Extracting Symbolic Knowledge from Artificial Neural Networks

Sebastian B. Thrun
University of Bonn
Dept. of Computer Science III
Römerstr. 164, D-5300 Bonn 1, Germany
E-mail: thrun@uran.cs.bonn.edu

Abstract

Although artificial neural networks have been applied successfully to a variety of real-world scenarios, they have often been criticized for exhibiting a low degree of human comprehensibility. Mechanisms that automatically compile neural networks into symbolic rules offer a promising perspective to overcome this practical shortcoming of neural network representations. While the need for such techniques has long been recognized, we are still lacking general methods that work efficient in practice.

This paper describes an approach to the extraction of symbolic *if-then* rules from artificial neural networks. The key mechanism of this approach is *Validity Interval Analysis* (VI-Analysis). VI-Analysis is a generic tool for extracting symbolic knowledge from Backpropagation-style artificial neural networks. It can make assertions about networks by propagating rule-like knowledge through the network. Rules are extracted by a generate-and-test procedure, using VI-Analysis for checking the consistency with the artificial neural network at hand. Empirical studies include results obtained for the MONK's benchmark problems, and a robot arm kinematics problem.

Keywords. machine learning, artificial neural networks, rule extraction, validity interval analysis, symbolic and subsymbolic representations

1 Introduction

In the last few years artificial neural networks have been applied successfully to problems of learning and generalization in a variety of real-world scenarios. If classification spaces are high-dimensional, if input-output relations are complex, if training data is noisy and attributes are real-valued, neural network approaches have often been reported to yield performance that compares favorably to other methods. For example, artificial neural networks have been successfully applied in the area of speech generation [Sejnowski and Rosenberg, 1986] and recognition [Waibel, 1989], vision and robotics [Pomerleau, 1989], [Hsu and Simmons, 1991], handwritten character recognition [LeCun *et al.*, 1990], medical diagnostics [Jabri *et al.*, 1992] and game playing [Tesauro, 1992]. Some of these approaches clearly outperformed other, more rule-based approaches. While this brief list does by no means cover the rapidly growing area of neural network applications, it nonetheless illustrates the important role which the field of artificial neural networks plays in Artificial Intelligence.

In this paper we will address a major shortcoming of neural network representations. While statistically, neural networks have frequently been reported to achieve high generalization accuracy, the classification concepts of these networks are usually barely comprehensible to human users. This is because artificial neural networks work by spreading activations through groups of densely interconnected processing units. These non-linear units are characterized by large sets of hard-to-interpret real-valued parameters, the so-called *weights* and *biases*. Distributed, non-discrete representations make it even harder to understand *what exactly* a network has learned, and under which circumstances it will fail to generate the correct answer. Indeed, in practice the low degree of comprehensibility has been a severe obstacle in applying artificial neural networks to real-world problems.

On the other hand, most rule-based, symbolic learning systems offer the desired higher degree of comprehensibility due to the sparse nature of symbolic rules. Symbolic learning mechanisms seek to generate small sets of sparse rules that describe the observed training data. Such rules can usually be much better interpreted by human experts and are thus easier to understand. Moreover, symbolic rules facilitate interfacing with other knowledge-based systems, such as expert systems or intelligent databases. If neural networks are the learning method of choice, e.g., if they are found to have the desired generalization properties, mechanisms which compile networks into sets of rules offer a promising perspective to overcome this obvious shortcoming of artificial neural networks.

The importance of rule verification and extraction for artificial neural networks has long been recognized. To date, there is a variety of techniques available which compile symbolic rules out of neural networks. The most popular approaches to rules extraction[1] translate networks one-to-one into rules. Network nodes are interpreted as logical entities whose preconditions are determined by their ingoing weights. Network structure is hence directly mapped into structure of logical inference. Since such a literal interpretation is only accurate, if network activations are binary and weights are approximately discrete, which is not necessarily the case in neural networks, rule extraction has often been studied in combination with appropriate schemes for structuring networks, along with modified training algorithms that enforce the desired network properties. Sparse connectivity facilitates this type rule extraction, and so do binary activation values. While this methodology is intriguing, as it draws a clear one-to-one correspondence between neural inference and rule-based inference, it is not universally applicable to arbitrary Backpropagation-style neural networks. This is because neural networks might not meet the strong representational and structural requirements necessary for these techniques to work successfully. If networks are trained with the standard Backpropagation procedure, they often exhibit rather distributed types of data processing, using real-valued activations in densely interconnected networks that lack topological structure. When the internal representation of the network is distributed in nature, individual hidden units typically do not represent clear, logical entities. One might argue that networks, if one is interested in extracting rules, should be constructed appropriately. But this would outrule most existing network implementations, as such considerations have barely played a role. In addition, such an argument would suppress the development of distributed, non-discrete internal representations, which have often be attributed for the generalization properties of neural networks. It is this more general class of networks that is at stake in this paper. The goal of this research is to extend the current technology of rule extraction to a much broader class of artificial neural networks that exhibit less structure and more complex internal representations.

In this paper we describe an approach to the extraction of rules from artificial neural networks. More specifically, the approach allows to extract *if-then* type rules from Backpropagation-style neural networks. The main engine for compiling networks into rules is Validity Interval Analysis[2] (in short: VI-Analysis). VI-Analysis is a generic tool for checking the consistency of rules with a neural network. It does this by an iterative process of propagating intervals of valid activations through the network. Unlike the approaches summarized above, VI-

[1] which are discussed in more detail in Section 5

[2] VI-Analysis was originally proposed in [Thrun and Linden, 1990] [Thrun, 1993].

Analysis analyzes networks as a whole by examining the relation between input and output of the network. Symbolic rules are extracted by a generate-and-test procedure. Candidate rules are generated by either of two mechanisms, a graph-based scheme used for discrete domains, or a random rule growing scheme that operates in domains with real-valued features. Given a specific set of rules, VI-Analysis is employed for checking its consistency with the neural network at hand. The extracted rules represent provably accurate descriptions of the network. In principle, rule extraction via VI-Analysis does not require discrete activations, local internal representations, special network topologies, or specialized training schemes. It is applicable to a wide variety of artificial Backpropagation-style neural networks. We have extracted rules from a variety of neural networks trained with the standard Backpropagation algorithm, neither of which was specifically constructed to facilitate the extraction of rules.

The remainder of this paper is organized as follows. The basic notion of VI-Analysis is presented in Section 2. Subsequently, in Section 3, we will show how VI-Analysis can be applied to verify rules. Two canonical extraction schemes are presented, one for discrete domains and one for classification domains with real-valued features. Section 4 reviews some empirical results using the XOR problem, the more complex MONK's problems and a robot arm kinematics problem. The paper is concluded by a survey of related approaches to rule extraction (Section 5), and a discussion of the strengths and weaknesses of the approach (Section 6).

2 Validity Interval Analysis

Validity Interval Analysis is a generic tool for analyzing Backpropagation-style artificial neural networks. It allows assertions to be made about the relation of activation values of a network by analyzing its weights and biases. VI-Analysis can be applied to arbitrary, trained networks—no requirements are made on the weights and biases or on the topology of the network at hand. The only assumption we make throughout this paper is that the non-linear transfer functions of the units in the network are monotonic and continuous[3], as is typically the case in Backpropagation networks.

The basic principle of VI-Analysis is based on so-called *validity intervals*. Such intervals

[3]See the discussion at the end of this paper for a relaxation of these constraints to piecewise monotonic and piecewise continuous transfer functions.

constrain the activation patterns[4] in the network at hand. More specifically, a validity interval of a unit specifies a maximum range for its activation value. Initially, the user may assign arbitrary intervals to all (or a subset of all) units. VI-Analysis then refines these intervals. It does this by iteratively detecting and excluding activation values that are provably inconsistent with the weights and the biases of the network. This mechanism, which will be described in turn, is truth-preserving in the sense that each activation pattern which is consistent with the initial set of intervals will also be consistent with the refined, smaller set of intervals.

Starting with an initial set of validity intervals—the "input" of VI-Analysis—, there are two possible outcomes of VI-Analysis:

- The routine *converges*. The resulting intervals form a subspace which includes all activation patterns consistent with the initial intervals.
- A *contradiction*, i.e. an *empty interval*, is found. If an empty interval is generated, meaning that the lower bound of an interval exceeds its upper bound, there will be no activation pattern whatsoever which can satisfy the constraints imposed by the initial validity intervals. Consequently, the initial intervals are *inconsistent* with the weights and biases of the network. This case will play a key role in rule extraction and verification.

In the remainder of this section we will describe the VI-Analysis algorithm. We start by reviewing the forward propagation rule of the Backpropagation algorithm and introduce basic notation. The description of VI-Analysis on a simple example, namely a network realizing the boolean function AND, is followed by a general description of the refinement procedure in VI-Analysis.

2.1 Notation

The following rules of activation propagation for artificial neural networks may be found in the literature on the Backpropagation training algorithm, see e.g. [Rumelhart *et al.*, 1986]. *Activation* values are denoted by x_i, where i refers to the index of the unit in the network. If unit i is an input unit, its activation value will simply be the external input value. If not, i.e.,

[4]By *activation pattern* we mean a vector of activations generated by the standard forward propagation equations given below.

if i refers to a hidden or an output unit, let $P(i)$ denote the set of units that are connected to unit i. The activation x_i is computed in two steps:

$$\begin{aligned} net_i &= \sum_{k \in P(i)} w_{ik} x_k + \theta_i \\ x_i &= \sigma_i(net_i) \end{aligned}$$

Here the auxiliary variable net_i is called *net-input* of unit i, and w_{ik} and θ_i are real-valued parameters called *weights* and *biases* (*thresholds*). These parameters are adapted during Backpropagation learning [Rumelhart *et al.*, 1986] in order to fit a set of training examples. σ_i denotes the transfer function (squashing function), which usually is given by

$$\sigma_i(net_i) \;=\; \frac{1}{1+e^{-net_i}} \quad \text{with} \quad \sigma_i^{-1}(x_i) \;=\; -\ln\left(\frac{1}{x_i} - 1\right)$$

Since VI-Analysis applies to arbitrary trained networks and does not demand further modification of the weights and biases of the network, we will omit the procedure for estimating weights and biases that originally gave Backpropagation its name [Rumelhart *et al.*, 1986]. Rather, we consider static network topologies and static values for the weights and biases, i.e., we assume the network has already been trained successfully. Validity intervals for activation values x_i are denoted by $[a_i; b_i]$. If necessary, validity intervals are projected into the net-input space of a unit i, where they will be denoted by $[a'_i; b'_i]$.

2.2 A Simple Example

Before we explain the procedure of interval refinement in its most general form, let us give some intuitive examples which serve as demonstrations of the main aspects of VI-Analysis. Consider the network shown in Figure 1a. This network approximates the boolean function AND and might be the result of Backpropagation learning. Let us assume we are interested in the classification achieved by the network when there is some small, real-valued noise added to the input values. In other words, we would like to characterize the robustness of the learned classification.

We will now give four examples of VI-analyzing this network, demonstrating the major processing steps and the different cases in VI-Analysis:

1. **Forward phase.** VI-Analysis consists of two alternating phases, a forward and a backward phase. In the forward phase, interval constraints on the activation space

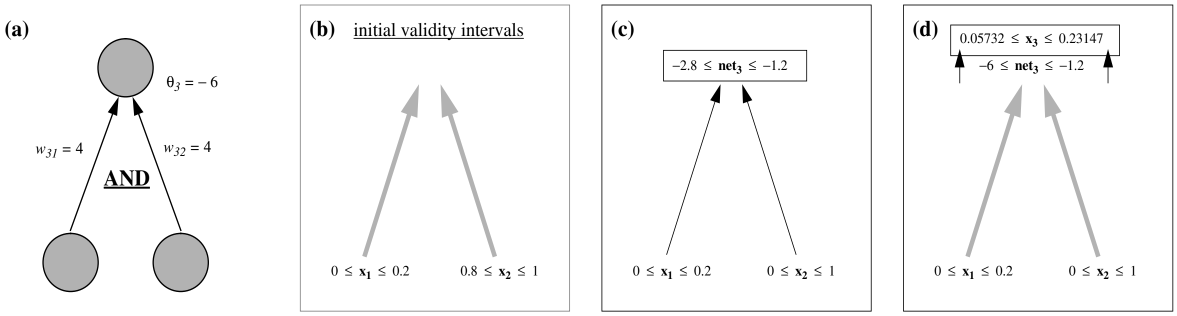


Figure 1: **VI-Analysis, forward phase.** (a) A simple artificial neural network that realizes the boolean function AND. (b) The initial input intervals are chosen to be $x_1 \in [0; 0.2]$ and $x_2 \in [0.8; 1]$. The output activation is not constrained. (c) Taking the weights and bias of the network into account, forward propagation of the input intervals leads to a maximum range for the net-input $net_3 \in [-2.8; -1.2]$. (d) This interval is mapped by the sigmoid transfer function to the final output validity interval: $x_3 \in [0.05732; 0.23147]$. The result of the analysis reads: *If* $x_1 \leq 0.2$ *and* $x_2 \geq 0.8$ *then* $x_3 \in [0.05732; 0.2314]$.

are propagated forward through the network, similar to the forward propagation of activations in a Backpropagation network. In VI-Analysis, whole activation intervals are propagated instead of values. To see how this is done, consider Figure 1b-d. In Figure 1b two initial intervals for the activation values of the input units are specified. The input x_1 is constrained to be in $[a_1; b_1] = [0; 0.2]$, and the input x_2 is in $[a_2; b_2] = [0.8; 1]$. Since the network is trained to realize AND, we expect the output to be smaller than 0.5 for all input vectors that match these interval constraints.

