Extracting Propositional Rules from Feed-forward Neural Networks — A New Decompositional Approach

Sebastian Bader and **Steffen Hölldobler** and **Valentin Mayer-Eichberger**
International Center for Computational Logic
Technische Universität Dresden, Germany

Abstract

In this paper, we present a new decompositional approach for the extraction of propositional rules from feed-forward neural networks of binary threshold units. After decomposing the network into single units, we show how to extract rules describing a unit's behavior. This is done using a suitable search tree which allows the pruning of the search space. Furthermore, we present some experimental results, showing a good average runtime behavior of the approach.

1 INTRODUCTION AND MOTIVATION

As the knowledge stored in a neural network is hidden in its weight, humans have no direct access to it. The goal of rule extraction is to obtain a human readable description of the output units behavior with respect to the input units. This is usually done using if-then rules describing under which conditions the output units will be active. Throughout this paper, we will use networks of bipolar binary threshold units and show how to extract propositional if-then rules.

Rule extraction attempts can be divided into pedagogical and decompositional approaches. The first conceives the network as a black box, while the latter decomposes the network, constructs intermediate rules and recomposes those. For a general introduction into the field we refer to [Tickle *et al.*, 1998].

Example 1. *Figure 1 and 2 show a simple network and the results of a naive pedagogical extraction. All possible inputs are presented to the network and the network is evaluated. For each active output unit, a rule is constructed such that its precondition matches the current input.*

Obviously, the naive pedagogical approach presented in Example 1 has an exponential complexity, as there are exponentially many different inputs, even for a single unit. In this paper, we will present new algorithms which allow for an efficient extraction. Even though, the problem itself is worst case exponential, our algorithms show a good average-case behavior. The approach presented here is closely related to the work by Krishnan et al [Krishnan *et al.*, 1999]. But we use a modified search tree, that can be constructed as need arises. Furthermore, we use a different set of pruning rules.

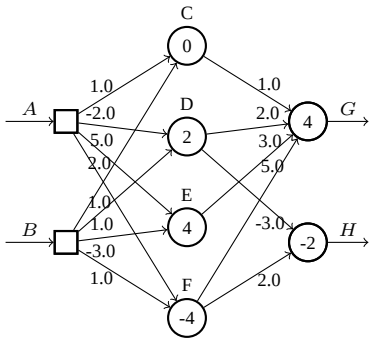


Figure 1: A simple 3 layer feed-forward connectionist system. Depicted are the thresholds of the nodes (inside the circles) and the weights of the connections. The rectangular nodes (A and B) serve as input units.

This paper is organized as follows: After reviewing some preliminary notions in Section 2, we present our approach in Section 3. This is followed by a presentation of some experimental results in Section 4. Finally we draw some conclusions and discuss further work in Section 5.

2 PRELIMINARIES

In this section we will briefly introduce some required notations from artificial neural networks and propositional rules and programs. For a more detailed introduction we refer to [Bishop, 1995; Rojas, 1996] and [Lloyd, 1988] respectively.

2.1 Artificial Neural Networks

A connectionist system (also called artificial neural network) is a directed graph of simple computational units (see Figure 1). Every unit has an activation value which is propagated along its weighted output connections to other units. Some distinguished units serve as input units, whose activation will be set from outside. All other units compute their activation based on the activation of their predecessor units and the weights of the connection. In our case, we will deal with so called bipolar binary threshold units only, i.e. units whose activation is either $+1$ or -1. A unit will be active ($+1$) if its current input exceeds its threshold, and inactive otherwise. Some of the units serve as output units. In Figure 1, those are marked with little outgoing arrows (G and

A	B	G	H
-1	-1	-1	+1
-1	+1	+1	+1
+1	-1	+1	+1
+1	+1	-1	+1

$$P_1 = \{ \quad H \leftarrow \bar{A} \wedge \bar{B}.$$
$$G \leftarrow \bar{A} \wedge B. \quad H \leftarrow \bar{A} \wedge B.$$
$$G \leftarrow A \wedge \bar{B}. \quad H \leftarrow A \wedge \bar{B}.$$
$$H \leftarrow A \wedge B.\}$$

Figure 2: A naive pedagogical extraction of the network shown in Figure 1. The table shows all possible inputs together with the corresponding outputs. The logic program corresponds exactly to the 1's of the output patterns.

