Generating rules from trained network using fast pruning

Rudy Setiono and Wee Kheng Leow
School of Computing
National University of Singapore
Lower Kent Ridge, Singapore 119260
rudys@comp.nus.edu.sg, leowwk@comp.nus.edu.sg

Abstract

Before symbolic rules are extracted from a trained neural network, the network is usually pruned so as to obtain more concise rules. Typical pruning algorithms require retraining the network which incurs additional cost. This paper presents FERNN, a fast method for extracting rules from trained neural networks without network retraining. Given a fully connected trained feedforward network, FERNN first identifies the relevant hidden units by computing their information gains. Next, it identifies relevant connections from the input units to the relevant hidden units by checking the magnitudes of their weights. Finally, FERNN generates rules based on the relevant hidden units and weights. Our experimental results show that the size and accuracy of the tree generated are comparable to those extracted by another method which prunes and retrains the network.

1. Introduction

Neural networks have been successfully applied in a variety of problem domains. In many applications, it is highly desirable to extract symbolic classification rules from these networks. In order to obtain a concise set of symbolic rules, redundant and irrelevant units and connections of a trained neural network are usually pruned before rules are extracted [1, 5, 7]. This process can be time consuming as most algorithms for neural network pruning such as Optimal Brain Surgeon [2] and Neural Network Pruning with Penalty Function (N2P2F) [6] are iterative in nature. They retrain the network after removing some connections or units. The retrained network is then checked to see if any of its remaining units or connections meet the criteria for further removal. More often than not, the amount of computations incurred during retraining is much higher than that required by the original network.

This paper presents a method for extracting symbolic rules from trained feedforward neural networks with a single hidden layer. The method removes the redundant weights based on their magnitude and no network retraining is needed. We can thus speed up the process of rule extraction considerably and make neural networks an attractive tool for generating symbolic classification rules.

2. FERNN: Fast Extraction of Rules from NN

FERNN consists of the following steps:

1. Train a fully connected network such that an augmented cross-entropy error function is minimized. The usual cross entropy error function is augmented by a penalty function so that irrelevant network connections have very small weights and can be removed without affecting the network's classification accuracy.

2. Use information gains to identify the relevant hidden units and build a decision tree that classifies the patterns in terms of the network activation values.

3. For each hidden unit whose activation values are used for node splitting in the decision tree, remove the irrelevant input connections to this hidden unit. We have developed a simple criterion for removing the network connections from the input units to the hidden units without affecting the network's classification accuracy.

4. For data sets with discrete attributes, replace each node splitting condition by an equivalent set of symbolic rules. A node splitting condition can be rewritten as either DNF (disjunctive normal form) rules or MofN rules. DNF rules are expressed as a disjunction of conjunctions, while MofN rules are expressed in the following form:
   ```
   if {at least/exactly/at most} M of
   the N conditions (C1,C2,...,CN) are
   satisfied, then ...
   ```

The neural network training method is described in detail in [6]. In the following subsections, we will focus on the rule generating algorithm.

2.1. An illustrative example

A network with 10 hidden units is trained to solve the Monk2 problem [8], which is an artificial problem of identifying the monks. The training data set consists of 169 patterns, each described by 6 discrete attributes as shown in Table 1. The attribute values are coded in binary 0 or 1 format. Hence, the network requires 17 input units plus an additional input with a constant value of 1 for the hidden unit bias. A pattern is classified as a monk if exactly two of its six attributes have their first values, i.e., if exactly two of $\{x_3, x_6, x_8, x_{11}, x_{15}, x_{17}\}$ equal 1. We trained a network to correctly classify all the training samples and illustrate how the classification rule can be recovered from this network.

To appear in IJCNN'99.

Table 1: The attributes of the Monk2 problem.

attribute	possible values	input units
head_shape	round,square,octagon	x_3, x_2, x_1
body_shape	round,square,octagon	x_6, x_5, x_4
is_smiling	yes, no	x_8, x_7
holding	sword,balloon,flag	x_{11}, x_{10}, x_9
jacket_color	red,yellow,green,blue	$x_{15}, x_{14}, x_{13}, x_{12}$
has_tie	yes,no	x_{17}, x_{16}

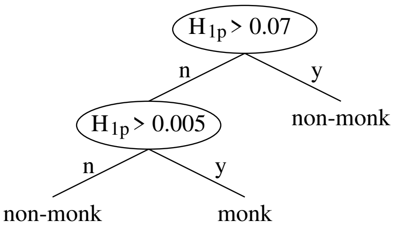

Figure 1: Decision tree generated by C4.5 from the network trained to solve the Monk2 problem.