A simple inspection of the weights and biases enable us to propagate interval bounds through the network. The minimum net-input net_3 is governed by $a'_3 = a_1 \cdot w_{31} + a_2 \cdot w_{32} + \theta_3 = 0 \cdot 4 + 0.8 \cdot 4 - 6 = -2.8$. Likewise, the maximum value for net_3 is $b'_3 = -1.2$, leading to the validity interval $[a'_3; b'_3] = [-2.8; -1.2]$ for net_3. Since we assume that all transfer functions σ are continuous, input intervals of the transfer function correspond directly to output intervals. The validity interval $[a_3; b_3]$ for x_3 is obtained by applying the transfer function to $[a'_3; b'_3]$ yielding $\sigma([a'_3; b'_3]) = [\sigma(a'_3); \sigma(b'_3)] = [\sigma(-2.8); \sigma(-1.2)] = [0.05732; 0.2314] = [a_3; b_3]$. Thus, the output activations have to be in $[a_3; b_3] = [0.05732; 0.2314]$. This completes the forward phase.

It should be noted that the result of this simple analysis can already be interpreted as a rule: *If* $x_1 \leq 0.2$ *and* $x_2 \geq 0.8$ *then* $x_3 \in [0.05732; 0.2314]$. This rule implies the

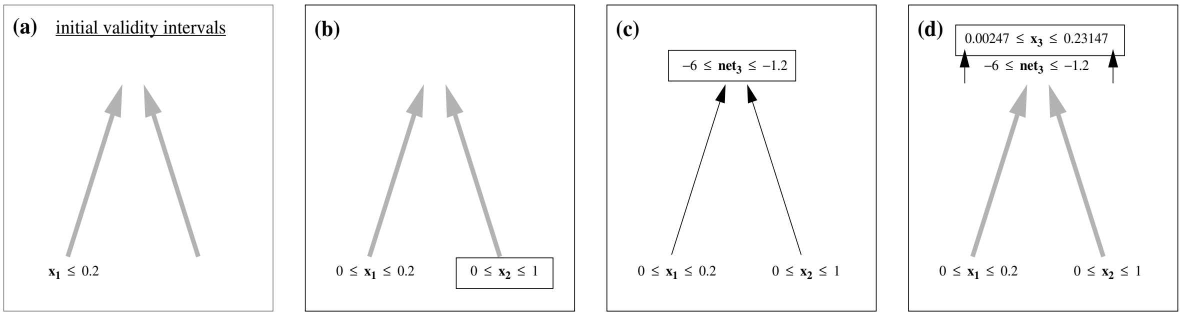


Figure 2: **VI-Analysis with open intervals.** (a) Analogous situation as in Figure 1, but one of the input units is unconstrained. (b) Activations are bounded by $[0; 1]$. (c)-(d) VI-Analysis leads to the rule: *If* $x_1 \leq 0.2$ *then* $x_3 \in [0.002472; 0.2314]$.

weaker rule *If* $x_1 \leq 0.2$ *and* $x_2 \geq 0.8$ *then* $x_3 < 0.5$, which describes the correct generalization of the learned AND function within the initial input intervals.

Notice that in this and the following examples the initial intervals have been specified manually. Later they will be generated automatically by virtue of a rule search mechanism (c.f. Section 3).

2. **Forward phase with unconstrained input values.** Figure 2 demonstrates again the forward propagation phase, but one of the input activations is not constrained by an interval. We assume that activation values are bounded, without loss of generality by $[0; 1]$. This is trivially true for non-input units, since the range of σ is $(0; 1)$. VI-Analysis can now be applied in the same manner, leading to $[a_3; b_3] = [0.002472; 0.2314]$ and the rule: *If* $x_1 \leq 0.2$ *then* $x_3 \in [0.002472; 0.2314]$ (c.f. Figure 2). The canonical interval $[0, 1]$ corresponds to the state of maximum ignorance about the activation of a unit, and hence is the default value if no more specific interval is known.

3. **Backward phase.** VI-Analysis also allows interval constraints to be propagated *backwards* through the network[5]. In Figure 3a two initial intervals are specified, one for the input activation $x_2 \geq 0.8$ and one for the output activation $x_3 \geq 0.8$. Since the network realizes the boolean function AND, it follows intuitively that the input x_1 should be ≥ 0.5.

   Figure 3b-d illustrates, too, the forward propagation phase of VI-Analysis. Note that in Figure 3d only the upper bound b_3 of the output interval is modified—the initial lower

[5] This process it not to be confused with the backward propagation of gradients in the Backpropagation training algorithm.

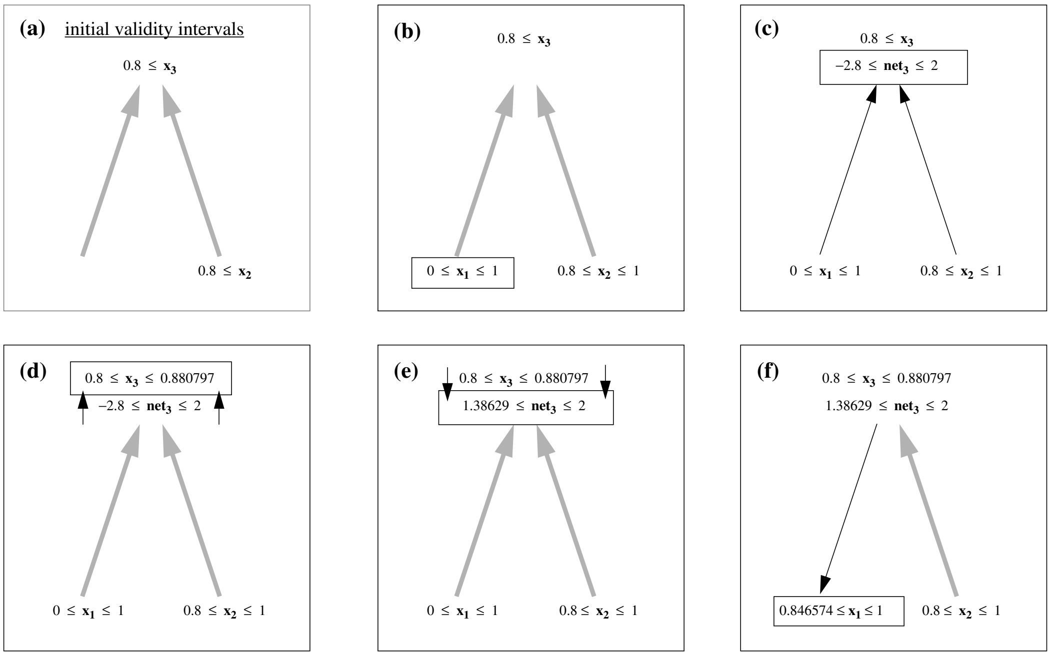


Figure 3: **VI-Analysis, backward phase.** (a) The initial intervals constrain both one of the input activations and the output activation. (b)-(d) Intervals are propagated forward. (e) The validity interval on x_3 is projected back to net_3 via the inverse sigmoid. In particular, $x_3 \geq 0.8$ implies $net_3 \geq 1.386$. (f) A simple calculation shows that for all activation values $x_1 < 0.8465 = (1.386 - \theta_3 + w_{32} \cdot 1.0) w_{31}^{-1}$ it is impossible to find an activation value $x_2 \in [0.8; 1]$ such that $net_3 \geq 1.386$. Therefore, $x_1 \geq 0.8465$. The resulting rule reads: *If* $x_2 \geq 0.8$ *and* $x_3 \geq 0.8$ *then* $x_1 \geq 0.8465$.

---

bound $a_3 = 0.8$ is tighter than the propagated lower bound $\sigma^{-1}(a'_3) = \sigma^{-1}(-2.8) = 0.05732$. Figure 3e-f illustrate the backward phase: First, the output interval $[a_3; b_3]$ is projected into the validity interval $[a'_3; b'_3]$ for net_3. This is done via the inverse transfer function: $\sigma^{-1}([0.8; 0.8807]) = [\sigma^{-1}(0.8); \sigma^{-1}(0.8807)] = [1.386; 2]$. Notice in our example the backward step increases the lower bound for net_3. Second, the validity interval $[a_1; b_1]$ is constrained to values that *will lead to a valid net-input value* net_3 *within* $[a'_3; b'_3]$. Consider for example the lower bound $a'_3 = 1.386$ for the net-input $net_3 = w_{31}x_1 + w_{32}x_2 + \theta_3$. Simple arithmetic shows that $x_1 = \frac{net_3 - w_{32}x_2 - \theta_3}{w_{31}}$ and thus $a_1 = \min x_1 = \frac{a'_3 - w_{32}b_2 - \theta_3}{w_{31}} = \frac{1.386 - 4 \cdot 1 + 6}{4} = 0.8465$. Henceforth, 0.8465 is a lower bound on the activation of x_1. This procedure illustrates the backward phase in VI-Analysis: Activations are constrained by the fact that they *must ensure that all successors* j *may receive a net-input within their validity interval* $[a'_j; b'_j]$*, given that all*

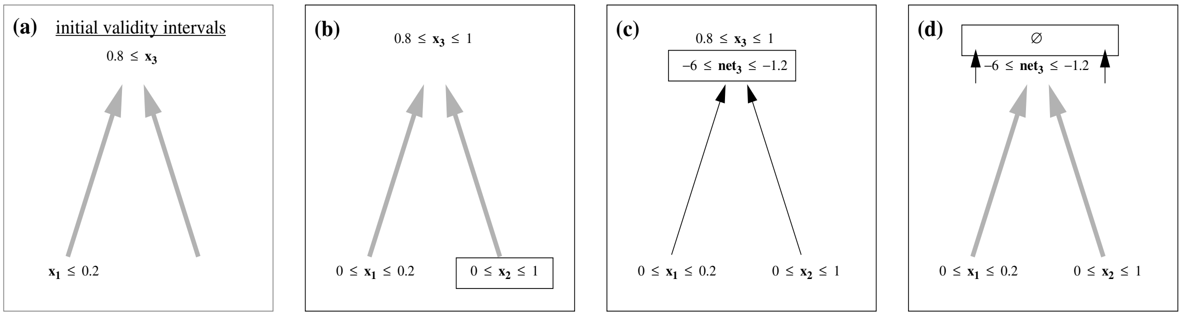


Figure 4: **VI-Analysis and inconsistencies.** (a) As in the Figure 3, there are two initial intervals assigned to one of the input units and the output unit. Since high output requires both inputs to be high, this set of intervals is inconsistent with the AND function. (b)-(d) VI-Analysis generates an empty output interval. More specifically, $x_3 \geq 0.8$ violates $x_3 = \sigma(net_3) \leq \sigma(-1.2) = 0.2314$. The empty interval proves the inconsistency of the initial constraints.

other units k connected to these successors contribute appropriate activation values within their intervals $[a_k; b_k]$. Activation ranges excluded by this step cannot occur.

Notice that if transfer functions are monotonic and continuous, the projections of compact intervals are, too, compact intervals. This is the reason why we consider exclusively monotonic and continuous transfer functions in this paper.

Although the backward phase in VI-Analysis looks considerably more complex than the forward phase, both phases are strongly related, as they are realized by the same mechanism in the general VIA algorithm presented below.

4. **Inconsistencies.** Finally, we will demonstrate how VI-Analysis can detect inconsistencies in the initial set of intervals. Figure 4 shows a situation in which two intervals $[a_1; b_1] = [0; 0.2]$ and $[a_3; b_3] = [0.8; 1]$ are specified, very similar to the situation shown in Figure 3. Now, however, the initial setting is inconsistent with AND, because low input x_1 implies low output x_3. Consequently, the forward propagation phase shown in Figure 4b-d, which is analogous to that in Figure 2, yields an empty set for the output values x_3. This is because the intersection of the intervals $[a_3, b_3] = [0.8; 1]$ and $\sigma([a'_3, b'_3]) = \sigma([-6; -1.2]) = [0.00247; 0.2314]$ is $\emptyset$. The logical conclusion of the occurrence of an empty set is the inconsistency of the initial intervals, since there is no activation pattern for this network which can satisfy all initial interval constraints.

This completes the description of the AND example. Although the example is simple, the main ideas of VI-Analysis have been demonstrated, namely:

- Constraints are specified by initial intervals on the activation patterns. These intervals can, for example, be specified by the user. In the context of rule extraction, as described below, they are automatically generated by some rule search mechanism. Initial intervals form the "input" of VI-Analysis.
- Constraints are propagated in both directions through the network by refining intervals. All refinements made during this process preserve the consistent activation space, i.e., VIA is truth-preserving.
- VI-Analysis converges to a refined set of intervals, and might also detect inconsistencies in the initial intervals.

In the next section we will describe the general VIA algorithm for arbitrary networks. This algorithm takes hidden units into account, and the notion of intervals is extended to the notion of linear constraints on activations. The reader will notice, however, that the principles are the same as those described above.

2.3 The General VIA Algorithm

Assume without loss of generality that the network is layered and fully connected between two adjacent layers. This assumption simplifies the description of VI-Analysis, although VI-Analysis can also be applied to arbitrary non-layered, partially connected network architectures, as well as recurrent networks not examined here. Consider a single weight layer, connecting a layer of preceding units, denoted by $\mathcal{P}$, to a layer of succeeding units, denoted by $\mathcal{S}$. Such a scenario is depicted in Figure 5. As we will see, the problem of refining interval bounds can be attacked by techniques of linear programming, such as the Simplex algorithm[6]. Assume there are intervals $[a_i, b_i] \in [0, 1]^2$ assigned to each unit in $\mathcal{P}$ and $\mathcal{S}$. The canonical interval $[0, 1]$ corresponds to the state of maximum ignorance about the activation of a unit, and hence is the default value if no more specific interval is known. In order to make linear programming techniques applicable, the non-linearities due to the transfer function of units in $\mathcal{S}$ must be eliminated. As in the AND example, this is done by projecting $[a_i; b_i]$ back to the corresponding net-input intervals $[a_i'; b_i'] = \sigma^{-1}([a_i; b_i]) \in \bar{\Re}^2$.[7] The resulting validity

[6]The particular implementation used for our experiments was taken from [Press, 1988]

[7]Here $\bar{\Re}$ denotes the set of real numbers extended by $\pm\infty$.

