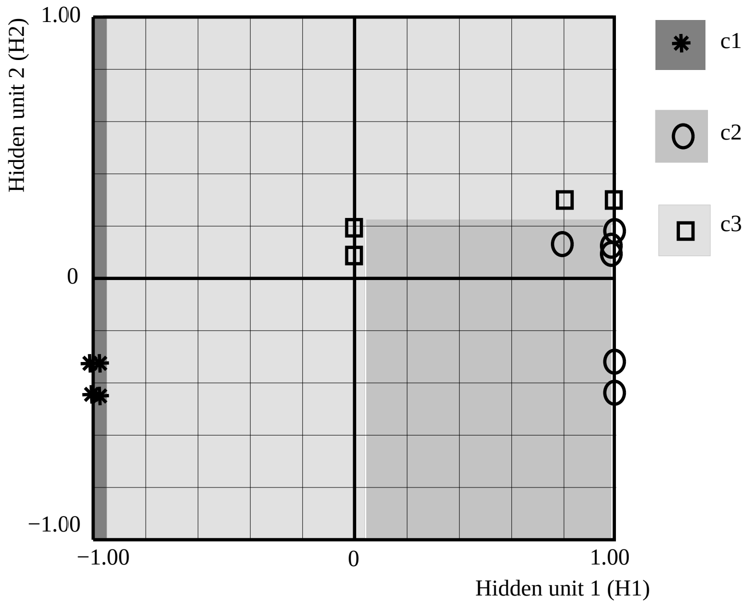

Figure 5: A scattered plot of hidden unit activations. The shaded regions represent the three classes $c1$, $c2$, and $c3$, and the symbols denote hidden unit activations in response to input patterns in the three classes.

The pruned neural network produced by PBNN has substantially fewer connections and hidden units than the network constructed by TBNN. The TBNN network uses six hidden units to implement the decision rules, one for each conjunction in the rules (Fig. 3). Each hidden unit is connected to two input units since each conjunction involves two input variables. Although mapping conjunctions to hidden units is a straightforward method of creating neural networks, the resulting networks tend to be larger than necessary, as the PBNN network in Fig. 4 shows.

To understand how PBNN network implements decision rules, we can plot its hidden unit activations in response to the input patterns. Fig. 5 shows that hidden unit 1 separates the patterns into 3 groups: those with activations $H1$ less than -0.98, between -0.98 and 0.02, and larger than 0.02. Hidden unit 2 distinguishes patterns with activation values $H2$ greater than 0.22 and those with activations less than 0.22. Thus, the network implements the following decision rules:

```
If H1 < -0.98, then class c1,
else if H2 > 0.22, then class c3,
else if H1 < 0.02, then class c3,
else class c2.
```

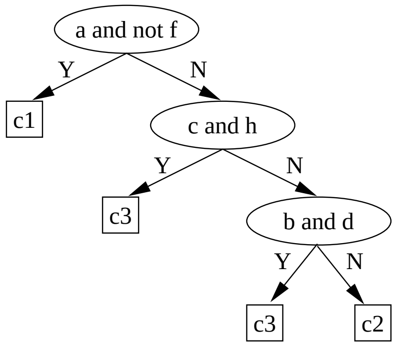

Figure 6: The decision tree of the neural network rules expressed in terms of the input variables.

The above rules are expressed in terms of hidden unit activations. Nevertheless, they can be re-written in terms of the input variables. The value of a hidden unit activation is determined by the weights of the connections from the input units to the hidden unit. The weights from input units a, c, e, and f to hidden unit 1 are $-5.73, 1.11, 2.76$, and 9.22, respectively. The weights from input units a, c, h and the bias are $-0.55, 0.10, 0.14$, and 0.07, respectively. With this information it is possible to group the input patterns according to their activation values. The knowledge of the possible input values that these patterns can have is also useful in recovering the original decision rules. For example, $H1 < -0.98$ if and only if $a = 1$ and $f = 0$. A large hidden unit 2 activation value $H2 > 0.22$ is obtained only when $a = 0$ and $c = h = 1$. Since $c = 1$ implies $a = 0$, we can conclude that $H2 > 0.22$ if and only if $c = h = 1$. In this way, the network rules can be converted from those involving hidden unit activations to those that involve the original Boolean variables. Figure 6 illustrates these rules depicted as a decision tree.

4 Injecting M-of-N Rules into Neural Networks

In addition to rules in disjunctive normal form, another very common type of rules is the M-of-N rules. These rules involve N conditions and require that at least M of these N conditions to be satisfied by a pattern. For some applications, M-of-N rules can be more intuitive and compact. For example, to decide if a credit application can be approved we require that at least 3 of the following 4 conditions to be met: 1. the amount of credit requested is not more than $25,000, 2. the applicant's monthly income is at least $3,000, 3. the applicant is not older than 55 years old, and 4. the applicant has been working in his current job for at least 2 years.

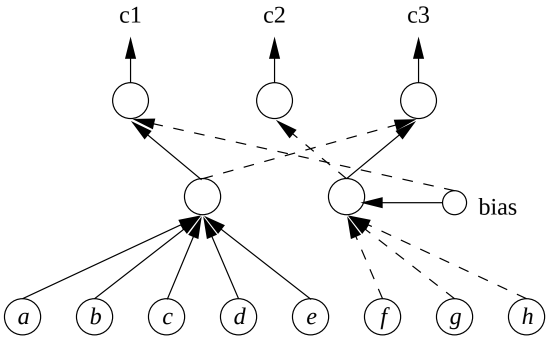

Figure 7: The PBNN network that implements the M-of-N rules.