H). Throughout this paper, we will use w_{AB} to refer to the weight of the connection from unit A to unit B and assume the weight to be 0, if there is no such connection.

2.2 Propositional Rules and Logic Programs

In this paper we will show how to extract propositional if-then rules from a neural network. These rules consist of a precondition and a consequence. We will consider rules with atomic consequences only, i.e. the consequence of a rule is a propositional variable. Furthermore, we will restrict the preconditions to be conjunction of (possibly negated) propositional variables only. We will treat rules with empty preconditions as facts, i.e. the consequence is assumed to be true without any condition. As propositional logic programs are just sets of those rules, we will use some notations customary in the logic programming community, as exemplified in Example 2.

Example 2. *Here is a simple propositional logic program, i.e. a set of propositional if-then rules, which will serve as a running example. The intended meaning of the rules is given on the right.*

$$P_2 = \{G \leftarrow \bar{A} \wedge B. \quad \%\ G \text{ is true, if } A \text{ is false and } B \text{ is true}$$
$$G \leftarrow A \wedge \bar{B}. \quad \%\ G \text{ is true, if } A \text{ is true and } B \text{ is false}$$
$$H.\} \quad \%\ H \text{ is always true}$$

3 THE COOP APPROACH

In this section, we will describe a new decompositional approach for the extraction of propositional rules from feed-forward neural networks of bipolar binary threshold units. First, we will show how to decompose a feed-forward network and introduce the required notations. In Section 3.2, we will show how to extract rules from a single perceptron. Afterwards, those intermediate results are composed.

3.1 Decomposition

As mentioned above, we will decompose the network into simple perceptrons, i.e. a single unit together with its incoming connections. To simplify the notions below, we introduce the *positive* and *negative form* of a perceptron.

Definition 3 (Perceptron). *A network consisting of a set of input units connected to a single output unit A is called a* perceptron *(denoted $\mathcal{P}_A$).*

Definition 4 (Positive and negative form). *The* positive form $\mathcal{P}_A^+$ *of a given perceptron $\mathcal{P}_A$ is obtained by multiplying all negative weights by -1 and negating the corresponding input symbols. The* negative form $\mathcal{P}_A^-$ *is obtained by multiplying all positive weights by -1 and negating the corresponding inputs.*

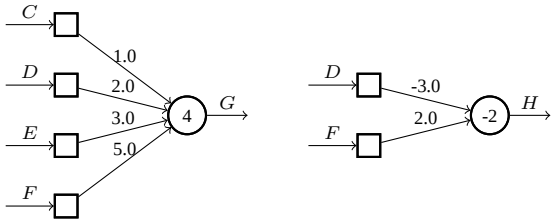


Figure 3: Two simple perceptrons $\mathcal{P}_G$ and $\mathcal{P}_H$

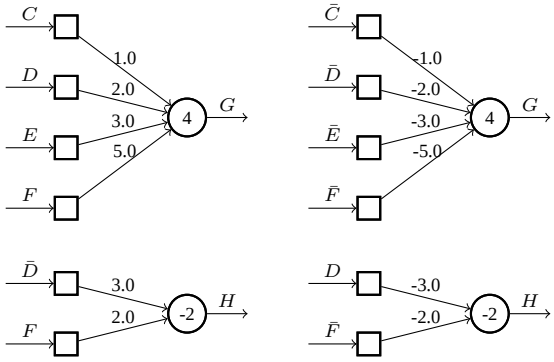


Figure 4: From left to right: $\mathcal{P}_G^+$, $\mathcal{P}_G^-$, $\mathcal{P}_H^+$ and $\mathcal{P}_H^-$

Example 5. *Figure 3 depicts two simple perceptrons, which can be seen as parts of the network shown in Figure 1. The positive and negative forms are depicted in Figure 4. For $\mathcal{P}_G$ we have $\mathcal{P}_G^+ = \mathcal{P}_G$, as there are no negative weights.*

In the sequel, we will often need to clamp some of the input units U of a given perceptron to be active. This will be done by *input patterns*. The intended meaning is, that all units occurring in an input pattern $I \subseteq U$ are considered to be active while the states of the non-included input units is not fixed. I.e. remaining units in $U \setminus I$ might be active or inactive. Therefore, an input pattern defines an upper and lower bound on the possible input of the perceptron.