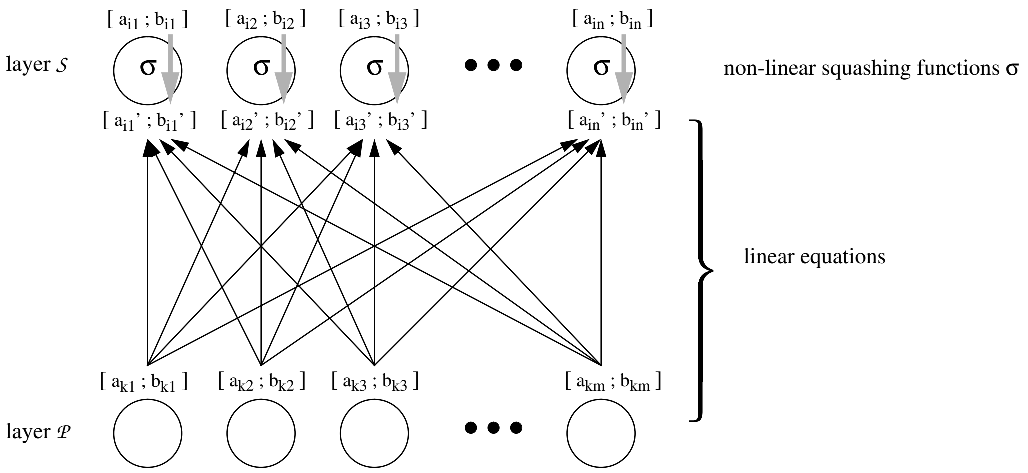


Figure 5: **Weight layer.** The set of units in layer $\mathcal{P}$ are connected to the unit in layer $\mathcal{S}$. A validity interval $[a_j; b_j]$ is assigned to each unit j ($j \in \mathcal{P} \cup \mathcal{S}$). By projecting the validity intervals for all $i \in \mathcal{S}$, intervals $[a_i'; b_i']$ for the net-inputs net_i are created. These, plus the validity intervals for all units $k \in \mathcal{P}$, form a set of linear constraints on the activations x_k in layer $\mathcal{P}$. Linear programming is now employed to refine all interval bounds.

intervals in $\mathcal{P}$ and $\mathcal{S}$ form a set of linear constraints on the activation values in $\mathcal{P}$:

$$\begin{array}{lrcll}
\forall k \in \mathcal{P}: & x_k & \geq & a_k & \\
 & x_k & \leq & b_k & \\
\forall i \in \mathcal{S}: & \sum\limits_{k \in \mathcal{P}} w_{ik} x_k + \theta_i & \geq & a_i' & \left[\text{by substituting } net_i = \sum\limits_{k \in \mathcal{P}} w_{ik} x_k + \theta_i\right] \\
 & \sum\limits_{k \in \mathcal{P}} w_{ik} x_k + \theta_i & \leq & b_i' & \left[\text{by substituting } net_i = \sum\limits_{k \in \mathcal{P}} w_{ik} x_k + \theta_i\right]
\end{array}$$

Notice that all these constraints are linear in the activation values x_k. Linear programming allows to maximize or minimize arbitrary linear combinations of the variables x_j while not violating a set of linear constraints. Hence, linear programming can be applied to refine lower and upper bounds for validity intervals. As in the AND example, constraints are propagated in two phases—notice the strong correspondence to the algorithm described in the last section and this more general scheme.

1. **Forward phase.** In order to refine the bounds a_i and b_i of a unit $i \in \mathcal{S}$, linear

programming is applied to derive new bounds, denoted by $\hat{a}_i$ and $\hat{b}_i$:

$$\hat{a}_i = \sigma(\hat{a}'_i) \quad \text{with} \quad \hat{a}'_i = \min net_i = \min \sum_{k \in \mathcal{P}} w_{ik} x_k + \theta_i$$

$$\hat{b}_i = \sigma(\hat{b}'_i) \quad \text{with} \quad \hat{b}'_i = \max net_i = \max \sum_{k \in \mathcal{P}} w_{ik} x_k + \theta_i$$

Notice that it is the min/max operator that is computed via linear programming. If $\hat{a}_i > a_i$, a tighter lower bound is found and a_i is updated by $\hat{a}_i$. Likewise, b_i is updated by $\hat{b}_i$ if $\hat{b}_i < b_i$.

2. **Backward phase.** In the backward phase the bounds a_k and b_k of all units $k \in \mathcal{P}$ are refined. This is done by using linear programming, too.

$$\hat{a}_k = \min x_k$$
$$\hat{b}_k = \max x_k$$

As in the forward phase, a_k is updated by $\hat{a}_k$ if $\hat{a}_k > a_k$, and b_k is updated by $\hat{b}_k$ if $\hat{b}_k < b_k$. Note that by definition the update rule in both phases ensures that intervals are changed monotonically, since they can only shrink or stay the same. This implies the convergence of VI-Analysis.

In our experiments, we used a plain implementation of the Simplex algorithm adopted from [Press, 1988] for the task of linear programming. Although this algorithm is known to take worst-case exponential time in the number of variables (i.e. units in a layer), we did not yet observe this algorithm to be too slow in practice. It should be noted that there exist more complex algorithms for linear programming which are worst-case polynomial [Karmarkar, 1984]. In [Thrun and Linden, 1990], a related refinement algorithm for VI-Analysis is described that works much faster, but is not as powerful.

Thus far, we have described the refinement procedure for the units connected to a single weight layer. VI-Analysis iteratively refines all intervals in the network. As in the initial AND example, both the forward and the backward procedure are iterated for all units in the network, either until an empty interval is generated (i.e., an inconsistency is found) or the algorithm converges. By refining the intervals of all units, constraints are propagated in both directions even through multiple hidden layers. If several weight layers are involved, however, VI-Analysis may fail in identifying all inconsistent activation values. This is because each weight layer is refined independently of other weights in the network. More specifically, when optimizing unit intervals connected to a single weight layer, the activations of the preceding units $i \in \mathcal{P}$ are considered to be independent. This worst-case analysis ignores dependencies on the activations in $\mathcal{P}$ which may arise from preceding weights and activations beyond the scope of the linear analysis. This results in an overly careful refinement of validity intervals. In the experiments described in this paper we found VI-Analysis to be powerful enough to successfully analyze all networks we applied it to. There are, however, some clearly suboptimal results due to the conservative nature of the refinement in VI-Analysis. They are discussed in Section 4.

3 Rule Extraction

How can VI-Analysis be applied to extract rules from backpropagation-style networks? VI-Analysis is a generic tool for proving or disproving relations between initial intervals assigned to the input and output activations of a network. In the context of rule extraction, this mechanism is employed to verify the consistency of rule conjectures with a neural network. In this section we will describe this verification process in more detail and give heuristic schemes to search the space of rules.

3.1 Rule Verification

The rules considered in this paper are propositional *if-then* rules. The precondition (preimage) of a rule can be a set of arbitrary linear conditions on the input features. For example, the precondition of a rule could be expressed as linear constraints on the values of the input feature. Likewise, the output of the rule can be a set of arbitrary linear conditions on the output activations. For the sake of simplicity we will consider here only rules where the precondition is given by a set of intervals on the individual feature values, and the output is a single target category. Rules of this type can be written as:

<u>If</u> input $\in$ *some hypercube* $\mathcal{I}$ *<u>then</u> class is* C

<u>If</u> input $\in$ *some hypercube* $\mathcal{I}$ *<u>then</u> class is not* C

for some target class C. Here $\mathcal{I}$ abbreviates the precondition-hypercube $\mathcal{I} = [a_i, b_i]^m$ of the rule to be verified.

In this section we will demonstrate how VI-Analysis can be used to prove conjectures like these. For simplicity, assume the output of the network is coded *locally*, i.e., there are c classes, and each output unit j ($j = 1, \ldots, c$) corresponds to one of these classes. Input patterns are classified by propagating activations through the network and then applying the winner-take-all rule to the output activations. In other words, the input is classified as belonging to class C_j ($j = 1, \ldots, c$) if and only if

$$\forall j' \neq j : \qquad x'_j \;<\; x_j$$

Rule conjectures are verified by contradictions. More specifically, let $via_{\mathcal{I}}[x_j < x'_j]$ denote the truth value of running VI-Analysis when the (linear) output constraint $x_j < x'_j$ is added to the initial interval constraints $\mathcal{I} = [a_i, b_i]^m$. We say that $via_{\mathcal{I}}[x_j < x'_j] = false$, if a contradiction is found by the VI-Analysis. If this is the case, the initial input intervals are inconsistent with the conjecture $x_j < x'_j$, and thus $x_j \geq x'_j$. Likewise, $via_{\mathcal{I}}[x_j < x'_j] = true$, if VI-Analysis converges without a contradiction. Then, nothing can be said about the relation of x_j and x'_j under the constraints imposed by the initial intervals $\mathcal{I}$.

The operator $via_{\mathcal{I}}[\cdot]$ can be applied to rule verification. Assume the target class C_j is represented by unit j for some $j \in \{1, \ldots, c\}$. Positive rules

<u>*If*</u> *input* $\in$ *some hypercube* $\mathcal{I} = [a_i, b_i]^m$ <u>*then*</u> *class is* C_j.

is provably correct if

$$\forall j' = 1, \ldots, c \;\text{ with }\; j' \neq j : \qquad via_{\mathcal{I}}[x_j < x'_j] \;=\; false$$

This is to say, proving a positive rule about class membership is the same as proving the non-membership of all other $c - 1$ classes. If VI-Analysis rejects for all $j' \neq j$ the hypothesis that the output of unit j will be smaller than the output of unit j', output unit j will consequently always take the largest activation value among all output units, which proves the conjectured rule. Note that proving this rule requires $c - 1$ iterations of the complete VI-Analysis procedure.

Negated rules can be proven via a single VI-Analysis. The rule

<u>*If*</u> *input* $\in$ *some hypercube* $\mathcal{I} = [a_i, b_i]^m$ <u>*then*</u> *class is* <u>*not*</u> C_j.

can be verified by showing that

$$via_{\mathcal{I}}[x_1 \leq x_j, x_2 \leq x_j, \ldots, x_{j-1} \leq x_j, x_{j+1} \leq x_j, \ldots, x_c \leq x_j] \;=\; false$$

If VI-Analysis finds the desired contradiction, then for each input pattern within the input intervals there exist some $j' \neq j$ with $x_{j'} > x_j$. Thus, the input is never in class C_j, which is exactly what one wants to show here.

Notice that VI-Analysis can also be applied to networks with a single output unit used to distinguish two classes only. In this case, positive rules can be verified by showing that $via_{\mathcal{I}}[x_{\text{output}} < 0.5] = false$, and negative rules can be verified by $via_{\mathcal{I}}[0.5 \leq x_{\text{output}}] = false$.

To summarize, VI-Analysis provides a generic tool for verifying the correctness of conjectured rules. It does this by (a) negating the rule, and (b) detecting logical contradictions in the negation. Such a contradiction is a proof that the opposite of the rule never holds true, which implies the correctness of the original rule. VI-Analysis is truth-preserving. Hence proofs of this kind are correct and not only approximations. It may happen, however, that VI-Analysis is not able to prove the correctness of the rule at hand. In general, there are two reasons why VI-Analysis may fail to do so: Either the rule does not correctly describe the classification of the network (and hence is wrong), or VI-Analysis is just not able to detect a contradiction. As stated in the previous section, VI-Analysis is not guaranteed to find contradictions due to the independent optimization of different weight layers. Not finding a contradiction does, of course, not imply that the opposite holds true.

3.2 Rule Extraction in Discrete Domains

VI-Analysis serves as the basic “theorem prover” for the extraction of symbolic rules from artificial neural networks. Given a procedure for generating hypotheses about a network’s classification, one can employ VI-Analysis to prove or disprove these hypotheses. Of course, there is a variety of themes for generating sets of candidate rules. One could, for example, simply enumerate all possible rules, if the domain of the network is discrete, or guess rules at random. We will in turn describe two canonical techniques for searching the space of rules, one for discrete domains, and one for domains with continuous-valued features. At least in the latter case, it will be impossible to generate rules by exhaustively testing all instances, since there are infinitely many. But even in the discrete case, exhaustive testing can be intractable if feature spaces are large. For both techniques, it is assumed that the network is used for a classification task.

