A search technique for rule extraction from trained neural networks

R. Krishnan a,*, G. Sivakumar b, P. Bhattacharya b

a Centre for Artificial Intelligence and Robotics, Raj Bhavan Circle, High Grounds, Bangalore 560 001, India
b Department of Computer Science and Engineering, Indian Institute of Technology, Powai, Mumbai 400 076, India

Abstract

Search methods for rule extraction from neural networks work by finding those combinations of inputs that make the neuron active. By sorting the input weights to a neuron and ordering the weights suitably, it is possible to prune the search space. Based on this observation, we present an algorithm for rule extraction from feedforward neural networks with boolean inputs and analyze its properties.

Keywords: Neural networks; Rule extraction; Knowledge discovery

1. Introduction

Artificial Neural Networks (ANNs) have empirically been shown to perform well on several machine learning problems. Multilayer feedforward networks trained using the backpropagation algorithm (Rumelhart et al., 1986) are considered to be the workhorse of neural networks. Reasonably satisfactory answers to questions like how many examples are needed for a feedforward neural network to learn a concept and what is the best neural network architecture for a particular problem domain (given a fixed number of training examples) are available, so it is now possible to train neural networks without guesswork. This makes neural networks an excellent tool for Data Mining (Fayyad and Uthurusamy, 1996), where the focus is to learn data relationships from huge databases. However, there are applications like credit approval and medical diagnosis where explaining the reasoning of the neural network is important. The major criticism against neural networks in such domains is that the decision making process of neural networks is difficult to understand (Andrews et al., 1995). This is because the knowledge in the neural network is stored as real-valued parameters (weights and biases) of the network, the knowledge is encoded in distributed fashion and the mapping learnt by the network could be non-linear as well as non-monotonic. One may wonder why neural networks should be used when comprehensibility is an important issue. The reason is that predictive accuracy is also very important and neural networks have an appropriate inductive bias for many machine learning domains. The predictive accuracies obtained with neural networks are often significantly higher than those obtained with other learning paradigms.
Recent research on understanding the working of a trained feedforward neural network has focused on extraction of symbolic rules (Andrews et al., 1995; Fu, 1994; Towell and Shavlik, 1993). The rule extraction task may be viewed as a search task (Fu, 1994) or as a learning task (Craven and Shavlik, 1994). In the search approach, rules are extracted at the level of individual (hidden and output) neurons in the neural network by looking at their weights and biases. One of the problems with the search approach is how to constrain the search space of possible rule combinations. In this paper we present a new method to constrain this search space and analyze some properties of this method. In Section 2, we explain the problem of rule extraction. In Section 3 we discuss the COMBO algorithm for rule extraction and analyze its properties in Section 4. Section 5 gives a brief overview of related work and concludes the paper.

2. The rule extraction problem

In a trained neural network, the knowledge acquired in the training phase is encoded in the network architecture, the activation function used and the weights and biases of the neurons. The task of rule extraction is to use one or more of the above pieces of information and extract a set of rules from the neurons. We consider the case where the inputs to the feedforward network are boolean and the outputs are also boolean (i.e. a classification problem).

Neurons in feedforward neural networks have activations defined by

\[ A_i = \text{Act} \left( \sum_j w_{ij} \times o_j + \theta_i \right), \tag{1} \]

\[ \text{Act}(x) = \frac{1}{1 + e^{-\alpha x}} \tag{2} \]

where \( A_i \) is the activation of neuron \( i \), \( w_{ij} \) is the weight on a link from neuron \( j \) to neuron \( i \), \( o_j \) is the activation of neuron \( j \), \( \theta_i \) is the bias on the neuron \( i \), \( \text{Act}(\ ) \) is a nonlinear activation function (the sigmoidal function), \( \alpha \) is a parameter controlling the slope of the sigmoidal function.

Search methods for extracting rules try to find combinations of the input values to a neuron that result in it having an activation near 1 (for a confirming rule) or an activation near 0 (for a disconfirming rule). For a neuron to have an activation near 1, we want to find that combination of weights such that the quantity in Eq. (1), \( A_i \approx 1 \). Similarly for a neuron activation near 0, we should have \( A_i \approx 0 \). The value of \( A_i \) at which \( \text{Act}(A_i) \approx 1 \) is called the threshold of the neuron.

