Extracting Propositional Rules from Feed-forward Neural Networks by Means of Binary Decision Diagrams

Sebastian Bader
Department of Computer Science, University of Rostock, Germany
sebastian.bader@uni-rostock.de

Abstract

We discuss how to extract symbolic rules from a given binary threshold feed-forward network. The proposed decompositional approach is based on an internal representation using binary decision diagrams. They allow for an efficient composition of the intermediate results as well as for an easy integration of integrity constraints into the extraction. We also discuss some experimental results indicating a good performance of the approach.

1 Introduction

During the training process, neural networks acquire knowledge by generalising from raw data. Unfortunately, this learnt knowledge is hidden in the weights associated to the connections and humans have no direct access to it. One goal of rule extraction is the generation of a human-readable description of the output units behaviour with respect to the input units. Usually, the result is described in form of if-then rules, giving conditions that activate (or inactivate) a given output unit. Rule extraction from connectionist systems is still an open research problem, even though a number of algorithms exists. For an overview of different approaches we refer to [Andrews *et al.*, 1995] and [Jacobsson, 2005]. Extraction techniques can be divided into *pedagogical* and *decompositional* approaches. While the first conceives the network as a black box, the latter decomposes the network, constructs rules describing the behaviour of the simpler parts, and then re-composes those results.

In [Bader *et al.*, 2007], we proposed the CoOp-algorithm, a decompositional approach for the extraction of propositional rules from feed-forward neural networks. Here, we discuss an extension of this approach. In this new extension, binary decision diagrams (BDD) are used to store intermediate results, i.e., rules extracted from single units (perceptrons). This representation has three advantages:

1. results are stored in a very compact form,
2. intermediate results can easily be combined, and
3. integrity constraints can easily be incorporated.

After a presentation of all necessary concepts in Section 2, we discuss the proposed extension of the CoOp-approach, i.e., the extraction of BDDs from simple perceptrons. Afterwards, we discuss how to compose the intermediate results and how to incorporate integrity constraints. In Section 6 first experimental results are presented and further work is discussed in Section 7.

2 Preliminaries

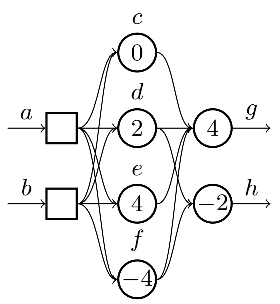
In this section, we introduce some necessary concepts. After defining feed-forward neural network and the rule-extraction problem, we discuss binary decision diagrams.

Feed-forward artificial neural networks (ANNs), also called *connectionist systems*, consist of simple computational units (neurons) which are connected. The set of units $\mathcal{U}$ together with the connections $\mathcal{C} \subseteq \mathcal{U} \times \mathcal{U}$ form an acyclic directed graph. In this paper we concentrate on networks with units applying the ± 1-threshold function. Such a neural network can be represented as a 6-tuple $\langle \mathcal{U}, \mathcal{U}_{\mathsf{inp}}, \mathcal{U}_{\mathsf{out}}, \mathcal{C}, \omega, \theta \rangle$. $\mathcal{U}_{\mathsf{inp}}, \mathcal{U}_{\mathsf{out}} \subseteq \mathcal{U}$ denote input and output units of the network, i.e., are sources and sinks of the underlying graph. The functions $\omega : \mathcal{C} \to \mathbb{R}$ assign a weight to every connection and $\theta : \mathcal{U} \to \mathbb{R}$ a threshold to every unit. Every unit u has an activation value $\mathsf{act}_u \in \{-1, +1\}$ which is set from outside for input units, or computed based on the activation value of its predecessor units and the threshold $\theta(u)$ as follows:

$$\mathsf{act}_u = \begin{cases} +1 & \text{if } \sum_{c=(v,u)\in\mathcal{C}} \mathsf{act}_v \cdot \omega(c) \geq \theta(u) \\ -1 & \text{otherwise} \end{cases}$$

Figure 1 shows a simple network serving as running example throughout the paper.