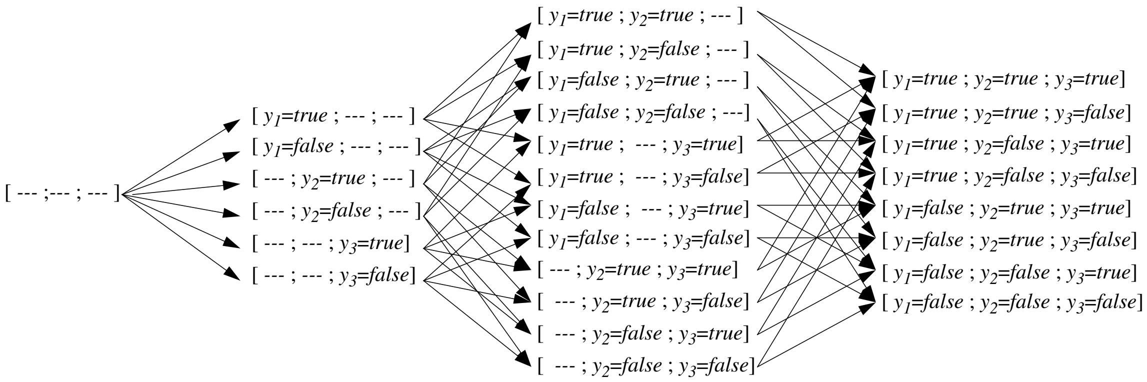


Figure 6: **Example rule-DAG.** A directed acyclic rule graph for rules over three boolean attributes y_1, y_2, and y_3 is shown. The graph is ordered from most general (left) to most specific (right) rules. Rule hypotheses (nodes) are generated and tested breadth-first. Once VI-Analysis succeeds in proving or disproving the class membership for a general rule, the whole subgraph spanned by this rule is removed from the search list.

3.3 Discrete Rules and Graph Search

If the domain of the networks is *finite*[8], i.e., if there are finitely many instances that exhaustively characterize the target function, there is at a first glance a simple but expensive strategy to extract rules from neural networks: Classify all input instances and apply boolean algebra to simplify the logical expression resulting from the network queries (e.g. by transforming this expression into disjunct normal form). Common examples for finite input domains are classification tasks over binary feature vectors. While this straightforward, exhaustive rule extraction technique will always generate a set of rules which correctly describes the function represented by the network, the process of testing each input pattern in the domain of the network is expensive and will be computationally intractable for larger domains.

An alternative way to extract rules is to apply VI-Analysis. VI-Analysis allows to cut whole regions in the search space by checking more general rules in the first place. To see this, consider the space of possible rules. This space can be described by c directed acyclic graphs (DAGs), one for each individual class C, in which nodes correspond to rules. These nodes are ordered from the most general to the most specific. The root of the graph forms the most general rule and is of the type *"everything is in class C"* or *"everything is not in class C."*

[8] In the literature on machine learning, finite domains are often referred to as *discrete domains*. We will use these terms interchangeably.

The leaves correspond to single instances in the input space of the network. Figure 6 shows such a rule graph. Each rule in the graph spans a subgraph of more specific rules, which are logically implied by this rule. VI-Analysis is now applied in breadth-first manner: As soon as VI-Analysis proves the correctness of a rule, the whole subgraph spanned by this rule node belongs to the same class and needs no further investigation. Unlike breadth-first search, the search is then continued in other branches of the DAG, until the whole domain is covered by rules. Searching from most general to most specific may decrease the number of tests significantly, as whole chunks of rules might be pruned away by proving a single rule. This reduction of the search space, however, comes at the expense of running VI-Analysis instead of single forward propagations, as would be required for the pure classification of an instance. How efficient this scheme is in practice, however, depends on the function modeled by the neural network. If simple rules cover large parts of the input space, VI-Analysis will be fast. This is, for example, the case in the MONK's problems reported below. If the input space is highly cluttered, however, rules found by VI-Analysis will be rather specific in nature, and the extraction of the rules will be slow.

3.4 Continuous-Valued Attributes

If the values of the features are continuous, as is the case in many real-world classification domains, the task of extracting rules becomes harder. In such domains it is impossible to exhaustively test all training instances, since by definition there are infinitely many. Indeed, it might take infinitely many rules to describe the network completely, since the decision boundaries between classes in multilayer networks is generally not linear and thus can only be approximated by infinitely many hypercubes. It are domains with continuous-valued attributes where VI-Analysis has a clear, principal advantage over simple generate-and-test schemes, since it can be applied to whole sub-regions in the classification space that are bounded by real values.

In this section we will describe an approach to rule extraction that generalizes single instances in the input space. More specifically, we will employ VI-Analysis to form general rules by iteratively growing the preimage of a rule. Assume a data point is given, which is classified by the network as belonging to some class C. This single data point forms already a rule, namely precisely the rule that this very input pattern will be classified as C. VI-Analysis, when the input intervals are constrained to this single data point, will easily prove the correctness of this rule. In fact, VI-Analysis degrades to the standard forward propagation of activation values if the input hypercube is a single point only, and the "proof" in this case is trivial.

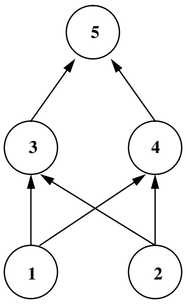
Now assume that the neighborhood of this training instance will be classified as category C as well. Then, a more general rule can be found that also covers neighboring points. The most straightforward scheme to find increasingly more general rules is to enlarge their input regions by small amounts, and to check the correctness of these new, more general rules. More precisely, starting with the current set of intervals, a randomly chosen input interval is extended to a more general interval by adding a small random value to one of the bounds of this interval. Each time an input interval is enlarged by some random amount, VI-Analysis is applied to verify if this more general rule is still consistent with the network. If this is the case, the larger input interval is maintained, otherwise it will be set to its previous values. This iterative growing procedure allows to generate increasingly more general rules, based on a single input pattern which acts as a seed. As the process of enlarging input intervals progresses, more and more general rules are formed until the intervals reach the boundaries of what can be shown to be correct by VI-Analysis. It should be noted that if the network transfer functions are continuous, there almost-certainly (i.e., with probability 1) exist some small, non-point rules that can be extracted by VI-Analysis.

Growing training data results in *legitimate preconditions* that ensure the same classification as the classification obtained for the training instance at hand. The resulting preimage generalizes the training instance in the input space. VI-Analysis bears close resemblance to generalization in symbolic explanation-based learning (EBL) techniques [DeJong and Mooney, 1986], [Mitchell *et al.*, 1986]. In symbolic EBL, *weakest preconditions* are extracted by observing and analyzing a chain of symbolic rule inferences. These weakest preconditions generalize single training instances in the input space. Unlike artificial neural networks, symbolic rules facilitate the extraction of such preimages, since they are easy to invert. Growing legitimate preimages by means of VI-Analysis can thus be viewed as the neural network analogue to the weakest preconditions operator in symbolic EBL.

4 Experimental Results

VI-Analysis was applied to extract rules in various discrete and real-valued domains. We will describe three domains of varying complexity, elucidating the extraction of rules through VI-Analysis in practice.

(a)



(b)

	to-node		
from-node	3 (hidden)	4 (hidden)	5 (output)
1 (input)	-4.60248	4.74295	
2 (input)	-3.19378	2.90011	
bias	2.74108	-1.49695	
3 (hidden)			-4.57199
4 (hidden)			4.64925
bias			2.10176

Figure 7: **Learned XOR.** (a) Network topology and (b) weights and biases.

4.1 Results Obtained for the XOR

In an initial experiment we applied VI-Analysis to a two-layer neural network which was trained to approximate the boolean function XOR. This function has been picked as a simple example that allows to illustrate how intervals are propagated in VI-Analysis. The network, including its weights and biases, is depicted in Figure 7. It was trained using the standard Backpropagation algorithm.

Five runs of VI-Analysis with different initial conditions were performed, both to evaluate the classification of the network and to demonstrate VI-Analysis. Figure 8 displays the results. Each row summarizes a single experiment, and each diagram shows the refinement of a validity interval over time. The five columns correspond to the five units in the network. The starting intervals in each of the five experiments were specified by hand.

1. In the first experiment one input activation was constrained to be small, while the other input activation had to be large. The resulting output interval indicates the intuitively correct classification of the network in this interval: The lower bound of the output is larger than 0.5, if the first input value is small and the second input value is large. It should be noted that a tighter upper bound on the output activation is found by this analysis as well.

2. The second experiment is basically the same as the first, except that in this case both input values are constrained to be small. Consequently, VI-Analysis generates a low output interval.

3. In this experiment the output activation was constrained, replacing one of the input

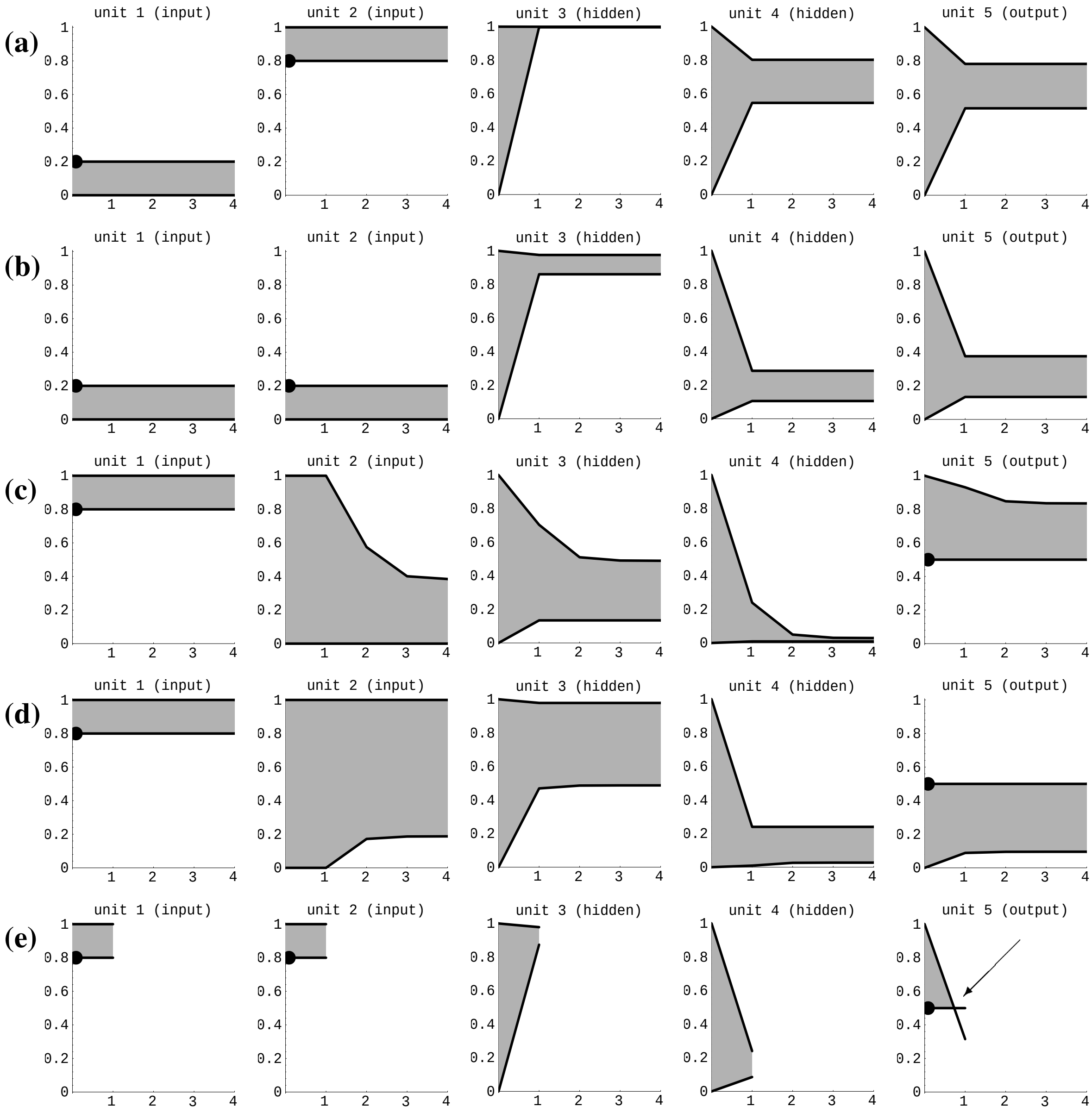


Figure 8: **VI-Analysis on XOR.** Each row summarizes one experiment. The gray area indicates the development of validity intervals over the iterations of VI-refinement for each of the 5 units (columns) in the XOR network. Dots indicate the initial setting of the intervals. (a) Forward propagation: If one input is low and the other is high, the output will be high. (b) Similar situation for two low inputs. (c) Backward propagation of constraints: If one input is high and the output is constrained to be high as well, the other input can be shown to be low. (d) Similar situation, but with low output. In this case there are low inputs for input unit # 2 that will cause low output, even though input #1 is high. This results demonstrates the generalization of the originally discrete XOR problem in the input space. (e) Inconsistency: Both input are forced to be high, but the output is forced to be low. After 1 iteration VI-Analysis detects a contradiction (arrow), since the lower bound of the output unit exceeds the upper bound. This proves that the initial intervals are inconsistent.