2.2 Identifying the relevant hidden units

Once a neural network has been trained, its relevant hidden units are identified using information gain method. For this purpose, the C4.5 algorithm is employed to build a decision tree using the hidden unit activations of training patterns that have been correctly classified by the neural network along with the patterns' class labels. These activation values are continuous in the interval $[0, 1]$ since each activation value has been computed as the sigmoid of the weighted inputs. Let D be the set of hidden unit activations of correctly classified patterns and $\overline{P}$ be the number of correctly classified patterns. For hidden unit j, its activation values H_{jp} in response to patterns $p, p = 1, 2, \ldots, \overline{P}$, are first sorted in increasing order. The set D is then split into two groups $D_1 \equiv \{H_{j1}, \ldots, H_{jq}\}$ and $D_2 \equiv \{H_{j,q+1}, \ldots, H_{j\overline{P}}\}$ where $H_{jq} < H_{j,q+1}$, and the information gained by this split is computed. This information gain is computed for all possible splitting of the activation values, and the maximum gain is taken as the information gain of hidden unit j.

Suppose that each pattern in the data set D belongs to one of the C classes, and n_c is the number of patterns in class $\mathcal{C}_c$, the expected information for classification is

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

where the number of patterns in the set D is $N = \sum_{c=1}^{C} n_c$. For the two subsets D_1 and D_2 of D, the expected information is similarly computed:

$$I(D_j) = -\sum_{c=1}^{C} \frac{n_{cj}}{N_j} \log_2 \frac{n_{cj}}{N_j} \tag{2}$$

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

$$\text{Gain}(H_{jq}) = I(D) - [I(D_1) + I(D_2)] \tag{3}$$

and the normalized gain is

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

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

For the Monk2 network, applying C4.5 on the hidden unit activation values gives the decision tree shown in Fig. 1. This tree indicates that the monks and non-monks in the data set can be distinguished by the activations of hidden unit 1 alone. The unit's activation values are split into two groups at the threshold value of 0.07. The values that are smaller than or equal to 0.07 are further split into two subgroups at the threshold value of 0.005.

2.3. Identifying the relevant inputs

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

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

$$\sum_{k \in S} |w_{jk}| < 2\,(U - L) \tag{4}$$

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

$$H_{jp} \leq (L + U)/2\,, \tag{5}$$

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

A similar proposition is obtained when $H_{jp} > H_{jt}$.

The decision tree in Figure 1 contains 2 tests involving H_1. The values of L are 0.07 and 0.005 and the corresponding values of U are 0.79 and 0.04, respectively.

- Test 1. The new threshold value is $(0.07 + 0.79)/2 = 0.43$. The weights of the connections from the input units to the hidden unit are sorted in increasing order of their absolute values. The first 11 weights sum up to 0.036 which is smaller than the threshold value of $2(0.79 - 0.07) = 1.44$. Therefore, they are removed.

- Test 2. The new threshold value is $(0.005 + 0.04)/2 = 0.023$. The same 11 weights as in Test 1 total less than $2(0.04 - 0.005) = 0.07$ and are removed.

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

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

2.4. Rule generation

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

MofN rule can be more concise than DNF rules and it is more general. The conjunction $x_1 \wedge x_2 \ldots \wedge x_K$ is equivalent to *exactly K of* $\{x_1, x_2, \ldots, x_K\}$, while the disjunction $x_1 \vee x_2 \ldots \vee x_K$ is equivalent to *at least 1 of* $\{x_1, x_2, \ldots, x_K\}$. FERNN always attempts to extract an MofN rule first before resorting to generating a DNF rule.

Proposition 2 *Let the node splitting condition be*

$$w_1x_1 + w_2x_2 + \ldots w_Kx_K \leq U \tag{6}$$

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

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

We can then replace condition (6) by the equivalent rule

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

(Proof omitted)

A similar proposition has been proved for the case $w_1x_1 + w_2x_2 + \ldots w_Kx_K > L$.

The relevant weights of a trained network can have any real values. Before we test if they satisfy the conditions in Proposition 2, the following steps can be performed to improve the chances of satisfying them:

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

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

- Divide all the weights by the smallest w_i.

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

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

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

$$1.9 < 1.1x_3 + x_6 + 1.1x_8 + x_{11} + 1.1x_{15} + 1.1x_{17} \leq 2.7 \tag{7}$$

The above condition can be transformed into the MofN rule:

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

by applying Proposition 2.