Definition 6 (Input pattern). *Let $\mathcal{P}_A$ be a perceptron and U be the set of input units. A subset $I \subseteq U$ is called an* input pattern. *The minimal and maximal input wrt. the input pattern I are defined as $i_{\min}(I) = \sum_{i \in I} w_{iA} - \sum_{u \in U \setminus I} |w_{uA}|$ and $i_{\max}(I) = \sum_{i \in I} w_{iA} + \sum_{u \in U \setminus I} |w_{uA}|$, respectively.*

Example 7. *For $\mathcal{P}_G^+$ from Figure 4 and $I = \{C, D\}$ we have $i_{\min}(I) = 1.0 + 2.0 - 3.0 - 5.0 = -5.0$ and $i_{\max}(I) = 1.0 + 2.0 + 3.0 + 5.0 = 11.0$.*

Next, we will introduce *coalitions* and *oppositions* as special types of input patterns. While coalitions can be seen as conditions to make a perceptron active, opposition prevent a perceptron from getting active. If some input patterns is a coalition then the perceptron will be active, independent of the state of all non-clamped input units. If it is an opposition, the perceptron will always by inactive.

Definition 8 (Coalition). *Let $\mathcal{P}_A$ be a perceptron with threshold θ, I be some input pattern for $\mathcal{P}_A$ and $i_{\min}(I)$ be the corresponding minimal input. I is called a* coalition*, if $i_{\min}(I) \geq \theta$. A coalition I is called* minimal*, if none of its subsets $I' \subset I$ is a coalition.*

Definition 9 (Opposition). *Let $\mathcal{P}_A$ be a perceptron with threshold θ, I be some input pattern for $\mathcal{P}_A$ and $i_{\max}(I)$ be the corresponding maximal input. I is called an* opposition*, if $i_{\max}(I) < \theta$. An opposition I is called* minimal*, if none of its subset $I' \subset I$ is an opposition.*

Example 10. *For $\mathcal{P}_G^+$ from Figure 4, we find $I = \{C, D, F\}$ to be a coalition, as $i_{\min}(I) = 1.0 + 2.0 + 5.0 - 3.0 = 5.0 > 4.0$. For $\mathcal{P}_G^-$, we find $J = \{\bar{F}\}$ to be an opposition, as $i_{\max}(J) = 1.0 + 2.0 + 3.0 - 5.0 = 1.0 < 4.0$.*

The set of coalitions and the set of oppositions can be used to describe the behavior of a perceptron. Furthermore, it is sufficient to consider the set of minimal coalitions and minimal oppositions respectively, which are uniquely determined. Those are the results of the extraction algorithm presented in the next section.

3.2 Extracting Coalitions and Oppositions

Here, we will show how to construct the set of minimal coalitions for a given perceptron. To keep notions and algorithms simple, we will first consider positive perceptrons only. At the end of the section we will show how to extract the set of minimal oppositions from a negative perceptron, and furthermore, how to apply the algorithms to arbitrary perceptrons. Before presenting the algorithms, we will try to convey some underlying intuitions. For positive perceptrons with inputs from $\{-1, +1\}$ only, we observe the following:

1. The empty input pattern (no unit needs to be active) generates the smallest minimal input.
2. The full input pattern (all inputs are active) generates the biggest minimal input.
3. If an input pattern is a coalition, all supersets are coalitions as well.