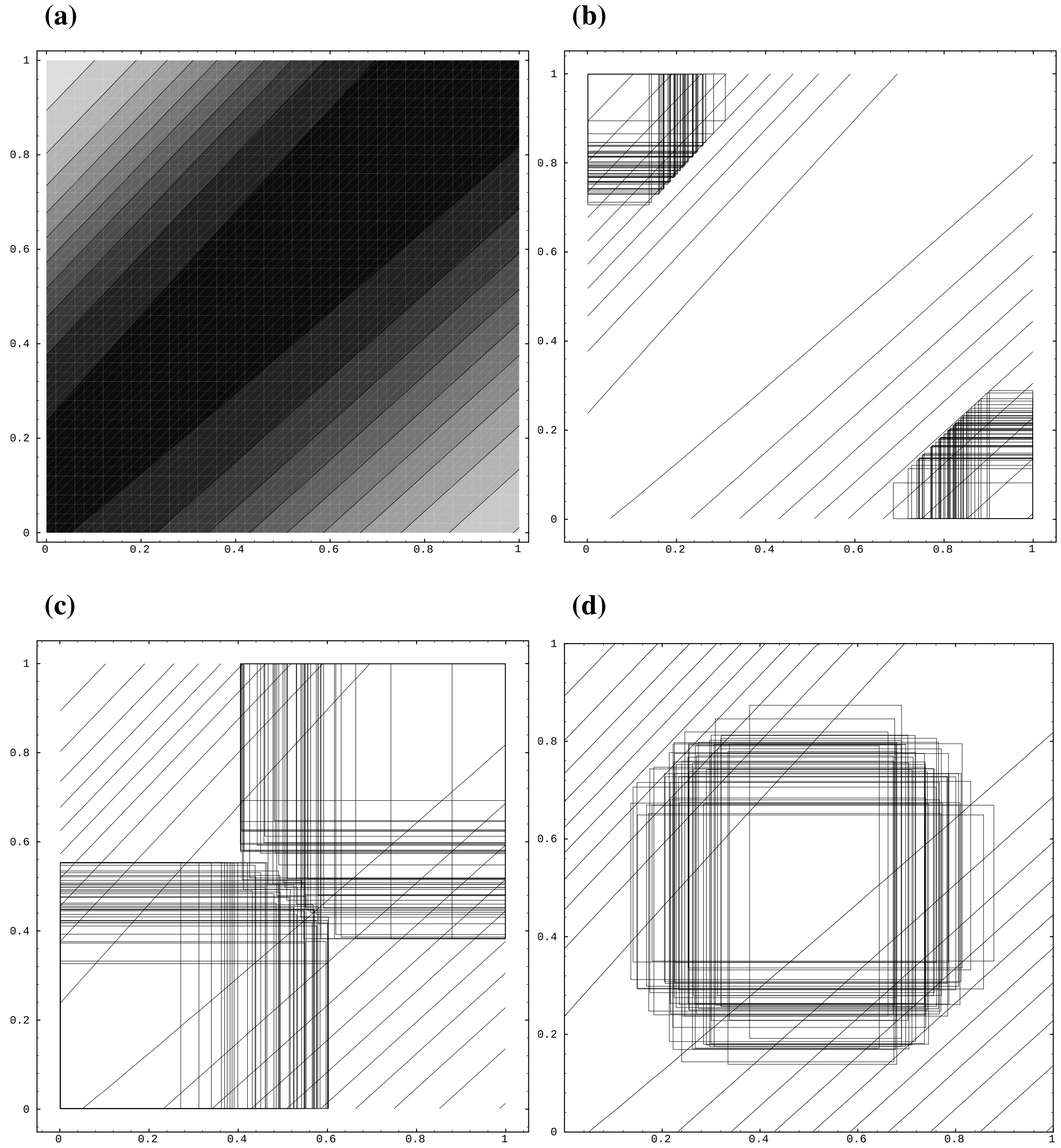


Figure 9: **Rules for XOR.** (a) shows the function generated by the network. The vertical and the horizontal axis measure the two input variables, and the gray-scale measures the network output (white=1, black=0). (b) Several rules found from growing the two patterns $(0, 0)$ and $(1, 1)$. For these hypercubes the output of the network will always be greater than 0.5. Note that the diagonal lines are taken from the diagram (a). (c) Same for the starting points $(0, 1)$ and $(1, 0)$ and (d) $(0.5, 0.5)$. Here the output is provably smaller than 0.5.

activation constraints. It can be seen that the output constraint is propagated backwards through the network, resulting in a small input interval for the initially unconstrained input unit. This result proves the correct generalization within the initial constraints: If the first value is large, and the output value is supposed to be large as well, the other input value must be small.

4. Similar situation, but this time the output values are constrained to be small. The result of this experiment differs from the previous result in that VI-Analysis does not constrain the second input to be larger than 0.5, which is what one would naturally expect. Indeed, closer examination of the network at hand demonstrates that such a rule would be incorrect, since there exist low input values for the second input unit which satisfy all initial constraints.

5. In the last experiments we imposed contradicting initial intervals on the activation values of the network. VI-Analysis thus generates an empty interval, which indicates the inconsistency of the initial intervals.

In order to investigate rule extraction in real-valued domains, one could imagine that real-valued noise corrupted the input values to the network. Input vectors might then be anywhere in the hypercube $[0, 1]^2$, and rules might cover sub-regions thereof. A Natural question to ask is: How will the network perform if the input is corrupted, i.e., how much noise will the network tolerate?

Figure 9 summarizes results obtained with the real-valued rule extraction technique described above using the same XOR network. Figure 9a displays the output of the XOR network over the two-dimensional hypercube spanned by the input units. Using the points $(1; 0)$, $(0; 1)$, $(0; 0)$, $(1; 1)$, and $(.5; .5)$ as a seed, the random rule growing technique was employed to generalize those in the input space. Figure 9b-d displays some resulting rules. Since rules are grown randomly, different runs of the rule growing algorithm result in different rules. Each box in Figure 9b-d corresponds to one of these rules. It can be seen that these rules, put together, cover a large fraction of the input space.

4.2 The MONK's Problems

A set of more complex, discrete classification problems are the so-called MONK's problems. The MONK's problems are called so since they were created for a comparison of learning techniques during a school that was held at a priory in Belgium in 1991 [Thrun *et al.*, 1991]. Since, they have frequently been used as benchmark problems for inductive machine learning algorithms. All three MONK's problems are discrete classification problems defined in an artificial "robot" domain adopted from [Wnek *et al.*, 1990], in which robots are described by six different attributes:

x_1: head_shape	$\in$	{round, square, octagon}
x_2: body_shape	$\in$	{round, square, octagon}
x_3: is_smiling	$\in$	{yes, no}
x_4: holding	$\in$	{sword, balloon, flag}
x_5: jacket_color	$\in$	{red, yellow, green, blue}
x_6: has_tie	$\in$	{yes, no}

The target concepts of the problems are:

- **Problem $\mathbf{M}_1$:** <u>if</u> head_shape = body_shape <u>or</u> jacket_color = red <u>then</u> $\mathbf{M}_1$.

  Figure 10 shows the resulting classification and the weights and biases of the trained network.

- **Problem $\mathbf{M}_2$:** <u>if</u> exactly two of the six attributes have their *first* value <u>then</u> $\mathbf{M}_2$.

  Figure 11 shows the resulting classification and the weights and biases of the trained network.

- **Problem $\mathbf{M}_3$:** <u>if</u> (jacket_color = green <u>and</u> holding = sword) <u>or</u> (jacket_color = <u>not</u> blue <u>and</u> body_shape = <u>not</u> octagon) <u>then</u> $\mathbf{M}_3$.

  Figure 12 shows the resulting classification and the weights and biases of the trained network. Note that in this problem the training set contained noisy training instances which accounts for the failure of Backpropagation to achieve 100% classification accuracy. The most successful network for $\mathbf{M}_3$ was trained with a weight decay term.

In Chapter 9 of the MONK's Report, Backpropagation solutions to the MONK's problems are presented (c.f. Figures 10, 11, and 12). The networks solving these problems had 17 input units, each corresponding to one of the 17 input feature values, and one hidden layer with 2 or 3 hidden units. Although the classification rate achieved by the Backpropagation algorithm compares favorably to other approaches presented in this study, it has been argued that Backpropagation is limited in that it does not allow for a precise interpretation of the results, unlike symbolic learning techniques such as decision trees. We applied VI-Analysis and DAG rule extraction to these networks. Note that these networks were not trained with the intention of easing their analysis—rather they established ad hoc solutions to the MONK's classification problems.

An important observation related to the particular 1-of-n coding of input features turned out to be essential for increasing the rule verification power of VI-Analysis in the context of the MONK's problems: The input coding of all networks was local, i.e., to each feature value we assigned a separate unit which was set to 1, if the corresponding feature was present, and 0 otherwise. Since different values of one and the same feature are mutually exclusive, exactly one (out of 2, 3 or 4, depending on the particular feature) input unit may be on, while all other units belonging to this feature must be off. A slightly weaker form of this constraint can be easily expressed in the language of linear programming: For each of the six features we constrained the input activations such that the sum of all activation values was exactly 1. This results in six new linear constraints on the input patterns, one for each of the six features describing the instances. These additional constraints were simply appended to the standard interval constraints $\mathcal{I}$. Without these additional input constraints VI-Analysis was unable to find reasonable rules at all. With these constraints, however, VI-Analysis managed to find small sets of rules that correctly described the categories as represented by the networks. The results of VI-Analysis for each of the three MONK's problems, respectively, are summarized in turn.

- **$\mathbf{M}_1$.** (c.f. Figure 10) The most general rules found by VI-Analysis are shown in Table 1. As can be seen from this table, VI-Analysis found a number of quite general rules for the target concept. It failed, however, in detecting all most general rules possible. For example, the attribute body_shape may only take the values round, square, or octagon, and thus the first three rules depicted in Table 1 can be combined yielding

  <u>if</u> jacket_color = red <u>then</u> $\mathbf{M}_1$.

  This rule also covers the fourth, sixth and eighth rules listed. Further logical simplification leads to the desired classification:

  <u>if</u> head_shape = body_shape <u>or</u> jacket_color = red <u>then</u> $\mathbf{M}_1$.

- **$\mathbf{M}_2$.** (c.f. Figure 11) As might be seen from the concept description of the problem $\mathbf{M}_2$, there is no compact, simple solution in terms of hypercube-type rules. This is

holding	sword								flag								balloon										
jacket_color	red		yellow		green		blue		red		yellow		green		blue		red		yellow		green		blue				
has_tie	y	n	y	n	y	n	y	n	y	n	y	n	y	n	y	n	y	n	y	n	y	n	y	n	is smiling	body shape	head shape
	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	y	rou	rou
	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	n	rou	rou
	#	#							#	#							#	#							y	squ	rou
	#	#							#	#							#	#							n	squ	rou
	#	#							#	#							#	#							y	oct	rou
	#	#							#	#							#	#							n	oct	rou
	#	#							#	#							#	#							y	rou	squ
	#	#							#	#							#	#							n	rou	squ
	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	y	squ	squ
	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	n	squ	squ
	#	#							#	#							#	#							y	oct	squ
	#	#							#	#							#	#							n	oct	squ
	#	#							#	#							#	#							y	rou	oct
	#	#							#	#							#	#							n	rou	oct
	#	#							#	#							#	#							y	squ	oct
	#	#							#	#							#	#							n	squ	oct
	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	y	oct	oct
	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	n	oct	oct

MONK's problem $\mathbf{M}_1$: weights and biases

	to-node			
from-node	hidden_1	hidden_2	hidden_3	output
input_1 (head_shape round)	-6.503145	0.618412	-1.660409	
input_2 (head_shape square)	1.210703	1.939613	2.972592	
input_3 (head_shape octagon)	5.356444	-3.597301	-1.266992	
input_4 (body_shape round)	-6.692434	2.129635	-2.032242	
input_5 (body_shape square)	6.457639	0.864312	4.260765	
input_6 (body_shape octagon)	0.225053	-2.428098	-1.839603	
input_7 (is_smiling yes)	0.096995	0.131133	0.053480	
input_8 (is_smiling no)	-0.011828	0.135277	0.107302	
input_9 (holding sword)	-0.076848	0.459903	-0.008368	
input_10 (holding balloon)	-0.016940	0.151738	0.148955	
input_11 (holding flag)	-0.087298	0.196521	0.023554	
input_12 (jacket_color red)	5.735210	4.337359	-0.865479	
input_13 (jacket_color yellow)	-2.257168	-1.410376	0.494681	
input_14 (jacket_color green)	-2.232257	-1.109825	0.382717	
input_15 (jacket_color blue)	-1.710642	-1.452455	0.479513	
input_16 (has_tie yes)	-0.109696	0.434166	0.276487	
input_17 (has_tie no)	-0.111667	0.131797	0.310714	
bias	0.486541	0.142383	0.525371	
hidden_1				9.249339
hidden_2				8.639715
hidden_3				-9.419991
bias				-3.670920