The process of finding rules for the hidden layer neurons is simplified by the inputs being boolean. In this case, for evaluating \( A_i \) in Eq. (1), we need to consider only the weights feeding into the neuron. The process of finding rules for the output layer neurons is more complex because the hidden layer neuron could have any activation in the interval \([0, 1]\). However, the hidden layer neuron can be made to approximate a boolean neuron by controlling the steepness of the activation function. By increasing the value of the \( \alpha \) parameter in Eq. (2), the hidden layer neuron can be made to approximate a boolean activation. Setting the value of the \( \alpha \) parameter to a high value (\( \approx 10 \)) ensures that all the hidden layer neurons in the network will have an activation of near 0 or near 1. This enables us to then treat the outputs of the hidden layer neurons as boolean quantities and focus only on its weights to the output layer neurons in the rule extraction process.

The extracted rules should be valid (the rules must hold regardless of the values of the unmentioned variables in the rules), must be maximally general (if any of the antecedents are removed, the rule should no longer be valid) and complete (all possible valid and maximally general rules must be extracted).

3. The proposed method

In this section, we explain the concept of a combination tree and how it is used in the rule generation. In order to treat the positive and negative weights of the neuron uniformly, we have adopted an admissible transformation of weights first used by Sethi and Yoo (1996a), to convert all the negative weights of the neuron to positive
quantities. This transformation allows us to work with positive weights only.

We consider a three-layer feedforward network trained using the backpropagation rule (Rumelhart et al., 1986). A confirming rule is a rule explaining when a neuron is turned on; a disconfirming rule explains when a neuron is turned off. We explain in detail the procedure for extracting a confirming rule; a similar procedure applies for disconfirming rules. As mentioned in Section 1, one of the most crucial issues in developing a rule extraction algorithm is how to constrain the size of the solution space searched. Suppose a neuron in the network has four positive weights labeled 1, 2, 3, and 4, respectively. With these four neurons, we can form \( 2^4 - 1 \) combinations. Ignoring the null combination (i.e., \( 4^0 \)), the other combinations can be considered to be nodes of a tree with a combination of size \( i \) at the \( i \)th level of the tree. Fig. 1 shows the combination tree for a neuron having four positive weights.

Instead of randomly trying combinations of the weights, we first sort the weights and then generate combinations of all possible sizes. The combinations for any particular size are ordered in descending order of the sum of the weights in the combination. Because of the ordering of the weights and because of the maximal generality restriction on the rules, it is possible to exclude some combinations from the search. We now explain how sorting the weights of the neuron and considering the combinations in the above order helps in pruning the search space.

3.1. Pruning the search in a combination tree

There are two types of pruning that can occur in a combination tree.

- **Prunings at the same level of the tree.** If a combination at any level fails, then all the other combinations at this level can be pruned away. This is because all combinations at the same level have the same length, and because of the ordering of the weights, if a combination at a level fails, then all other combinations at that level will also fail as their weighted sum will sum to less than the combination that failed. Hence they need not be considered in the search for rules.

- **Prunings at deeper levels of the tree.** If a combination at a level succeeds in forming a rule then all the combinations of higher sizes of which the present combination forms a subset need not be considered. Although these combinations will succeed in forming rules, these rules will be subsumed by the rules formed from the current combination and can be excluded because of the maximal generality condition.

Consider the combination tree shown in Fig. 1. The level 2 combinations are 12, 13, 14, 23, 24, 34. If 12 fails to generate the positive antecedent of a rule, we need not try the combinations 13, 14, 23, 24, 34 and they can be pruned away. On the other hand if 12 succeeds in forming a positive antecedent of the rule then all the combinations in its subtree i.e. 123, 124 and 1234 can be pruned away as they will form more specific rules than the

![Fig. 1. A combination tree.](image)
combination 12 and can be removed because of the maximum generality restriction.