Starting from the empty input pattern (observation 1), input symbols are added according to their weights, such that inputs with larger weights are added first. If all inputs are added, but no coalition is found, we can conclude that there is none (observation 2). As soon as a coalition is found, all supersets are known to be coalitions as well (observation 3) and the algorithm can continue with adding the next-smaller input instead. Algorithm 1 constructs a search tree used to guide the extraction. Each node of the tree represents the input pattern containing all symbols on the path to the root.

Example 11. *For the perceptron $\mathcal{P}_G^+$ shown in Figure 3, we have $w_{CG} = 1.0, w_{DG} = 2.0, w_{EG} = 3.0$ and $w_{FG} = 5.0$, therefore, we determine the order $F \succ E \succ D \succ C$. Applying Algorithm 1, we obtain the search tree shown in Figure 5 on the left. For $\mathcal{P}_H^+$, we have $w_{\bar{D}G} = 3.0$ and $w_{FG} = 2.0$, therefore, we determine the order $\bar{D} \succ F$ and obtain the tree shown in Figure 5 on the right.*

As mentioned above, while looking for a coalition we will generate input patterns by adding symbols according to their weights. This can be done by left-depth-first search using the tree just constructed. The following two rules are used to prune the tree and hence the search space:

1. If the minimal input of a node is above threshold, cut all children.
2. If the minimal inputs of a node and all its descendants are below the threshold, cut all right siblings.

Rule 1 reflects Observation 3 from above, because child nodes represent supersets of the corresponding input pattern. If a certain node and none of its children is a coalition, we can cut all right-hand siblings, as their minimal inputs will be even smaller. This follows from the order of the symbols used while constructing the tree. The complete extraction of the smallest set of minimal coalitions is given as Algorithm 2.

Input: A positive perceptron $\mathcal{P}_A^+$.
Output: A coalition search tree suitable for Alg. 2.

```
1  Fix an order ≻ on the input units such that: if
   w_BA ≥ w_CA then B ≻ C.
2  Create a root node (representing the empty pattern).
3  Add a child labeled X for each input symbol X (sorted
   left to right descending wrt. ≻).
4  foreach new child labeled Y do
5      add a new sub-child for every symbol Z with
       Y ≻ Z (descending sorted wrt. ≻).
```

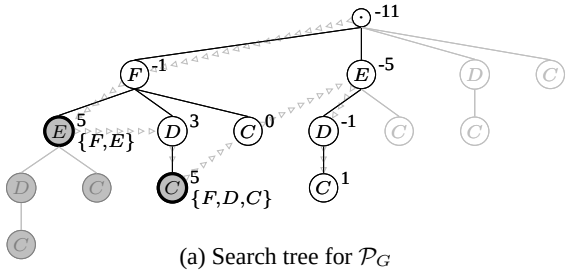
Algorithm 1: Construction of the coalition search tree.

Input: A positive perceptron $\mathcal{P}_A^+$ with threshold θ_A.
Output: The set of minimal coalitions.

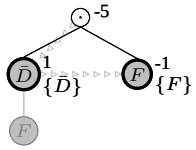
```
1  Construct the search tree for P_A^+ using Algorithm 1.
2  Make the root node the current node.
3  while there is a current node do
4      Compute i_min for the current node.
5      if i_min ≥ θ_A then
6          Mark the current node as coalition and cut all
           children (Pruning rule 1).
7      else
8          if the current node has no child then
9              while the parent of the current node is the
               direct predecessor do
10                 Cut all unvisited siblings of the parent.
11                 Use parent as current node.
12     Make the next unvisited node (left-depth-first) the
       current node.
13     if there is no unvisited node then stop.
14 Return the set of coalitions corresponding to the
   marked nodes.
```

Algorithm 2: Constructing minimal coalitions.