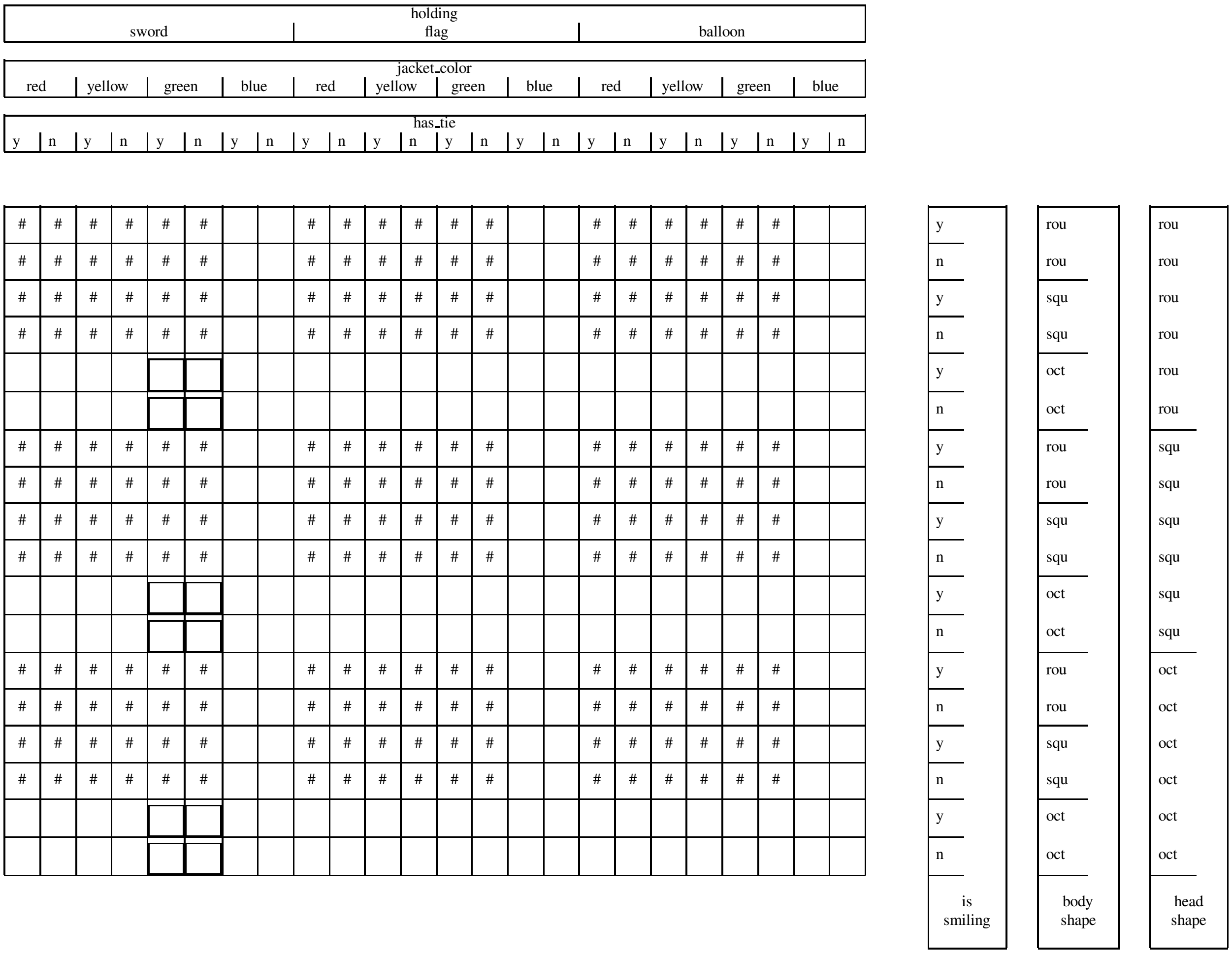
Figure 10: **MONK's problem $\mathbf{M}_1$:** The network learned and generalized successfully.

holding	sword								flag								balloon								is smiling	body shape	head shape
jacket_color	red		yellow		green		blue		red		yellow		green		blue		red		yellow		green		blue				
has_tie	y	n	y	n	y	n	y	n	y	n	y	n	y	n	y	n	y	n	y	n	y	n	y	n			
																									y	rou	rou
												#		#		#				#		#		#	n	rou	rou
												#		#		#				#		#		#	y	squ	rou
				#		#		#		#	#		#		#			#	#		#		#		n	squ	rou
												#		#		#				#		#		#	y	oct	rou
				#		#		#		#	#		#		#			#	#		#		#		n	oct	rou
												#		#		#				#		#		#	y	rou	squ
				#		#		#		#	#		#		#			#	#		#		#		n	rou	squ
				#		#		#		#	#		#		#			#	#		#		#		y	squ	squ
		#	#		#		#		#								#								n	squ	squ
				#		#		#		#	#		#		#			#	#		#		#		y	oct	squ
		#	#		#		#		#								#								n	oct	squ
												#		#		#				#		#		#	y	rou	oct
				#		#		#		#	#		#		#			#	#		#		#		n	rou	oct
				#		#		#		#	#		#		#			#	#		#		#		y	squ	oct
		#	#		#		#		#								#								n	squ	oct
				#		#		#		#	#		#		#			#	#		#		#		y	oct	oct
		#	#		#		#		#								#								n	oct	oct

MONK's problem $\mathbf{M}_2$: weights and biases

from-node	hidden_1	hidden_2	output
	to-node		
input_1 (head_shape round)	-4.230213	3.637149	
input_2 (head_shape square)	1.400753	-2.077242	
input_3 (head_shape octagon)	1.479862	-2.492254	
input_4 (body_shape round)	-4.363966	3.835199	
input_5 (body_shape square)	1.154510	-2.347489	
input_6 (body_shape octagon)	1.542958	-2.227530	
input_7 (is_smiling yes)	-3.396133	2.984736	
input_8 (is_smiling no)	1.868955	-2.994535	
input_9 (holding sword)	-4.041057	4.239548	
input_10 (holding balloon)	1.293933	-2.195403	
input_11 (holding flag)	1.160514	-2.272035	
input_12 (jacket_color red)	-4.462360	4.451742	
input_13 (jacket_color yellow)	0.749287	-1.869545	
input_14 (jacket_color green)	0.640353	-1.727654	
input_15 (jacket_color blue)	1.116349	-1.332642	
input_16 (has_tie yes)	-3.773187	3.290757	
input_17 (has_tie no)	1.786105	-3.296139	
bias	-1.075762	-0.274980	
hidden_1			-11.038625
hidden_2			-9.448544
bias			5.031395

Figure 11: **MONK's problem $\mathbf{M}_2$:** The network learned and generalized successfully.



MONK's problem $\mathbf{M}_3$: weights and biases			
	to-node		
from-node	hidden_1	hidden_2	output
input_1 (head_shape round)	-0.029477	-0.008986	
input_2 (head_shape square)	-0.376094	-0.364778	
input_3 (head_shape octagon)	-0.051924	-0.028672	
input_4 (body_shape round)	0.991798	0.991750	
input_5 (body_shape square)	1.031170	1.027708	
input_6 (body_shape octagon)	-1.284263	-1.279808	
input_7 (is_smiling yes)	-0.303940	-0.314212	
input_8 (is_smiling no)	-0.216766	-0.221040	
input_9 (holding sword)	-0.064305	-0.052110	
input_10 (holding balloon)	-0.257165	-0.243988	
input_11 (holding flag)	-0.131509	-0.122790	
input_12 (jacket_color red)	1.001415	1.004192	
input_13 (jacket_color yellow)	0.898066	0.896869	
input_14 (jacket_color green)	0.670929	0.673218	
input_15 (jacket_color blue)	-1.280272	-1.272798	
input_16 (has_tie yes)	-0.354472	-0.355268	
input_17 (has_tie no)	0.040973	0.037927	
bias	-0.319686	-0.343492	
hidden_1			1.762523
hidden_2			1.759077
bias			-1.501492

Figure 12: **MONK's problem $\mathbf{M}_3$:** Due to noise in the training set, the network had a small error in the classification, indicated by the boxes in the classification diagram.

Table 1: Rules extracted from the network trained on the MONK's problem $\mathbf{M}_1$.

head_shape	body_shape	is_smiling	holding	jacket_color	has_tie	class
	round			red		$\mathbf{M}_1$
	square			red		$\mathbf{M}_1$
	octagon			red		$\mathbf{M}_1$
round				red		$\mathbf{M}_1$
round	round					$\mathbf{M}_1$
square				red		$\mathbf{M}_1$
square	square					$\mathbf{M}_1$
octagon				red		$\mathbf{M}_1$
octagon	octagon					$\mathbf{M}_1$
round	square			yellow		not $\mathbf{M}_1$
round	square			green		not $\mathbf{M}_1$
round	square			blue		not $\mathbf{M}_1$
round	octagon			yellow		not $\mathbf{M}_1$
round	octagon			green		not $\mathbf{M}_1$
round	octagon			blue		not $\mathbf{M}_1$
square	round			yellow		not $\mathbf{M}_1$
square	round			green		not $\mathbf{M}_1$
square	round			blue		not $\mathbf{M}_1$
square	octagon			yellow		not $\mathbf{M}_1$
square	octagon			green		not $\mathbf{M}_1$
square	octagon			blue		not $\mathbf{M}_1$
octagon	round			yellow		not $\mathbf{M}_1$
octagon	round			green		not $\mathbf{M}_1$
octagon	round			blue		not $\mathbf{M}_1$
octagon	square			yellow		not $\mathbf{M}_1$
octagon	square			green		not $\mathbf{M}_1$
octagon	square			blue		not $\mathbf{M}_1$

because $\mathbf{M}_2$ is a parity problem. Consequently, VI-Analysis can only find a set of highly cluttered, specific rules that are hard to interpret by human observers. In order to prove the correctness of $\mathbf{M}_2$ efficiently, we extended the expressive power of the rule language in VI-Analysis to arbitrary linear constraints on the input activations of the network at hand. In the case of $\mathbf{M}_2$, we VI-analyzed the three simple linear constraints:

- if *the sum of all first feature values is* $\leq$ *1* then not $\mathbf{M}_2$.
- if *the sum of all first feature values is 2* then $\mathbf{M}_2$.
- if *the sum of all first feature values is* $\geq$ *3* then not $\mathbf{M}_2$.

Note that these rules are of the type m-of-n. VI-Analysis successfully verified all three conjectured rules. Thus, it was analytically shown that the network has learned the correct classification.

The VI-Analysis of $\mathbf{M}_2$ demonstrates the true expressive power of the VI-Analysis. The rule language consists of all sets of linear constraints on the input and the output activations, rather than hypercubes only. It should be noted, however, that no automated

Table 2: Rules extracted from the network trained on the MONK's problem $\mathbf{M}_3$.

head_shape	body_shape	is_smiling	holding	jacket_color	has_tie	class
	round			red		$\mathbf{M}_3$
	round			yellow		$\mathbf{M}_3$
	round			green		$\mathbf{M}_3$
	square			red		$\mathbf{M}_3$
	square			yellow		$\mathbf{M}_3$
	square			green		$\mathbf{M}_3$
				blue		not $\mathbf{M}_3$
	octagon					not $\mathbf{M}_3$

search was involved in analyzing $\mathbf{M}_2$, as the tested hypotheses were generated manually. Although mechanisms which search the space of all linear hypotheses in the input space (or whole sets thereof) can easily be designed, larger hypothesis spaces will imply an increased computational complexity for searching. Such mechanisms are beyond the scope of this paper.

- $\mathbf{M}_3$**.** (c.f. Figure 12). For this problem VI-Analysis gave the most satisfactory results. The rules generated by VI-Analysis for the $\mathbf{M}_3$ problem are listed in Table 2. Here VI-Analysis succeeded in extracting the most general rules. The two negative rules depicted in this Table suffice to describe the class $\mathbf{M}_3$ completely.

In summary, in most cases VI-Analysis was able to find the most general rules by trying few out of many rules only. It failed in one of these problems, namely $\mathbf{M}_1$, to find the most general rules, producing instead a slightly more complex set of rules. In the second example, which does not have a simple hypercube representation, the rule description language had to be extended to represent m-of-n type rules. Once the appropriate rules were provided by hand, VI-Analysis was able to check these extended rules as well, since the extended rules could also be expressed in terms of linear constraints.

4.3 A Robot Arm Domain

VI-Analysis was also applied to a robot arm kinematics domain. We trained an artificial neural network to model the forward kinematics function of a 5 degree-of-freedom robot arm. The arm, a Mitsubishi RV 41, is depicted in Figure 13. The kinematic function determines, based on the position of the 5 joints, the position of the tip of the manipulator in (x, y, z) workspace coordinates and the angle of the manipulator h to the table. The analytic

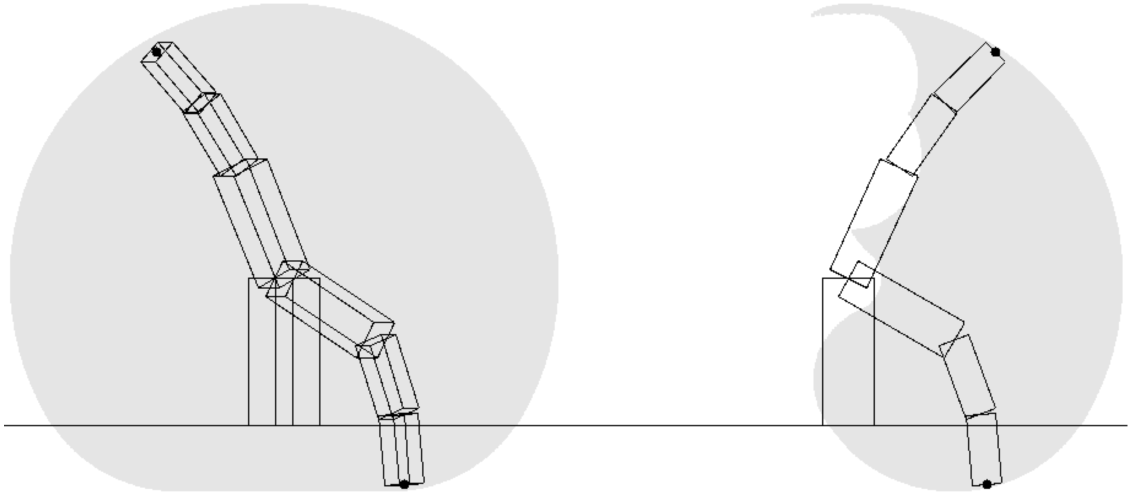


Figure 13: **The robot arm.** (a) Front view. Two different configurations are displayed. The grey area indicates the total workspace of the arm. Note that the workspace intersects partially with the table. (b) Two-dimensional view from the side, with a slice of the workspace.

kinematic equations are given by:

$$\begin{aligned}
x &= \sin(\alpha_1) \cdot [250\text{cm} \cdot \sin(\alpha_2) + 160\text{cm} \cdot \sin(\alpha_2 + \alpha_3) \\
&\qquad + 147\text{cm} \cdot \sin(\alpha_2 + \alpha_3 + 90° - \alpha_4)] \\
y &= \cos(\alpha_1) \cdot [250\text{cm} \cdot \sin(\alpha_2) + 160\text{cm} \cdot \sin(\alpha_2 + \alpha_3) \\
&\qquad + 147\text{cm} \cdot \sin(\alpha_2 + \alpha_3 + 90° - \alpha_4)] \\
z &= 300\text{cm} + 250\text{cm} \cdot \cos(\alpha_2) + 160\text{cm} \cdot \cos(\alpha_2 + \alpha_3) \\
&\qquad + 147\text{cm} \cdot \cos(\alpha_2 + \alpha_3 + 90° - \alpha_4) \\
h &= \alpha_2 + \alpha_3 - \alpha_4 \\
\text{with} &\quad \alpha_1 \in [-90°, 90°], \alpha_2, \alpha_3 \in [0°, 120°], \alpha_4 \in [-30°, 120°], \alpha_5 \in [-180°, 180°]
\end{aligned} \tag{1}$$

Note that the rotation of the wrist, given by α_5, does not play any role in these equations. Nonetheless it was used as an input to the network.