Because every unit can be active ($\mathsf{act}_u = +1$) or inactive ($\mathsf{act}_u = -1$) only, we can associate a propositional variable u to it, which is assumed to be true if and only if the unit u is active, and we use $\bar{u}$ to denote the negation of u. Furthermore, we can characterise network inputs as interpretations $I \subseteq \mathcal{U}_{\mathsf{inp}}$ of the propositional variables $\mathcal{U}_{\mathsf{inp}}$. We use $\mathsf{act}_u(I)$ to denote the state of unit u if all input units contained in I are active and all other input units are inactive. Using this notation, we can define the rule extraction problem as follows: The *rule extraction problem* for a given node u of a feed-forward



ω	c	d	e	f	g	h
a	1	-2	5	2		
b	1	1	-3	1		
c					1	0
d					2	-3
e					3	0
f					5	-2

Figure 1: A simple network serving as running example. The threshold are shown within the nodes and the connection weights are shown on the right.

threshold network $\langle \mathcal{U}, \mathcal{U}_{\mathsf{inp}}, \mathcal{U}_{\mathsf{out}}, \mathcal{C}, \omega, \theta \rangle$ is the construction of a propositional formulae F over $\mathcal{U}_{\mathsf{inp}}$ such that for all interpretations I we find

$$\mathsf{act}_u(I) = \begin{cases} +1 & \text{if } I \models F \\ -1 & \text{otherwise} \end{cases}$$

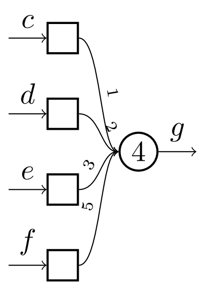
I.e., we are looking for propositional formula stating necessary and sufficient conditions (in terms of the activation of input units) such that the unit u is active.

Usually not all input combinations make sense in a given application domain, because some of them would correspond to invalid states of the world. We use the term *valid inputs* to denote the set of allowed input combinations. Even more important for the extraction is the fact that all training samples are taken from this subset. Therefore, the network learns to solve a task under the implicit conditions hidden in the selection of inputs. *Integrity constraints* are a way to make those conditions explicit during the extraction. An integrity constraint is a formula IC over $\mathcal{U}_{\mathsf{inp}}$ describing the set of valid inputs $V \subseteq \mathcal{P}(\mathcal{U}_{\mathsf{inp}})$ as follows: For all $I \subseteq \mathcal{U}_{\mathsf{inp}}$ we find $I \models \text{IC}$ if and only if $I \in V$. Using integrity constraints, we can reformulate the extraction problem as follows: The rule extraction problem for a given network N and a given integrity constraint IC is the construction of a propositional formulae F over $\mathcal{U}_{\mathsf{inp}}$ such that for all interpretations I with $I \models \text{IC}$ we find

$$\mathsf{act}_u = \begin{cases} +1 & \text{if } I \models F \\ -1 & \text{otherwise} \end{cases}$$

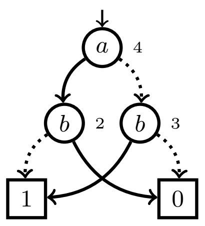
Networks can be decomposed into their basic building blocks, namely single units together with their incoming connections. Those single units can be seen as simple sub-networks (perceptrons) consisting of a number of inputs and a single output unit, together with the corresponding weighted connections. To simplify the notations we use $\mathcal{P}_p = \langle \theta, \mathcal{I}, \omega \rangle$ to denote the perceptron corresponding to the unit p together with its threshold θ, the set of predecessor units $\mathcal{I}$ and the weight function ω. Figure 2 shows the perceptron for the output unit g.