3.2. The COMBO algorithm for rule generation

The COMBO rule extraction algorithm is based on the pruning strategy discussed above and works in three stages. In the first stage we extract rules for the output layer neurons. We have considered the hidden layer neurons to represent boolean concepts by making its activation function have a high gain as explained in Section 2. In the second stage we extract rules for the hidden layer neurons. In the final stage the rules obtained in the first stage are rewritten in terms of rules obtained in the second stage to get rules explaining the input–output relationship. The algorithm for extracting a confirming rule for an individual neuron is as below.

1. In the first step, all the negative weights of the neural network are converted to positive quantities. This is done as follows.

   For binary inputs: Convert all the negative weights to positive weights by using the following admissible transformation (Sethi and Yoo, 1996a). Replace each input literal \( x \), which has a negative weight with its negated literal, \( \neg x \). Replace its negative weight, \( w \) with \( \neg w \) and set the new threshold \( T_{\text{new}} = T_{\text{old}} - w \).

   For bipolar inputs: Networks with bipolar inputs use \(-1\) to represent the ‘off’ state of the neurons. We first convert the network with bipolar inputs to a network with binary inputs using the following transformation. Each weight in the bipolar network is transformed to a weight of the corresponding binary network using the equation \( w_{ij}^{\text{binary}} = 2w_{ij}^{\text{bipolar}} \). The threshold of the binary network is set as \( T_{\text{binary}} = T_{\text{bipolar}} + \sum w_{ij}^{\text{bipolar}} \). Then the admissible transformation is applied to this network with binary inputs to convert all the negative weights of the neuron to positive quantities.

2. Sort the weights of the neuron for which a rule is required in descending order.

3. Generate combinations of the sorted weights in the ascending order of their sizes. First, all combinations of size one, then combinations of size two, and so on, are generated. Within a combination of size \( k \), the combinations are ordered as follows. Given two combinations \( C_1 \) and \( C_2 \), \( C_1 \) is before \( C_2 \) if \( \sum_{w_i \in C_1} w_i > \sum_{w_i \in C_2} w_i \).

4. Start with combinations of size 1. For the next combination in the untried list, check if

\[
\sum W_C + \text{bias on neuron} > \text{threshold of neuron},
\]

where \( W_C \) are the weights in the combination. Note that \( \text{Act}(\text{threshold}) \approx 1 \), so if the neuron has activation above this value, the concept corresponding to it is true. The actual value of this threshold depends on the value of \( z \) in Eq. (2).

   (i) If the above inequality is not satisfied, remove all further combinations of the same size from the untried list. Here the size of the combination is the number of weights it represents. For instance, the combination 12 has size 2.

   (ii) If the above inequality is satisfied for the current combination, insert the current combination in the success list. Remove all combinations of greater sizes of which the current combination is a subset. For example, in Fig. 1, if the combination 12 satisfies the above inequality, remove the combinations 123 and 1234 from the untried list.

Repeat this step until there are no combinations in the untried list.

5. With each combination in the success list, form the corresponding rule.

The conforming rules for the entire network are generated as follows.

1. Do steps 1–5 above for all the neurons in the output and hidden layers. The output layer neuron’s rules will have the hidden layer neurons as the antecedents while the hidden layer neurons will have the input layer neurons as antecedents.

2. Rewrite the above set of rules such that the antecedents are the input layer neurons and the consequents are the output layer neurons.

Disconfirming rules are generated by a similar method.

To illustrate our method, we present two examples. The first is based on an example presented in (Sethi and Yoo, 1996a). In the second example,
we discuss rule extraction for the encoder problem (Rumelhart et al., 1986).

Example 1. Consider a neuron whose weight vector is \( w = [2, 2, -1, -2] \) and threshold \( T = 0.5 \). After applying the admissible transformation, the weight vector becomes \( w = [2, 2, 1, 2] \) and the new threshold becomes \( T = 3.5 \). After sorting, the weight vector becomes \( w = [2, 2, 2, 1] \). The indices of the sorted weights in the original weights is given by the vector \([1, 2, 4, 3]\) and they represent the literals \( x_1, x_2, x_4, x_3 \). This results in the combination tree shown in Fig. 2. The sum of the weights in each contribution is shown in brackets. For a rule to be formed, the sum of weights in the combination has to be greater than or equal to the modified threshold viz. 3.5. At the first level of the tree, the first combination has a weighted sum of 4 which exceeds the threshold of 3.5 and hence it can form a rule. The boolean expression for the combinations that succeeded in forming rules is given by

\[
f = x_1x_2 \lor x_1x_3 \lor x_2x_4.
\]