The robot arm is mounted on a flat table. As can bee seen in Figure 13, the workspace intersects with this table. Hence, some configurations of the joints are safe, namely those for which $z \geq 0$, while others cannot be reached physically without a collision that would damage the robot (unsafe). Therefore when operating the robot arm one has to be able to tell safe from unsafe. Having trained a neural network to model the kinematics of the robot

Figure 14: **Rule found by VI-Analysis.** The robot arm illustrates a configuration which is classified as being SAFE. The grey-shaded area indicates the preimage of the generalized rule. This region, too, will be predicted as being SAFE by the network. (a) Front view. (b) Two-dimensional side view (including a slice of the rule).

arm, one could, of course, check the safety of an individual joint configuration by predicting the workspace coordinates using the neural network. Here we are interested in a set of rules that describes the subspace of safe joint configurations. Since the configuration space is real-valued, rules can be extracted by VI-Analysis in combination with the rule extraction procedure described in Section 3.4.

We trained a network with 5 input units, 5 hidden units and 4 output units to model the kinematics of the robot arm. A total of 8 192 training examples was used, resulting in a considerably accurate model of the kinematics of the robot arm. After training, rules were extracted by enlarging random robot arm configurations.

For example, starting with the initial configuration

$$\vec{\alpha} = (30°; 80°; 20°; 60°; -20°),$$

random rule growing yields the following rule:

$$\underline{\mathit{if}} \quad \left\{ \begin{array}{rc} & 9.0586° \leq \alpha_1 \leq 61.5257° \\ \underline{and} & 64.469° \leq \alpha_2 \leq 87.7774° \\ \underline{and} & \alpha_3 \leq 44.0281° \\ \underline{and} & 39.601° \leq \alpha_4 \leq 79.708° \\ \underline{and} & -60.2571° \leq \alpha_5 \leq 20.7301° \end{array} \right\} \quad \underline{\mathit{then}} \quad \mathit{SAFE}$$

This rule is graphically depicted in Figure 14. Figure 14a shows the front view of the initial arm configuration. The grey shaded area are configurations for which the generalized rule applies, i.e., which will be predicted as being safe as well. Figure 14b shows a side view of

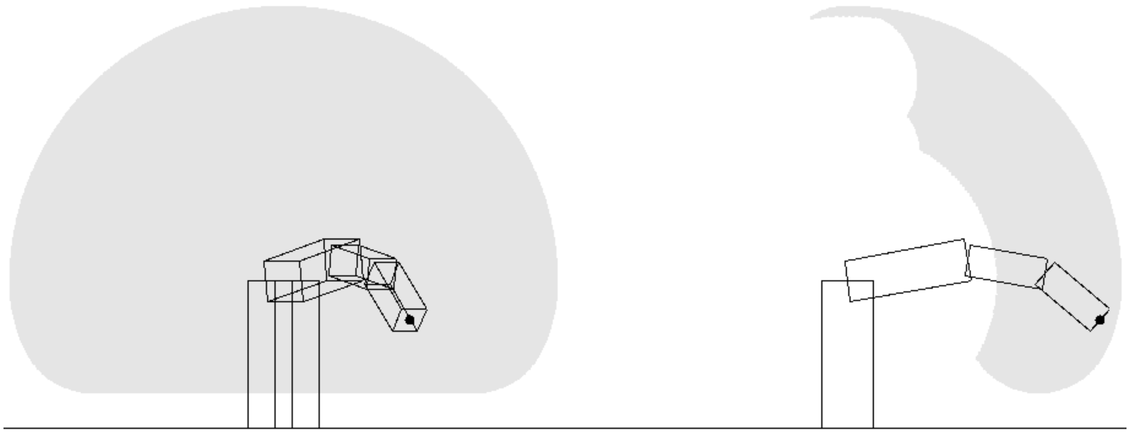


Figure 15: **Modified rule search.** Irrelevant interval bounds were eliminated first, before enlarging the rule by small random amounts.

the arm, along with a slice of the rule domain (i.e., the joint α_1 is kept fixed). It should be noted that the rule covers only a small faction (0.125%, measured by volume) of the total configuration space. This result is prototypical. Generally, for configurations close to the class boundary, i.e. where the tip of the manipulator is close to the table, we observed that the extracted rules were rather specific. If instead the initial configuration was closer to the center of a class, VI-Analysis was observed to produce more general rules that had a larger coverage in the workspace.

The plain rule growing routine enlarges intervals at random. Many learning problems studied in AI include completely irrelevant features, i.e., features whose values does not matter for the target concept. If this is the case, it might make sense to search for irrelevant features first when enlarging input intervals. For examples, there are irrelevant features in the robot arm domain. As can be seen from Equation (2), the z-coordinate, which is the crucial value for determining safety, does not depend on the configuration of the joints α_1 and α_5. Therefore, one would expect these dimensions to be promising directions for enlargement. In addition, one would expect that decreasing α_2, which will cause the arm to raise, will in most cases not make safe configurations unsafe. Consequently, there are certain interval bounds that could be omitted completely—for example those of α_1 and α_5, and the lower bound of α_2. In order to detect irrelevant features and eliminate whole interval bounds, the weight growing rule was modified: First, each individual bound was checked if it could be dropped completely. Subsequently, the resulting rule was enlarged even further by the random rule

Figure 16: Rule extraction from a network with 10 hidden units.

growing procedure. This algorithm has the advantage of identifying irrelevant input features in the first place, yielding rules that cover much larger parts of the input space. For example, for the same starting point reported above, the following rather general rule was found:

$$\underline{\textit{if}} \quad \left\{ \begin{array}{rc} & \alpha_2 \leq 90.5155^\circ \\ \underline{and} & \alpha_3 \leq 27.335^\circ \end{array} \right\} \quad \underline{\textit{then}} \quad \textit{SAFE}$$

This rule is visualized in Figure 15. Note that this single rule covers as much as 17.1% of the input space. Throughout our extensive tests we never observed rules with as large a coverage when growing intervals purely randomly.

In order to examine scaling properties of VI-Analysis, we also trained more complex networks to model the same kinematic function. In particular, in one experiment we doubled the number of units in the hidden layer, and in another experiment we introduced a second hidden layer with five hidden units. Example rules that were extracted from these more complex networks from the same initial instance are

$$\underline{\textit{if}} \quad \left\{ \begin{array}{rc} & 24.4317^\circ \leq \alpha_1 \leq 44.2328^\circ \\ \underline{and} & 75.379^\circ \leq \alpha_2 \leq 82.2432^\circ \\ \underline{and} & 11.7157^\circ \leq \alpha_3 \leq 29.4668^\circ \\ \underline{and} & 50.1783^\circ \leq \alpha_4 \leq 72.8586^\circ \\ \underline{and} & -40.9808^\circ \leq \alpha_5 \leq -0.393539^\circ \end{array} \right\} \quad \underline{\textit{then}} \quad \textit{SAFE,}$$

extracted from the network with 10 hidden units, and

$$\underline{\textit{if}} \quad \left\{ \begin{array}{rc} & 23.6702^\circ \leq \alpha_1 \leq 41.7284^\circ \\ \underline{and} & 76.5268^\circ \leq \alpha_2 \leq 81.1186^\circ \\ \underline{and} & 13.6148^\circ \leq \alpha_3 \leq 27.6168^\circ \\ \underline{and} & 55.0935^\circ \leq \alpha_4 \leq 64.9995^\circ \\ \underline{and} & -43.5665^\circ \leq \alpha_5 \leq -4.95944^\circ \end{array} \right\} \quad \underline{\textit{then}} \quad \textit{SAFE,}$$

Figure 17: Rule extraction from a network with two hidden layers, each consisting of 5 hidden units.

extracted from the network with 5 hidden units in each of two hidden layers. These more specific rules cover as few as 0.0016% and 0.00032%, respectively, of the total input space. They are illustrated in Figures 16 and 17, respectively. Note that these rules correspond directly to the rule shown in Figure 14, as they have been generated from the same initial configuration by randomly enlarging the intervals. It is clearly to be seen that more complex networks yield more specific rules. This effect was consistently observed for arbitrary initial configurations, although the differences were not always as strong as in this example. It is due to the way intervals are refined in VI-Analysis. As stated above, VI-Analysis is not guaranteed to detect inconsistencies, even if they exist and could be detected in principle. This is because weight layers are considered independently, and VI-Analysis is not able to propagate non-linear dependencies that arise across multiple layers. As more hidden units or hidden layers are introduced, the network becomes more complex and so will the non-linear dependencies. Consequently, the extracted rules are more specific, and more rules are needed to cover the whole input space.

We will now describe a series of experiments in which we extracted whole sets of rules in order to gradually replace the network. Assume, the kinematics network is repeatedly used for determining if positions are safe or not. Queries of this type can be used to initiate the extraction of a rule which generalizes this query. Assume such rules are memorized and later, as further queries arrive, used instead of the network, if possible. As the number of queries increases, the set of rules grows and covers increasing amounts of the input space. This will, since the sequential evaluation of a network is generally slow, speed up the classification of unknown instances—an effect that has been studied extensively in the context of symbolic problem solvers [Minton, 1988], [Veloso, 1992]. However, there is a trade-off, since as the corpus of rules becomes too large, searching through the rules may exceed the time required for network evaluation.

In a series of experiments we collected rules generated from random queries. The goal was

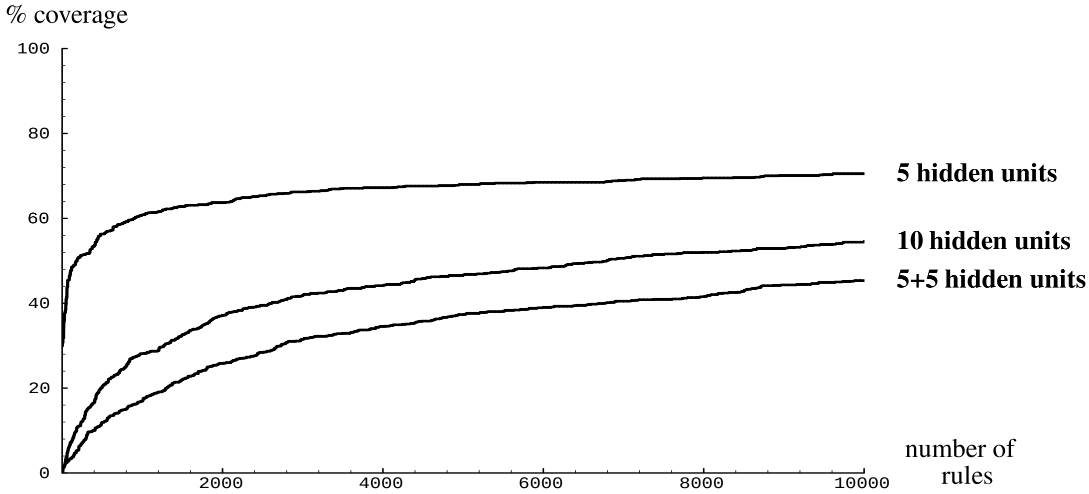


Figure 18: **Coverage.** Each curve plots the coverage as a function of the number of extracted rules for the three different networks.

to characterize the increased coverage by the rule set. The detailed procedure was as follows.

1. Generate a random joint configuration.
2. Query if a rule is already known for this joint configuration. If this is the case, continue at step 1.
3. Let the neural network predict the safety of the configuration.
4. Generate a generalized rule for this query and memorize it. Continue at step 1.

Figure 18 illustrates the result. The horizontal axis measures the number of memorized rules, and the vertical axis measures the degree as to which the input space is covered by these rules. As expected, the network with five hidden units yields the most general rules, and thus gives the largest coverage. For example, after the extraction of 185 rules 50% of the input space are covered by rules. The network with two hidden layers has, at any given point, the smallest coverage. As the number of queries goes to infinity, the coverage will approach 100% for any of these networks. It will, however, take infinitely many rules to cover the input space completely, as the decision boundaries for SAFETY are certainly non-linear.

The average coverage of a single rule, extracted from the three different networks, is summarized in the following table.

	5 hidden units	10 hidden units	5 and 5 hidden units
first 10 rules	3.54%	0.129%	0.0192%
first 100 rules	0.675%	0.0686%	0.0296%
first 1 000 rules	0.151%	0.0387%	0.0216%
first 10 000 rules	0.0215%	0.0120%	0.00902%

The numbers represent the fraction of the total input space in percent. Note that rules may overlap. It can be seen from these numbers that as the number of rules increases, the average coverage of new rules decreases. For example, while the average coverage for the network with 5 hidden units is initially 3.54%, after 10 000 rules the it has shrunk to 0.0215%. This is because with increased coverage most of the "easy-to-generalize" configurations are already covered by rules, and new rules will be generated closer and closer to the thin region separating safe and unsafe configurations. There, however, rules are increasingly more specific, as small variations may already change the classification. We also observed that positive rules, i.e., rules that describe safe positions, typically cover much larger fractions of the input space than negative rules. This is because it is much easier to generalize in the positive hemisphere, since the positive region is larger and much more homogeneous. The negative space is highly cluttered. Moreover, if negative configurations are drawn randomly with uniform distribution in the configuration space, almost all negative examples will be very close to the class boundary, where generalization is hard. For example, after the extraction of 10 000 rules using the network with 5 hidden units, 79.3% of all positive examples are covered by rules, but only 5.1% of the negative examples. The total coverage is 70.3%, since most examples are positive if joint configurations are sampled uniformly. A similar relation was found for the other networks.