Binary decision diagrams (BDD) are a data structure to represent propositional formulae in a very compact way and to manipulate them easily. A nice introduction



$$\begin{aligned} \mathcal{P}_g &= \langle \theta, \mathcal{I}, \omega \rangle \text{ with} \\ \theta &= 4 \\ \mathcal{I} &= \{c, d, e, f\} \\ \omega(x) &= \begin{cases} 1 & \text{if } x = c \\ 2 & \text{if } x = d \\ 3 & \text{if } x = e \\ 5 & \text{if } x = f \end{cases} \end{aligned}$$

Figure 2: The perceptron corresponding to unit g.



$$\begin{aligned} BDD &= \langle \prec, 0, 1, R, N \rangle \text{ with} \\ R &= 4 \\ N &= \{\langle 2, b, 0, 1 \rangle, \langle 3, b, 1, 0 \rangle, \\ &\quad \langle 4, a, 2, 3 \rangle\} \\ \text{pf}(2) &= (b \wedge \bot) \vee (\neg b \wedge \top) = \neg b \\ \text{pf}(3) &= (b \wedge \top) \vee (\neg b \wedge \bot) = b \\ \text{pf}(4) &= (a \wedge \neg b) \vee (\neg a \wedge b) \end{aligned}$$

Figure 3: A simple BDD, nodes are annotated with their variables and their ID on the right. High branches are depicted as solid and low branches as dashed lines. On the right you find the underlying data structure and the logic formulae corresponding to the internal nodes.

can be found for example in [Andersen, 1999]. Intuitively, a BDD is a directed acyclic graph with a variable associated to every node and such that all nodes $n \neq 0, 1$ have exactly two successors, called high and low branch of n. The nodes 0 and 1 are the sinks of the graph. We use $\langle \prec, 0, 1, R, N \rangle$ to refer to a BDD with sinks 0 and 1, a set of nodes N and a root-node with identifier R. And we use $\langle i, v, h, l \rangle$ to denote the node with identifier i, with variable $v = \text{var}(i)$, and with high and low branch pointing to the nodes with identifiers h and l, respectively.

Usually a BDD is assumed to be ordered and reduced. It is called ordered iff there exists a linear order $\prec$ on the variables and the successors of a node are marked with variables that are bigger with respect to $\prec$. It is called reduced if no two nodes for the same variable have identical high and low branch, and for no node high and low-branch coincide.

BDDs represent propositional formulae in if-then-else normal form. The corresponding formula for a given node is defined recursively as follows:

$$\text{pf}(0) := \bot \qquad \text{pf}(1) := \top$$
$$\text{pf}(i) := (\text{var}(i) \wedge \text{pf}(h)) \vee (\neg\, \text{var}(i) \wedge \text{pf}(l))$$

Figure 3 shows a simple BDD using a graphical representation as well as the underlying data structure and the corresponding logic formulae for every node.

3 From Perceptrons to Search Trees

In this section, we discuss an algorithm to extract a BDD from a single unit such that the BDD represents necessary and sufficient condition on the inputs to turn the unit active. Following [Bader *et al.*, 2007], we define input patterns I as subsets of the inputs $\mathcal{I}$ of a given perceptron $\mathcal{P}_p = \langle \theta, \mathcal{I}, \omega \rangle$ which are assumed to be active. The inputs not contained in I can be either active or inactive. And we define the corresponding minimal input $i_{\min}(I)$ as follows:

$$i_{\min}(I) = \sum_{a \in I} \omega(a) - \sum_{a \in \mathcal{I} \setminus I} |\omega(a)|$$

The minimal input is computed by adding the contribution of the fixed inputs $\sum_{a \in I} \omega(a)$ and the minimal input caused by all other inputs. A perceptron is called *positive*, if all weights are positive. For the following constructions, we assume the perceptrons to be positive. In Section 5, we discuss how to apply the extraction to arbitrary perceptrons.