Example 12. *For the perceptrons $\mathcal{P}_G^+$ and $\mathcal{P}_H^+$ shown in Figure 4, Algorithm 2 returns $\mathcal{C}_G = \{\{E, F\}, \{C, D, F\}\}$ and $\mathcal{C}_H = \{\{\bar{D}\}, \{F\}\}$.*



(a) Search tree for $\mathcal{P}_G$



(b) Search tree for $\mathcal{P}_H$

Figure 5: The search trees (a) for the extraction of the perceptrons G (with $\theta_G = 4$ and $F \succ E \succ D \succ C$); and (b) for H (with $\theta_H = -2$ and $\bar{D} \succ F$). (Minimal) coalitions are marked with a gray background (and a thick border) and unvisited nodes are shaded. Every node corresponds to the input pattern containing all symbols on the path to the root, as exemplified for the two minimal coalitions. The numbers denote the corresponding minimal input. The triangular lines show the path taken by Algorithm 2,

Even though the worst case complexity is exponential[1], the algorithm performs reasonably well, as demonstrated in Section 4. This follows from the effectiveness of the pruning rules, and as a consequence, from the fact that the search tree does not need to be computed completely.

While using $+1$ and -1 as activation values and the positive form for the extraction of coalitions, we find that the extraction of opposition is "dual" while working on the negative form of the perceptron. Therefore, we will list the differences only:

- For oppositions negative perceptrons are used as inputs.
- In Algorithm 1, the order must be reversed, i.e. if $w_{BA} \leq w_{CA}$ then $B \succ C$.
- In Algorithm 2, instead of computing the minimal input, we would compute the maximal input and check whether it is below the threshold.

Example 13. *Applying the modified algorithm to the perceptron $\mathcal{P}_D$ (i.e. unit D from Figure 1 with its incoming connections) yields $\mathcal{O}_D = \{\{A\}, \{\bar{B}\}\}$.*

We used the positive form of a perceptron to extract coalitions and the negative form to extract oppositions. For the sequel we will understand a negated input symbol occurring in some input pattern to clamp the corresponding input unit as inactive. Note that thus an input pattern can be used for the

[1]Assume a perceptron with n equal weights and with a threshold of 0. Then there are $\binom{n}{\lceil n/2 \rceil}$ coalitions.

Unit	Minimal Coalitions	Minimal Oppositions
C	$\{\{A\}, \{B\}\}$	$\{\{\bar{A}, \bar{B}\}\}$
D	$\{\{\bar{A}, B\}\}$	$\{\{A\}, \{\bar{B}\}\}$
E	$\{\{A, \bar{B}\}\}$	$\{\{\bar{A}\}, \{B\}\}$
F	$\{\emptyset\}$	$\emptyset$
G	$\{\{E, F\}, \{C, D, F\}\}$	not needed
H	$\{\{\bar{D}\}, \{F\}\}$	not needed

Table 1: Minimal coalitions and oppositions for the network from Figure 1.

original perceptron as well as for the positive and negative form. This allows us to apply the algorithms to arbitrary perceptrons[2]. Table 1 shows all intermediate extraction results.

3.3 Composition of the Intermediate Results

In this section, we will show how to compose the intermediate results to obtain a description of the output unit's behavior wrt. the input units.

The intended meaning of a set of coalitions like $\mathcal{C}_G = \{\{E, F\}, \{C, D, F\}\}$ is, that "E and F", or "C, D and F" should be active in order to make neuron G active, this can be represented as the propositional formula $((E \wedge F) \vee (C \wedge D \wedge F))$. We will refer to the propositional formula obtained from a set of coalitions $\mathcal{C}_F$ as $\mathrm{pf}(\mathcal{C}_F)$. If there is no coalition for a given perceptron $\mathcal{P}_F$, i.e. $\mathcal{C}_F = \emptyset$, we can conclude that there is no input such that F will be active, hence $\mathrm{pf}(\mathcal{C}_F) = \mathrm{false}$. In contrast, for $\mathcal{C}_F = \{\emptyset\}$, we can conclude that F will always be active, hence $\mathrm{pf}(\mathcal{C}_F) = \mathrm{true}$. Analogously, the intended meaning of a set of oppositions like $\mathcal{O}_D = \{\{A\}, \{\bar{B}\}\}$ is, that whenever A is active or B is inactive, the neuron D will be inactive. This can be represented as $(A \vee \bar{B})$. Again, we will refer to the corresponding formula as $\mathrm{pf}(\mathcal{O}_F)$.