Neural networks, due to their inherent parallel processing of the inputs can be easily constructed to implement M-of-N rules. On the other hand, since most decision tree methods check the attributes one at a time, they can be expected to generate large trees. As an illustration to compare the performance of PBNN and TBNN, we use the following example

```
if 3-of-(a, b, c, d, e), then c1,
else if 3-of-(f, g, h), then c2,
else c3.
```

In this experiment, all 2^8 patterns were generated and used to build the decision tree and generate the neural network. The decision tree generated by C4.5 – an improved version of ID3 [4] – has 63 nodes and it classifies 6 of the 2^8 patterns incorrectly. On the other hand, the neural network generated by PBNN method has only 2 hidden units (Fig. 7) and it correctly classifies all the patterns.

As in the previous example, the network's classification rules can be explained by analysing the network's hidden unit activations (Fig. 8). A hidden unit 1 activation value is defined as the hyperbolic tangent of the linear combination of the inputs $0.17a + 0.17b + 0.17c + 0.17d + 0.17e$, while the corresponding linear combination for hidden unit 2 is $-2.4f - 2.4g - 2.4h + 7.17$. Only patterns with hidden unit 1 activation activation value greater than 0.34 belong to class $c1$. Among patterns with activation values that are less than 0.34, those with negative hidden unit 2 activation values belong to class $c2$ and those with positive activation values belong to class $c3$. Let $\delta(x)$ be the hyperbolic tangent function and $\delta^{-1}(x)$ its inverse. We compute $\delta^{-1}(0.34) = 0.35$ and $\delta^{-1}(0) = 0$. The patterns can be classified as follows:

```
If 0.17 a + 0.17 b + 0.17 c + 0.17 d + 0.17 e > 0.35, then c1,
```

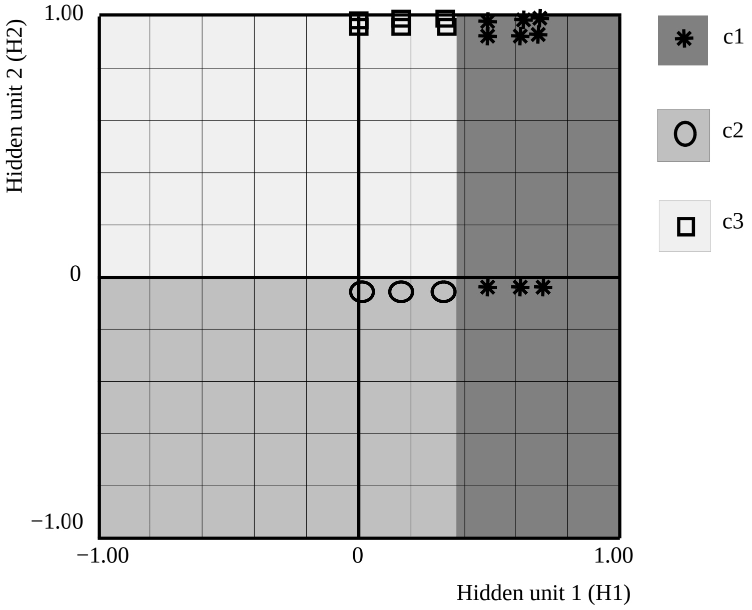

Figure 8: A scattered plot of the hidden unit activations of the network in Fig. 7.

```
else if -2.4 f - 2.4 g - 2.4 h + 7.17 < 0, then c2,

else c3.
```

The above set of rules can be simplifies as:

```
If a + b + c + d + e > 2.06, then c1,

else if f + g + h > 2.98, then c2,

else c3.
```

It is obvious that the above set of rules is equivalent to the original M-of-N rules.

5 Conclusions

This paper presents a pruning-based method (PBNN) for constructing neural networks from decision trees. The PBNN method has several advantages over Ivanova and Kubat's TBNN method. First, instead of calculating a fixed set of weights for a given problem as is the case for TBNN, the PBNN method determines appropriate weight values by training on input patterns. Second, due to the pruning process, PBNN generates networks that are smaller than those constructed by TBNN. Third, the pruning process also compresses the internal representations of the networks. As a result, equivalent decision trees extracted from the pruned networks are smaller and simpler than those induced by well-known algorithms such as ID3 and C4.5. Fourth, PBNN is more general than TBNN since it can be applied to generate networks that implement either the DNF or M-of-N rules.

References

[1] G. R. Dattatreya and L. N. Kanal, Adaptive improvement of pattern recognition trees, In *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, 393–397, 1983.

[2] I. Ivanova and M. Kubat, Initialization of neural networks by means of decision trees, *Knowledge-Based Systems*, 8(6):333–344, 1995.

[3] Y. Park, A mapping from linear tree classifiers to neural net classifiers, In *Proc. IEEE International Conference on Neural Networks*, 94–100, 1994.

[4] J. R. Quinlan, Induction of decision trees, *Machine Learning*, 1:81–106, 1986.

[5] J. R. Quinlan, Decision trees as probabilistic classifiers, In *Proc. 4th International Workshop on Machine Learning*, 31–37, 1987.

[6] J. Schuermann and W. Doster, A decision theoretic approach to hierarchical classifier design, *Pattern Recognition*, 17:359–269, 1984.

[7] I. K. Sethi, Entropy nets: From decision trees to neural networks, *Proc. of IEEE*, 78:1605–1613, 1990.

[8] I. K. Sethi, Neural implementation of tree classifiers, *IEEE Trans. on Systems, Man, and Cybernetics*, 25(8):1243–1249, 1995.

[9] R. Setiono, A penalty-function approach for pruning feedforward neural networks, *Neural Computation*, 9(1):185–204, 1997.

[10] R. Setiono and L. C. K. Hui, Use of quasi-Newton method in a feedforward neural network construction algorithm, *IEEE Trans. on Neural Networks*, 6(1):273–277, 1995.

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