The construction of BDDs below is based on the search trees described in [Bader *et al.*, 2007]. These search trees contain a node for every possible input pattern. Children of a given node correspond to input patterns which contain exactly one symbol more and all nodes are sorted with respect to their minimal inputs. If the minimal input of some node exceeds the threshold, that node is marked (I.e., the corresponding input pattern represents a sufficient condition to turn the perceptron active). The complete tree is pruned by removing all those nodes for which no descendant is marked and all those nodes which are descendants of marked nodes. The construction of a pruned tree is shown in Algorithm 1. Figure 4 shows the full and the resulting pruned search tree on top of it.

```
Input: A positive perceptron $\mathcal{P}_p^+$.
Output: A pruned search tree.
1 Fix an order $\prec$ such that $b \prec c$ if $\omega(b) \geq \omega(c)$.
2 Create a root node for the empty input pattern.
3 Add a child labelled $x$ for each input symbol $x$
  (sorted wrt. $\prec$).
4 foreach newly added node labelled $y$ do
5     Add a new child $c$ for every symbol $z$ with
      $y \prec z$ (sorted wrt. $\prec$).
6     Label $c$ with the corresponding pattern $I$.
7     Mark $c$ if $i_{\min}(I) > \theta(p)$.
8 Remove all descendants of marked nodes.
9 Remove all nodes for which no descendant is
  marked.
```

Algorithm 1: Constructing a pruned search tree.

Exploiting the structure of these search trees, we can easily construct BDDs representing conditions to turn the perceptron active. I.e., we find the perceptron to be active for all those input patterns which, understood as interpretation, turn the logic formula corresponding to the BDD true.

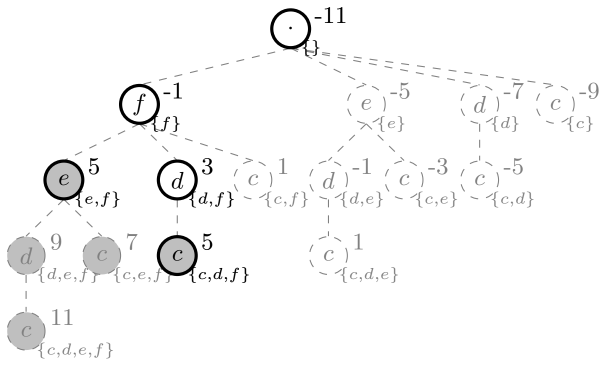


Figure 4: The pruned search tree for the perceptron $\mathcal{P}_g$ from above. The underlying full tree is depicted in grey using dashed lines. The nodes contain the newly added symbol and are annotated with the corresponding input pattern and the resulting minimal input. All nodes for which the minimal input exceed the threshold of $\theta(g) = 4$ are shown with grey background.

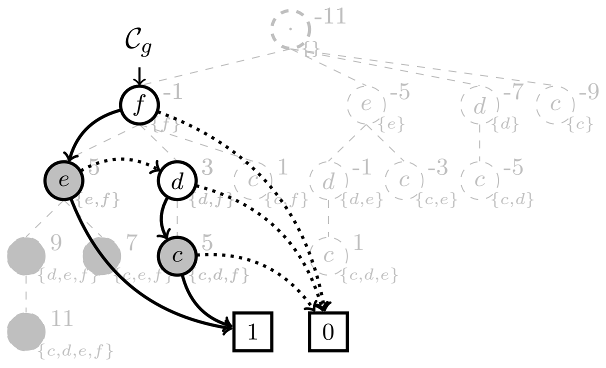


Figure 5: The BDD corresponding to the pruned search tree from Figure 4.

4 From Search Trees to BDDs

Before presenting an algorithm to construct BDDs from a given pruned search tree, we introduce some further notations. Every node in the search tree is represented as a pair $\langle I, C \rangle$ with I being the corresponding input pattern and C being the set of children. id(n) denotes a unique identifier for the node n (e.g., the corresponding input pattern, or some index), this identifier is also used as internal index for the BDD nodes. We assume id$(n) := 0$ if there is no node n. var(n) denotes the symbol which is added to the input pattern at node n.