Algorithm 3 takes a feed-forward network and one output unit A and returns a propositional formula describing A's behavior with respect to the network's input units. It will create and manipulate a propositional formula $\mathcal{F}$, which finally can be rewritten as a logic program.

Input: A network N and an output unit A.
Output: A formula describing A's behavior.

1 Initialize $\mathcal{F}$ as $\mathcal{F} = \mathrm{pf}(\mathcal{C}_A)$.
2 **while** *there occur symbols in $\mathcal{F}$ referring to non-input units of N* **do**
3 Pick one occurrence of a (possibly negated) non-input symbol B.
4 **if** B *is negated* **then** Replace $\bar{B}$ with $\mathrm{pf}(\mathcal{O}_B)$ **else** Replace B with $\mathrm{pf}(\mathcal{C}_B)$.
5 Return $\mathcal{F}$.

Algorithm 3: Extracting one unit of a given network

[2]As mentioned above, the positive and negative forms were introduced to keep notions and algorithms simple. They will actually never be constructed in a real implementation. Instead, the algorithms could be modified by adding case distinctions.

Example 14. *Applying Algorithm 3 to the network from Figure 1 we obtain*[3]*:*

$$\begin{aligned} G &= (E \wedge F) \vee (C \wedge D \wedge F) \\ &= ((A \wedge \bar{B}) \wedge \mathrm{true}) \vee ((A \vee B) \wedge (\bar{A} \wedge B) \wedge \mathrm{true}) \\ &\big[= (A \wedge \bar{B}) \vee (\bar{A} \wedge B) \big] \\ H &= (\bar{D} \vee F) \\ &= ((A \vee \bar{B}) \vee \mathrm{true}) \\ &\big[= \mathrm{true} \big] \end{aligned}$$

Note that the formulae $G = (A \wedge \bar{B}) \vee (\bar{A} \wedge B)$ *and* $H = \mathrm{true}$ *could also be represented as program* P_2 *from Example 2.*

The non-determinism introduced in line 3 of Algorithm 3 is a don't-care non-determinism, i.e. we are free to choose any symbol without changing the result. But an "informed heuristic" could speed up the extraction. In Example 14 we applied the usual laws of propositional logic after applying Algorithm 3. In fact, those rules can also be applied before, i.e. directly after replacing a symbol with the corresponding coalition or opposition.

Proposition 15. *Algorithm 3 is sound and complete.*

I.e. for a given feed-forward network and a given output unit A, the algorithm always returns a correct formula describing A's behavior wrt. to the network's input units. The proposition follows from the fact that the network contains no cycles and from the correctness of the laws of propositional logic.

4 DISCUSSION

In this section, we will briefly discuss some related work, the extension of the approach to non-binary units and report on some preliminary experimental results.

4.1 Related Work

As mentioned in the introduction, our approach is closely related to the *COMBO* algorithm introduced in [Krishnan *et al.*, 1999]. But, in contrast to that approach, the search tree can be built incrementally. Each level of the *combination trees* constructed for the *COMBO* algorithm needs to be sorted wrt. the minimal input. This involves the construction, evaluation and sorting of possibly exponentially many nodes of the tree, even though they might be cut of.

4.2 Extension to Non-Binary Units

We are currently investigating the applicability of the COOP-approach to non-binary units. A first approach, which was taken in the experiments described below, employes bipolar sigmoidal ($\tanh$) units instead of binary ones. Consequently, the network becomes trainable by standard learning methods, like back-propagation. After some iterations, all weights were multiplied by 2, yielding steeper activation functions. We stopped the training process whenever the error made by the network did not increase significantly after multiplying the weights. In this case units behave like binary ones, i.e. small activation values do not occur any more, only values close to -1 and $+1$. Therefore, we can treat those units as binary threshold units and apply our algorithms directly.