Example 2. In this example, we extract rules using COMBO for the encoder problem. In this problem, a set of orthogonal input patterns is mapped to a set of orthogonal output patterns using a small set of hidden units. The neural network has eight inputs and eight outputs. The hidden layer has three neurons. The task is to make the neural network learn an identity mapping. For instance if the input vector is \([10000000]\) the output is also \([10000000]\). The network basically learns to encode the eight patterns in three bits using the hidden layer neurons. The weights for the hidden and output layers of the trained network are shown in Tables 1 and 2, respectively. The following rules were extracted for the output layer neurons.

\[
\begin{align*}
\alpha_1 &= h_1h_2h_3, \\
\alpha_2 &= h_1h_2, \\
\alpha_3 &= h_1h_2h_3, \\
\alpha_4 &= h_2h_3, \\
\alpha_5 &= h_1h_2h_3, \\
\alpha_6 &= h_1h_2h_3, \\
\alpha_7 &= h_1h_2h_3, \\
\alpha_8 &= h_1h_2h_3.
\end{align*}
\]

For the first hidden layer neuron, after applying the admissible transformation, the threshold becomes \( T_{new} = 0.56 + 2.94 + 3.46 + 1.05 + 7.14 + 0.5 = 15.65 \). The following rules were extracted by COMBO for this neuron. \( h_1 = i_3i_6i_8, \ h_1 = i_3i_6i_8, \ h_1 = i_3i_6i_8, \ h_1 = i_1i_6i_8, \ h_1 = i_3i_6i_8, \ h_1 = i_3i_6i_8, \ h_1 = i_3i_6i_8, \ h_1 = i_3i_6i_8 \).

Hence we find that for the first hidden node to be on, the literals should occur in the following

![Fig. 2. Combination tree for neuron with weight vector \( w = [2, 2, -1, -2] \) and \( T = 0.5 \).](image)
form. $i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8$. Similarly, for the second hidden node to be off, the literals should occur as $i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8$. Also for the third hidden node to be off, the literals should occur as $i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8$. Thus, $i_1$ is the only non-negated literal occurring in all the above rules. Hence for the rule for the first output neuron, $o_1 = h_1 h_2 h_3$, the input $i_1$ should be on. This confirms that the network has learnt the mapping for the first input and output neurons.

4. Analysis of the COMBO algorithm

The rules generated by COMBO are valid, maximally general and complete.

**Proposition 1.** The rules generated by COMBO are valid.

**Proof.** A rule is valid if it holds regardless of the activations of the neurons not specified in the rule. Suppose $W_C$ are the weights in the combination. A rule is formed only if the following condition is satisfied:

$$\sum_{W_C \in \mathcal{W}} W_C > T_{\text{new}}.$$  

Here $T_{\text{new}}$ is the modified value of the threshold after the admissible transformation. It is easy to see that in the presence of the weights not in the above combination, the above relation will still hold. This is because all the weights not in the combination are also positive the presence of such weights will only increase the weighted sum. Hence all the rules generated by COMBO are valid. □

**Proposition 2.** The rules generated by COMBO are maximally general.

**Proof.** In the search for confirming rules in the combination tree, when a combination succeeds all the combinations of the subtree of which it is the root are pruned away. Hence, all the combinations that could form subsumed rules are pruned away and so the rules generated are maximally general. □

**Proposition 3.** The rules generated by COMBO are complete.

**Proof.** COMBO prunes away combinations in two ways: breadth-wise and depth-wise along a tree. In the former case, the pruned combinations cannot form a rule. In the latter case, the pruned combinations can form rules, but these are subsumed by other rules. All other possible rules will be generated. Hence the extracted rules form a complete set. □