The construction of a BDD for a given search tree is shown as Algorithm 2. This algorithm transforms a search tree into a BDD, by traversing the tree in a left-depth-first manner. A node's high branch points to 1, if its minimal input exceeds the threshold. Otherwise, it points to its left-most child, or to 0 if there is no child. The low-branch points to the right sibling, or to 0 if there is none. The result for the perceptron $\mathcal{P}_g$ is shown in Figure 5.

```
Input: A search tree $T$ for $\mathcal{P}_p = \langle \theta, \mathcal{I}, \omega \rangle$ wrt. $\prec$.
Output: A corresponding OBDD $\langle \prec, 0, 1, R, N \rangle$.
1  if $T$ is empty then
2      $R = 0$ and $N = \{\}$
3  else if $T$ contains only the root node then
4      $R = 1$ and $N = \{\}$
5  else
6      $R = \mathrm{id}(r_l)$ for the leftmost child of the root.
7      $N = \{\}$.
8      foreach leaf node $n$ in $T$ do
9          Add $\langle \mathrm{id}(n), \mathrm{var}(n), 1, l \rangle$ to $N$ with $l = \mathrm{id}(r)$
           for the right sibling $r$.
10     foreach node $\langle I, C \rangle$ in $T$ with left sibling $l$ do
11         Let $\langle \mathrm{id}(l), \mathrm{var}(l), h_l, l_l \rangle$ be the node for $l$
12         Let $\langle \mathrm{id}(l_l), \mathrm{var}(n), h_{l_1}, l_{l_1} \rangle$ be the node for
           the leftmost child $l_l$ of $l$
13         if $\mathrm{mci}(l) - 2\omega(l) + 2\omega(n) > \theta$ then
14             Add $\langle \mathrm{id}(n), \mathrm{var}(n), l_{l_1}, l_n \rangle$ to $N$ with
               $l_n = \mathrm{id}(r_n)$ for the right sibling $r_n$ of $n$
15     foreach other non-root node $\langle I, C \rangle$ do
16         Add a node $\langle \mathrm{id}(n), \mathrm{var}(n), \mathrm{id}(c), l \rangle$ to $N$ for
           the leftmost child $c$ and $l = \mathrm{id}(r_n)$ for the
           right sibling $r_n$ of $n$ and $l = 0$ if there is
           none
```
Algorithm 2: Constructing a BDD.

Please note, that the BDD can be constructed without constructing the search tree first. The tree is used only to describe the underlying ideas. All conditions tested in Algorithm 2 can be tested by expanding the tree step-by-step. Looking a little closer at the constructed search tree we find that some sub-trees have an identical internal structure, which is exemplified in Figure 6. If the condition tested in Line 13 of Algorithm 2 is fulfilled, two neighbouring sub-trees are structured identically. $\mathrm{mci}(n)$ denotes the minimum of all minimal inputs associated to nodes below n. Please note, that $\mathrm{mci}(n)$ can be computed without expanding the sub-tree by looking at the associated input pattern.

The mentioned structural equivalence can be exploited by using a shortcut into the already constructed BDD and thus preventing the expansion of an identical sub-tree. Figure 6 contains a number of those shortcuts, e.g., one from node $\{b\}$ to the node $\{c, a\}$, because the children of $\{b\}$ are annotated the same way as the node below and right of $\{c, a\}$. Please note that this identity can be recognised without expanding the second sub-tree, i.e., the construction of whole tree below $\{b\}$ can be avoided.

The condition on Line 13 is fulfilled whenever the perceptron shows a so called n-of-m behaviour, i.e., if there are m inputs from which n suffice to turn the perceptron active. In this case, there will be n equivalent sub-trees, which can be shortcut. As discussed in [Towell and Shavlik, 1993], this occurs quite frequently while training neural networks.

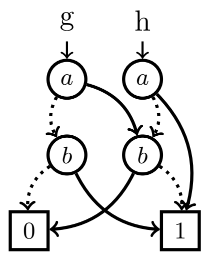


Figure 7: The global BDD for the network from Figure 1.

5 Composition of Intermediate Results

In the previous section, we have been concerned with positive perceptrons only. But we can easily turn every perceptron into a positive one, by multiplying negative weights by -1 and inverting the corresponding input symbols. By doing so, we can apply the algorithm to all output units of a given network, and obtain a BDD describing necessary conditions with respect to the predecessor units that turn the output unit active. But some of the input symbols may have been inverted. I.e., we need another algorithm to construct BDDs stating conditions which turn a perceptron inactive. Due to the symmetry of the threshold function, we find this algorithm to be dual to Algorithm 1 and 2. I.e., by inverting the order and the inequalities we obtain an algorithm that constructs such a BDD.