As a second approach, we try to adapt the following idea from [d'Avila Garcez *et al.*, 2001]. A unit is considered active (inactive), whenever its activation value is above (below) some threshold $a_{\min}$ ($a_{\max}$). For a given $a_{\min}$, we can compute a necessary minimal input $i_{\min}$. For a given perceptron with threshold θ, we could now apply our algorithm, by using $\theta - i_{\min}$ as threshold and instead of using $+1$ and -1 as activation vales, we use $a_{\min}$ and $a_{\max}$. This allows to apply the algorithms described above. We believe, that soundness will be preserved, but whether this holds for completeness as well will be investigated in the future.

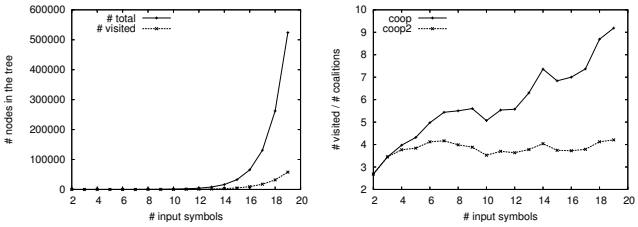


Figure 6: The plot on the left shows the total number of nodes in the tree and the average number of visited nodes wrt. the number of input symbols. The plot on the right shows the ratio of visited nodes and found coalitions.

4.3 Some Preliminary Experimental Results

To evaluate the average runtime behavior of the algorithm, we generated random perceptrons for which we computed the number of visited nodes wrt. the number of input symbols. This, together with the total number of nodes in the tree, is depicted in Figure 6 on the left.[4] The plot shows that only a small fraction of nodes is visited. Nevertheless, the number of visited nodes seems to grow exponentially. This is not surprising as the number of minimal coalitions grows exponentially as well.

Furthermore, we computed the ratio of visited nodes and coalitions found – again wrt. the number of input symbols. The results are shown in Figure 6 on the right. This test indicates, that the number of visited nodes wrt. found coalitions seems to increase for a larger number of input symbols. For the variant "coop2", this ratio seems to stabilize around 4, i.e. the algorithm needs to visit 4 times as many nodes as it finds coalitions. In this variant we used some more techniques to improve the pruning rules, which are beyond the scope of this paper. E.g., we cached the minimal input values necessary before entering a node in the tree. If this input is not reached, the subtree can be pruned. Furthermore, we tried to identify equivalent sub trees.

[3] Note, that several replacements were done in one line and parentheses were omitted if unnecessary. The last lines were obtained by applying the usual laws of propositional logic.

[4] Instead of measuring the time, we used the number of nodes, because we used a very preliminary implementation in Prolog only.

4.4 The Monks Problem

The monks problems as described in [Thrun *et al.*, 1991], are learning problems where robots are described by the following six attributes:

- *head shape* is {*round* (a_1), *square* (a_2), *octagon* (a_3)},
- *body shape* is {*round* (b_1), *square* (b_2), *octagon* (b_3)},
- *is smiling* is {*yes* (c_1), *no* (c_2)},
- *holding* is {*sword* (d_1), *balloon* (d_2), *flag* (d_3)},
- *colour* is {*red* (e_1), *yellow* (e_2), *green* (e_3), *blue* (e_4)},
- *has tie* is {*yes* (f_1), *no* (f_2)}.

The following three classifications are to be learned:

1. *head shape* = *body shape* or the *colour* is *red*
2. exactly two of the six attributes take their first value
3. *colour* = *green* and *holding* = *sword*, or the *colour* $\neq$ *blue* and *body shape* $\neq$ *octagon*