When it is not possible to express a node splitting condition as an MofN rule, FERNN generates DNF rule. A general purpose algorithm X2R [3] is used to automate the DNF rule generation process. X2R takes as input a set of discrete patterns and their corresponding target outputs and produces a set of classification rules with 100% accuracy if there are no identical patterns with different class labels.

3. Experimental results

The effectiveness of FERNN has been tested on 15 problems listed in Table 2. The data sets were obtained from the UCI repository [4] or generated according to the function definitions given in [9]. Each data set was randomly divided into three subsets: the training set (40%), the cross validation set (10%) and the test set (50%). For all experiments, the initial number of hidden units in the network was 10.

Table 2: Comparison of various algorithms.
a = predictive accuracies, and N = tree size.

	C4.5		N2P2F+C4.5		FERNN	
Dataset	a	N	a	N	a	N
Monk1	100.0	15	99.9	10.2	99.9	10.5
Monk2	75.6	45	100.0	5.0	99.8	5.1
Monk3	100.0	9	99.9	3.2	99.9	3.0
CNF12a	100.0	41	100.0	7.4	100.0	7.7
CNF12b	97.0	87	99.9	30.7	99.9	26.6
DNF12a	100.0	33	100.0	11.4	99.9	10.2
DNF12b	100.0	43	99.6	19.6	99.9	12.2
MAJ12a	91.9	63	99.9	3.0	99.9	3.0
MAJ12b	81.5	195	100.0	3.0	100.0	3.0
MUX12	100.0	147	99.9	25.5	99.7	28.2
Australian	83.2	36	83.5	7.1	83.9	10.3
BCancer	93.7	13	94.1	5.9	95.1	6.8
HeartD	73.7	32	82.3	10.6	82.7	11.3
Pima	71.1	75	72.6	17.8	72.8	17.3
Sonar	79.8	13	81.8	8.8	83.5	9.5

Table 2 compares the tree size (i.e., the number of nodes) and the predictive accuracy of the decision trees generated by FERNN with those generated by C4.5 and the combined algorithm N2P2F+C4.5. For the combined algorithm, N2P2F [6] stopped pruning when the accuracy of the network on the cross validation set dropped by more than 1% from its best value. C4.5 was then used to generate decision trees using the activation values of the pruned networks. FERNN and C4.5 do not require the cross validation set, hence, 50% of the data was used for each training session. The tree size and predictive accuracy for FERNN and N2P2F+C4.5 are averaged over 20 test runs.

The decision trees extracted by N2P2F+C4.5 and FERNN are smaller than those generated by C4.5. This is expected since the decision nodes of the trees generated by N2P2F+C4.5 and FERNN usually involve several original attributes of the data. In contrast, the decision nodes of the trees generated by C4.5 involve only individual attributes. Using multiple attributes in the decision nodes may result in simpler rules, such as the MofN rules for the Monk2 and MAJ12 problems. Multi-attribute decision nodes may also improve the accuracy of the tree because samples from real world problems may be better separated by oblique hyperplanes. This is the case with the heart disease data set (HeartD in Table 2) where significant improvement is achieved by the neural network methods over C4.5.

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

4. Conclusion

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

References

[1] R. Blassig, "GDS: Gradient descent generation of symbolic rules," in *Adv. in Neural Info. Proc. Sys. 6*, 1994, 1093–1100, San Mateo, CA: Morgan Kaufmann.

[2] B. Hassibi and D.G. Stork, "Second order derivatives for network pruning: Optimal Brain Surgeon," in *Adv. in Neural Info. Proc. Syst. 5*, 1993, 164–171, San Mateo, CA: Morgan Kaufmann.

[3] H. Liu and S.T. Tan, "X2R: A fast rule generator," in *Proc. of IEEE Int. Conf. on SMC*, New York: IEEE Press, 1995, 1631–1635.

[4] C. Merz, and P. Murphy, UCI repository of machine learning databases `http://www.ics.uci.edu/mlearn/MLRepository.html`, Irvine, CA: University of California, Dept. of Info. and Comp. Sci. 1996.

[5] R. Setiono. "Extracting rules from neural networks by pruning and hidden-unit splitting," *Neural Computation*, 9(1) 205–225, 1997.

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

[7] R. Setiono and H. Liu. "Symbolic representation of neural networks," *IEEE Computer*, 29(3), 71–77, 1996.

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

[9] R. Vilalta, G. Blix, and L. Rendell. "Global data analysis and the fragmentation problem in decision tree induction," in van Someren, M., and Widmer, G., eds., *ECML-97*, 312–326, 1997, Springer-Verlag.