Once we have extracted the BDD for a given output unit, we can continue by substituting the nodes testing non-input nodes (i.e., nodes not corresponding to input units of the network) by their corresponding BDDs. A non-negated node is replaced by the BDD as constructed above, and negated nodes are replaced by the dual BDDs. As mentioned above, BDDs have been designed to allow for an efficient manipulation of logic formulae. And in fact it is straightforward to compose the intermediate results into an overall diagram by simply replacing the nodes. But this is not the best approach, because the resulting 'global' BDD would not be ordered any more. But while expanding the BDD, we can keep it ordered (and reduced) as described in [Andersen, 1999]. After expanding all non-input nodes, we obtain a final BDD representing necessary and sufficient conditions on the network's input to turn a given output unit active.

Using BDDs as internal data structure has some further advantages. We can actually extract all output units into the same global BDD. Doing so leads automatically to a sharing of intermediate results, because common substructures are contained only once within this BDD. Figure 7 shows the final 'global' BDD for the network from Figure 1. Please note that the right node labelled b is used for both output units g and h.

Furthermore, we can integrate integrity constraints in a straightforward fashion. Instead of starting with an empty BDD, we extract the output nodes into a BDD representing the integrity constraints. To exemplify this, a network has been trained to the Encode-Decoder task. It contains 8 input, 8 output units and 3 hidden units

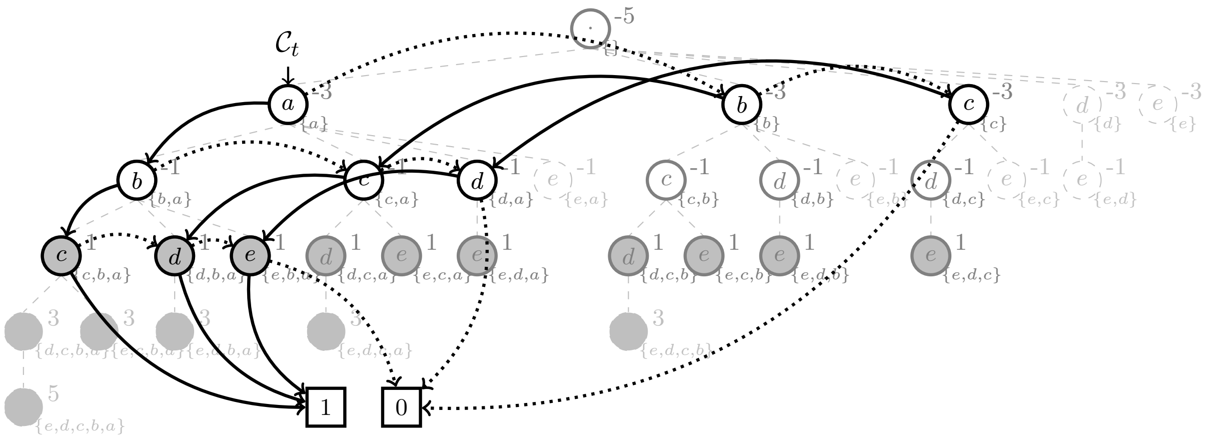


Figure 6: A larger BDD and its underlying search tree. The tree has been constructed for a perceptron with 5 inputs whose weights are all 1 and with threshold 0. I.e., this perceptron is active if 3 of the five inputs are +1. Please note that there are certain symmetries in the tree: All label and minimal inputs of the sub-trees of node $\{b\}$ coincide with those of the right sub-trees of $\{a\}$ if a is substituted by b. This has been exploited by linking from $\{b\}$ to $\{c, a\}$.

and is trained to learn the identity mapping for all inputs in which exactly one unit is active. I.e., the network has to learn a compressed representation within the hidden layer. But applying the algorithm presented above yields an unwanted result shown in Figure 8 on the left. Using the integrity constraint that at most one input is active at a time yields the BDD shown on the right. 900 nodes have to be constructed (including all intermediate results while constructing the BDD) for the 'normal' BDD, but only 124 while using the integrity constraint. This shows the advantage of using integrity constraints right from the beginning of the extraction process. Usually they are used to refine the extraction result afterwards. This would be possible here as well by simple computing the conjunction of the 'normal' BDD with one representing the integrity constraint. But starting with the constraint avoids the construction of many intermediate nodes which would be removed afterwards.