We used bipolar sigmoidal networks with 17 input units (labeled $a_1, \ldots, f_2$) a single output unit (labeled cl) and either 1 (problem 1) or 2 (problem 2 and 3) hidden units. Furthermore, we allowed shortcut connections from the input to the output layer. These architectures were chosen to minimize the size of the networks. We used a single train-test set, containing all available examples. After training the networks and multiplying the weights as described above, we applied the COOP algorithm to extract the single perceptrons. Afterwards, the results were composed as described above and further refined using the integrity constraints resulting from the encoding (i.e., e_1 and e_2 can not be active simultaneously). Finally, we obtained the following logic programs:

$$
\begin{aligned}
P_1 = \{ & cl \leftarrow a_1 \wedge b_1. \\
& cl \leftarrow a_2 \wedge b_2. \\
& cl \leftarrow a_3 \wedge b_3. \\
& cl \leftarrow e_1. \} \\
P_2 = \{ & cl \leftarrow a_1 \wedge b_1 \wedge \bar{c_1} \wedge \bar{d_1} \wedge \bar{e_1} \wedge \bar{f_1} \\
& cl \leftarrow a_1 \wedge \bar{b_1} \wedge c_1 \wedge \bar{d_1} \wedge \bar{e_1} \wedge \bar{f_1} \\
& \ldots 11 \text{ clauses more} \\
& cl \leftarrow \bar{a_1} \wedge \bar{b_1} \wedge \bar{c_1} \wedge d_1 \wedge \bar{e_1} \wedge f_1 \\
& cl \leftarrow \bar{a_1} \wedge \bar{b_1} \wedge \bar{c_1} \wedge \bar{d_1} \wedge e_1 \wedge f_1 \} \\
P_3 = \{ & cl \leftarrow d_1 \wedge e_3 \\
& cl \leftarrow \bar{b_3} \wedge \bar{e_4} \}
\end{aligned}
$$

The programs describe the required classifications. E.g. program P_1 encodes: *head shape* = *body shape* ($a_1 \wedge b_1$ or $a_2 \wedge b_2$ or $a_3 \wedge b_3$) or the *colour* is *red* (e_1). This shows, that the COOP-approach is able to extract meaningful rules from a trained neural network, even though this is just a preliminary experiment on some artificial domain.

5 CONCLUSIONS

In this paper, we presented a new decompositional approach to extract propositional if-then rules from a feed-forward network of binary threshold units. Our approach is sound and complete, i.e. every rule extracted from the network is correct and all contained rules are extracted. Even though our running example is a 3 layered feedforward network, the approach is not limited to layered architectures, but rather to cycle-free networks.

For the extraction algorithm of a single perceptron (Section 3.2) we will further investigate, whether ideas underlying the "M-of-N" approach by Towel and Shavlik [Towell and Shavlik, 1993] can help to speed up the system. Furthermore, we will try to develop some dedicated "informed heuristics", as mentioned in Section 3.3, to guide the extraction on the level of whole networks. Another candidate for further improvement is the caching of intermediate results while composing the coalitions and oppositions.

First experiments, presented in Section 4, indicate that our approach shows a good average-complexity, but a detailed analysis needs to be done in the future. Furthermore, we would like to evaluate our algorithm and compare it to other approaches using benchmark problems, like the Monks-problem or problems from molecular biology as described in [d'Avila Garcez *et al.*, 2001].

Acknowledgments

We would like to thank two anonymous referees for their valuable comments on a preliminary version of this paper. Sebastian Bader is supported by the GK334 of the German Research Foundation (DFG).

References

[Bishop, 1995] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[d'Avila Garcez *et al.*, 2001] Artur S. d'Avila Garcez, Krysia Broda, and Dov M. Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, 125:155–207, 2001.

[Krishnan *et al.*, 1999] R. Krishnan, G. Sivakumar, and P. Bhattacharya. A search technique for rule extraction from trained neural networks. *Non-Linear Anal.*, 20(3):273–280, 1999.

[Lloyd, 1988] John W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1988.

[Rojas, 1996] Raul Rojas. *Neural Networks*. Springer, 1996.

[Thrun *et al.*, 1991] S. Thrun *et al.* The MONK's problems: A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, 1991.

[Tickle *et al.*, 1998] Alan. B. Tickle, Robert Andrews, Mostefa Golea, and Joachim Diederich. The truth will come to light: directions and challenges in extracting the knowledge embedded within mined artificial neural networks. *IEEE Transactions on Neural Networks*,, 9(6):1057–1068, 1998.

[Towell and Shavlik, 1993] Geoffrey G. Towell and Jude W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13:71–101, 1993.