4.1. Complexity of the method

For the case of success (i.e. some rules exist for the neuron), the success could occur at any level in the combination tree. Suppose there are $N$ weights for a given neuron. The maximum number of combinations that could succeed in the combination tree for the $N$ weights is $N \binom{N}{N/2}$. This happens when all the weights are equal and the quantity (bias – threshold) is half the sum of all the weights to the neuron. To get to the level in the tree where combinations of size $N/2$ are tried, one combina-
tion in each of the preceding levels should have failed. Hence the maximum number of combinations that need to be tried is \( N \choose 2 + \frac{1}{2} (N - 1) \). For large values of \( N \), the term \( \frac{1}{2} (N - 1) \) can be neglected, hence the maximum number of combinations that could succeed is \( N \choose 2 \). Hence the maximum number of rules that could be extracted is \( N \choose 2 \). Using Stirling’s approximation

\[
m! \approx \sqrt{2\pi m} \left( \frac{m}{e} \right)^m
\]

we get

\[
N \choose 2 \approx 2^N \sqrt{\frac{2}{\pi N}}
\]

Thus, the worst case complexity of the method is \( O(2^N) \).

4.2. Bounds on the minimum number of prunings

We now analyze the minimum number of prunings done by the COMBO algorithm. As noted earlier, prunings can occur in two ways viz. breadth-wise and depth-wise. If at a node, a combination succeeds in forming a rule, depth-wise pruning takes place, otherwise breadth-wise pruning takes place. Pruning will take place at all the internal nodes of the tree. At the leaf nodes of the tree, pruning may or may not take place. For example, the tree nodes in Fig. 1 that are not capable of resulting in pruning are not only 4, 34, 234, 1234 but also 14, 24, 124, 134 (if they are successful in forming a rule). In general, all those combinations containing the digit ‘\( N \)’ (where \( N \) is the total number of weights) could result in no prunings.

Now we get an estimate of the tree leaves that could result in no pruning. At level \( i \), ‘\( N \)’ can occur in combinations \( N-1 \choose i-1 \) times. Therefore the total number of times \( N \) can occur in all combinations is

\[
\sum_{i=1}^{N} N-1 \choose i-1
\]

These are the cases in which pruning may not occur in the worst case.

Hence in

\[
\sum_{i=1}^{N} N \choose i - \sum_{i=1}^{N} N-1 \choose i-1 = 2^N - 2^{N-1}
\]

combinations, pruning will take place. So the minimum number of tree nodes pruned will be

\[
\frac{2^{N} - 2^{N-1}}{2} = 2^{N-2}.
\]

However, this estimate is pessimistic, as there are neurons which result in definitely more than one pruning. For example, from Fig. 1, it can be observed that whether the combination ‘1’ succeeds or fails, three other combinations will be pruned away.

5. Related work and conclusions

A number of researchers have proposed algorithms for extracting rules from a trained feed-forward neural network. The main focus of such algorithms is the mechanism by which the search space is reduced. Fu’s method (Fu, 1994) relied on three heuristics for reducing the search space. The method of Towell and Shavlik (1993) relies on clustering similar weights and replacing links with similar weights by a single set of links. The method of Saito and Nakono (1988) limits the number of antecedents of each rule. Alexander and Mozer (1994) have attempted to reduce the complexity of the search using template matching, where rules are extracted by matching the weight vector of a neuron against a set of canonical weight templates. Sethi and Yoo (1996a) have proposed a backtracking search method for rule extraction. Of all the above, only Fu’s method generates valid, maximally general and complete rules.

In this paper we have proposed a new search algorithm for rule extraction that relies on sorting the weights and considering combinations of the weights in the order of their weighted sum. Our algorithm generates valid, maximally general and complete rules. Although the algorithm has an exponential complexity, it is still viable for use in small or medium sized neural networks. It is still an open question whether there is a polynomial time algorithm for rule extraction by search.

The pruning properties of the algorithm have been investigated and a lower bound for the number of possible prunings has been calculated. Although a stricter bound could possibly be
obtained, our conjecture is that this method results in the maximum number of prunings.

Future work will involve extending our method for multivalued and continuous valued inputs. Related work on these lines is discussed in (Sethi and Yoo, 1996b; Taha and Ghosh, 1997).

References