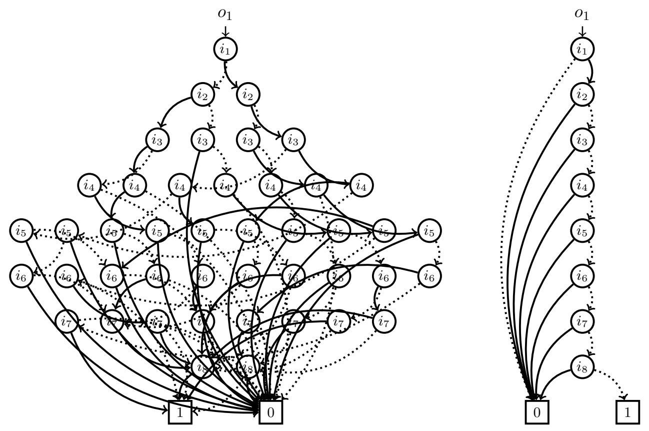


Figure 8: The result of extracting output unit 1 from an 8-3-8 encoder-decoder network. The BDD on the left is the result of the 'normal' extraction. On the right the constraint that at most one unit is active has been incorporated.

6 Experimental Evaluation

To evaluate the approach a Prolog implementation has been used to gather some statistics. The results are shown in Table 1. The table shows average numbers for different numbers of inputs, the size of the full search tree, the number of minimal input patterns, the size of the corresponding BDD and the number of BDD nodes per input pattern. All numbers have been collected from 100 random perceptrons per size. The extraction using the full search tree is not feasible due to the exponential growth. The number of input patterns is a conservative lower bound for the size of the pruned search tree, because those trees have at least one node per minimal input pattern. The result shows that the use of BDD proposed here yields a very compact representation. Even though the number of nodes in the BDD grows, the ratio (node/IP) of size of the BDD and the number of minimal coalitions decreases.

$\lvert\mathcal{I}\rvert$	|T|	#IPs	|BDD|	node/IP
1	2	0.58	2.58	4.448
5	32	4.31	8.49	1.969
10	1024	63.51	49.97	0.786
15	32768	1270.45	313.25	0.246
20	1048576	25681.70	1863.90	0.072

Table 1: The size of the full search tree (|T|), the number of minimal input patterns as a lower bound for the size of the pruned search tree (#IPs) and the corresponding BDDs (|BDD|) for different number of inputs ($\lvert\mathcal{I}\rvert$).

A second experiment has been performed to show the effect of the usage of integrity constraints while extracting the BDDs. A network with 6 inputs, 4 hidden and 2

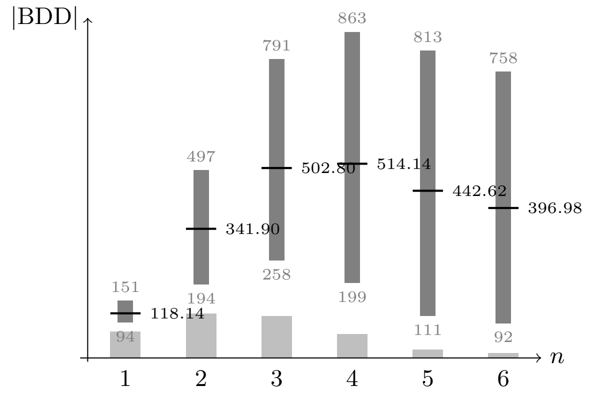


Figure 9: Resulting BDD sizes of the extraction for different $\max_n$-integrity constraints. The bars indicate minimal, maximal and average sizes of the BDDs. The size of the sub-BDD for the constraint is shown in grey.

output units has been used for the experiment. The possible inputs have been constrained by a $\max_n$ integrity constraint for $0 \leq n \leq 6$. The results are presented in Figure 9. For every n the experiment has been conducted for the same 100 randomised networks and the following numbers have been collected: the size of the sub-BDD encoding the constraint, the minimal, maximal and average size of the final BDD. Please note that the numbers show the total number of internal nodes constructed for the BDD, i.e., including all necessary intermediate nodes. For $n = 1$, i.e., the biggest restriction, we obtain very small BDDs. The size of the BDD grows up to $n = 4$ and decreases again for $n > 4$. From those observations we can conclude that the incorporation of integrity constraints into the extraction process can lead to big savings in terms of nodes constructed for the final BDD. Without their use during the extraction, we would have to construct the BDD corresponding to $n = 6$. This big BDD of ≈ 400 nodes would have to be refined with respect to the constraints afterwards. There seem to be cases (e.g., for $n = 4$) where the use of integrity constrain yields larger BDDs, but nonetheless, the final BDD does not have to be revised afterwards, and the difference is not too big.

7 Conclusions and Future Work

A novel approach for the extraction of propositional rules from feed-forward networks of threshold units has been presented. After decomposing the network into perceptrons, binary decision diagrams representing preconditions that activate or inactivate the perceptron have been extracted. Those intermediate representations can be composed using the usual algorithms for BDDs, or they can be combined during their construction by extracting one into the other. The latter approach does also allow for an incorporation of integrity constraints – already during the extraction of the intermediate results. As already mentioned in [Bader *et al.*, 2007], the pruned search trees constructed above are related to the approach presented in [Krishnan *et al.*, 1999]. But due to a different order, we do not need to expand them completely, which would otherwise be necessary.

The extraction as presented here is applicable to all feed-forward networks composed of binary threshold units computing ± 1-threshold function. This limitation can be softened by allowing arbitrary symmetric threshold functions. The symmetry is necessary to construct negative and positive forms of the perceptron without changing the global network function.

Finally, we discussed first experimental results indicating a good performance of the approach. On the one hand, we obtain a very compact representation and on the other hand, we circumvent the construction of non-necessary intermediate results while incorporating integrity constraints right from the start.

Nonetheless, much remains to be done. In particular, the extraction for non-threshold units has to be studied. For the encoder-decoder experiments mentioned above the algorithm has simply been applied to networks computing the symmetric hyperbolic tangent as activation function. Interestingly, the result coincide with our expectations. This is due to the fact, that networks when trained to compute crisp decisions tend to behave like threshold networks. But the details of this need to be investigated in the future. Furthermore, a detailed analysis of the performance is necessary, in particular using networks trained for real-world problems. The approach as presented here detects equivalent sub-BDDs for n-of-m patterns. But there are more cases for equivalent sub-BDDs [Mayer-Eichberger, 2008]. Those have to be integrated into the extraction procedure. It would also be interesting to study the evolution of a network during the training process by repeatedly applying the extraction method and compare the results.

Acknowledgements The author is thankful for the comments of Valentin Mayer-Eichberger and two anonymous reviewers.

References

[Andersen, 1999] H. R. Andersen. An introduction to binary decision diagrams. Lecture Notes, 1999.

[Andrews *et al.*, 1995] R. Andrews, J. Diederich, and A. Tickle. A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge–Based Systems*, 8(6), 1995.

[Bader *et al.*, 2007] S. Bader, S. Hölldobler, and V. Mayer-Eichberger. Extracting propositional rules from feed-forward neural networks — a new decompositional approach. In *Proceedings of the 3rd International Workshop on Neural-Symbolic Learning and Reasoning, NeSy'07*, January 2007.

[Jacobsson, 2005] H. Jacobsson. Rule extraction from recurrent neural networks: A taxonomy and review. *Neural Computation*, 17(6):1223–1263, 2005.

[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.

[Mayer-Eichberger, 2008] V. Mayer-Eichberger. Towards solving a system of pseudo boolean constraints with binary decision diagrams. Master's thesis, Universidade Nova de Lisboa, SEP 2008.

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