It is apparent that more complex networks yield more specific rules. The region covered after the extraction of n rules from the smallest network was by a factor 2.2 (factor 3.1) larger than the corresponding region using a more complex network, featuring 10 hidden units (5 and 5 hidden units). These numbers represent the average for the first 10 000 rules. Although this difference is not as dramatic as the difference given in the example rules in Figures 14, 16 and 17, it nonetheless demonstrates that more general rules are generally extracted from smaller networks, which is what one would expect.

We repeated this experiment with the modified rule search mechanism, which eliminates irrelevant intervals first before rules are enlarged randomly. The results are depicted in Figure 19. The average coverage of rules, extracted from the three different networks using

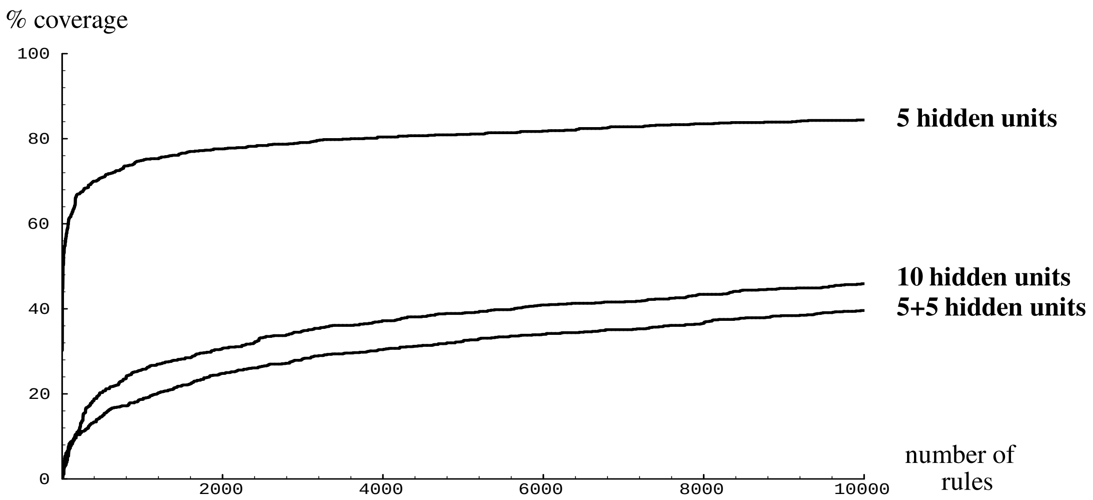


Figure 19: **Coverage using the modified rule search mechanism.** Each curve plots the coverage as a function of the number of extracted rules for the three different networks. Here, irrelevant interval bounds were searched first.

this modified extraction scheme, are:

	5 hidden units	10 hidden units	5 and 5 hidden units
first 10 rules	9.79%	0.254%	0.136%
first 100 rules	2.59%	0.181%	0.135%
first 1 000 rules	1.20%	0.0775%	0.0800%
first 10 000 rules	0.335%	0.0276%	0.0272%

For the small network (5 hidden units) the modified extraction rule is clearly superior to the pure random approach. For example, it takes only 10 rules to cover 50% of the input space, as opposed to 185 in the pure random approach. The coverage after the extraction of 10 000 rules is 84.4%, unlike 70.3% in the case of pure random rule growing. At any point in time, the total volume covered by rules was by a factor 1.21 larger than the total volume when rules were extracted via the pure random scheme. For larger networks, however, the modified extraction scheme did not outperform random weight growing, as can be seen from the comparison of Figure 18 and Figure 19. Closer inspection of the rules showed that here the initial elimination of irrelevant results rarely succeeded. In the cases where it did succeed, the coverage still remained rather small, and random rule growing barely succeeded in enlarging the rule by larger amounts. To conclude, the modified extraction scheme was found to be superior in some cases and inferior in others.

5 Related Work

Several researchers have proposed a variety of mechanisms to extract rules from artificial neural networks, which have been constructed from training data. Unlike the technique proposed in this paper, most of the approaches seek to assign semantic concepts to the individual hidden and output units of a network. Often they translate each hidden unit into a separate rule. For example, Towell and Shavlik [Towell, 1991], [Towell and Shavlik, 1992] describe a method which analyzes the weights and biases of a neural network in order to translate the network step-by-step into a set of rules with equivalent structure. In order to do so, the weights and biases of the network are truncated and discretized, resulting in approximately correct rules. These rules correspond directly to the links and units in the network. Consequently, the technique benefits if the networks at hand are only sparsely connected, and activation values are binary. In their approach, initial domain knowledge is employed for pre-structuring the networks. Similar methods have been proposed by Fu [Fu, 1989], and Mahoney and Mooney [Mahoney and Mooney, 1993], [Mahoney and Mooney, 1992]. Tresp and Hollatz [Tresp and Hollatz, 1993] and Giles and Omlin [Giles and Omlin, 1993] describe rule extraction methods for a restrictive class of network architectures with specific transfer functions. Tresp and Hollatz's method is restricted to single-layer networks with Gaussian activation functions. In the case of [Giles and Omlin, 1993], the networks are higher-order recurrent networks which are trained to approximate finite state automata.

In all the approaches listed above certain assumptions are made about the structure, as well as the sparseness of the networks, in order to make the task of rule extraction manageable. VI-Analysis differs from these approaches, since it analyses the network as a whole instead of translating the network unit-by-unit. Moreover, the rules found by VI-Analysis are provably accurate, making VI-Analysis a promising candidate for larger networks with multiple hidden layers and non-discrete representations.

Other rule extraction mechanisms rely on special training procedure that are applied during network training. For example, McMillan [McMillan, 1992], [McMillan *et al.*, 1992] describes a system in which the task of rule extraction is simplified by imposing regularization constraints on the network during training. Once the network is trained in this manner, dependencies are sparse, and the mapping to a set of rules is straightforward. A second neural network training scheme is described by Craven and Shavlik [Craven and Shavlik, 1993]. In their rule extraction algorithm, a weight regularization term is applied during training which aims at grouping weight values into discrete classes. Discrete weights facilitate the extraction of certain types of symbolic rules (namely m-of-n rules) from trained networks. Craven and Shavlik managed, most notably, to extract rules from a reduced version of Sejnowski and Rosenberg's NETtalk domain [Sejnowski and Rosenberg, 1986]. Note that the weight regularization term may replace the need for initial knowledge, as reported in [Towell, 1991] and [Towell and Shavlik, 1992]. In both of these extraction schemes the effectiveness of the rule extraction mechanism, as well as the degree of correctness of the extracted rules, relies crucially on the particular training procedure invoked. VI-Analysis differs from these approaches in that it does not make any assumptions regarding the training procedure for the network at hand. Rule extraction based on VI-Analysis is thus applicable to a much broader class of networks.

VI-Analysis bears close resemblance to sensitivity analysis. Unlike all of the above approaches which translate networks unit-by-unit, sensitivity analysis characterizes the network output by systematic variations in the input patterns and examining the changes in the network classification (in some cases including the changes in its derivatives). Like VI-Analysis, sensitivity analysis analyses the network as a whole. An approach to rule extraction based on sensitivity analysis has been proposed by Goh and Wong [Goh and Wong, 1993]. Sensitivity analysis, however, yields only approximately correct rules.[9]

It should be noted that once a network has been trained, the one-to-one compilation of networks into rules is much faster than the extraction of rules through VI-Analysis. It should also be noted that some of the rule extraction mechanisms listed above have not been designed with the same objectives as the method proposed in this paper. For example, some researchers have argued that rule-enforcing constraints on the training procedure, as well as certain structure on the network topology, might significantly improve the generalization rate, given that the target concept can be easily described by rules. This observation, which undoubtedly is valid, is to be kept in mind when comparing different approaches to the extraction of rules. Since in this paper we are mainly interested in rule extraction from arbitrary networks, we have made no assumption about the training procedure at hand. It should be noted that rule extraction using VI-Analysis is also applicable to those more structured networks. Indeed, we expect the resulting rule sets to be even better if rule-enforcing regularization terms are already applied during training.

[9] It seems feasible that further analysis of the magnitude of the weights and biases of the network (or alternatively the higher-order input-output derivatives) can be employed to generate provably correct rules based upon sensitivity analysis.

6 Discussion

In this paper we have proposed a technique for extracting *if-then* type rules from arbitrary, pre-trained artificial neural networks. The major mechanism for the extraction of rules is VI-Analysis. VI-Analysis is a generic tool for analyzing trained neural networks by propagating and refining rule-knowledge (validity intervals) through the network in both forward and backward direction. This paper demonstrates how VI-Analysis can be employed as a proving-engine for the verification of symbolic rules. On top of this, two systematic rule search schemes, one for discrete domains and one for domains with real-valued features, are proposed. They have been evaluated empirically, using the XOR function, the MONK's benchmark problems, and a robot arm kinematics domain.

The approach described in this paper differs from the majority of approaches to rule extraction in several aspects. Most importantly, VI-Analysis analyzes a network as a whole, rather than translating the network into rules unit-by-unit. This has the advantage that rules can be extracted from complex and densely interconnected networks with distributed, hard-to-interpret representations. It comes at the price of increased computational requirements for the extraction of rules. While the one-to-one translation of rules is typically at most quadratic in the number of units of the network, VI-Analysis employs multiple runs of a linear programming algorithm, which, in the current implementation, may take exponential time. We were able to apply this technique successfully to the MONK's problems and a robot kinematic domain. The complexity of both of these domains is moderate. In the more complex domain of NETtalk [Sejnowski and Rosenberg, 1986], [Craven and Shavlik, 1993] we failed in generating complete set of rules. This is because there are 203 input units, which code a window of 7 letter-like tokens, when using the standard NETtalk representation. A complete DAG has thus $3^{203} \approx 7.17 \cdot 10^{96}$ nodes. A reduced DAG, in which only seven single letter can occur in the preimage of a rule instead of seven sets of letters, still has $30^7 \approx 2.19 \cdot 10^{10}$ nodes. Even if VI-Analysis manages to shortcut large chunks of this graph by showing the correctness of more general rules, the number of rules to be checked is still far beyond the abilities of our computers. On the other hand, we found that once meaningful rules were proposed by hand in the NETtalk domain, VI-Analysis managed to verify these rules. Hence, more efficient schemes for generating candidate rule sets are needed.

In this paper we proposed two basic approaches for generating candidate rules, one for discrete domains, and one for domains with real-valued features. The latter mechanism, for example, uses a random search strategy to enlarge the preimage of a rule progressively.

There are a wide variety of more advanced search strategies that can be used to generate rule sets. For example, parallel search techniques such as Genetic Algorithms [Holland, 1984], [Goldberg, 1989] are applicable to the problem of finding rule with maximal coverage. At a first glance, the genetic string could encode the current setting of validity intervals, and the performance measure to be maximized may be the volume covered by these intervals, or a similar function. Symbolic learning algorithms such as decision tree learning [Quinlan, 1986] may also be used to generate rule hypotheses. Symbolic learning procedures directly generate sets of rules which approximate the set of training instances. Once such rules have been generated, they are promising candidates for the description of an artificial neural network trained on the same data set.

VI-Analysis supports the extraction of propositional *if-then* rules that can be expressed by a combination of linear constraints. We have demonstrated that this language suffices for expressing hypercube constraints along with other linear constraints. It furthermore allows the representation of m-of-n rules. In fact, the language of linear constraints includes most types of rules studies in the context of artificial neural networks. It excludes, however, various rule types of rules studied in symbolic AI. To give a simple example, the rule *"the output is always smaller than the input"* can not be verified by VI-Analysis, since linear constraints may not be applied across several layers. VI-Analysis also excludes higher-order rules, i.e., rules over products of variables, as studied for example by Giles and Omlin [Giles and Omlin, 1993]. If such rules are to be verified, non-linear optimization techniques must replace the Simplex algorithm. More general rule languages, however, are clearly desirable, and point out a promising and important direction for future research.

Another key characteristic of VI-Analysis stems from the way intervals are refined and propagated through the network. VI-Analysis excludes only those activations from the set of valid activations which can be shown analytically to be inconsistent with the initial conditions and the weights and biases of the network. As a consequence, the extracted rules are provably correct, in the sense that they are provably consistent with the network at hand. This is considered to be a strength of this approach, since the accuracy of the rules does not depend on the complexity of the network. Obviously, rule extraction mechanisms with limited accuracy are less likely to scale to large networks with many hidden layers. On the other hand, the correctness also imposes limitations. VI-Analysis might yield overly specific rules. This has been illustrated, for example, by varying the complexity of the networks in the robot arm kinematics domain in Section 4. VI-Analysis might fail to verify rules, even if they are consistent with the network, because individual weight layers are analyzed independently of each other. Rule verification mechanisms that analyze several weight layers simultaneously offer a promising alternative for the extraction of rules. Since networks represent non-linear functions, non-linear search techniques would have to come to bear. Non-linear optimization usually may suffer from local minima, and it is generally unclear whether the resulting rules would still accurately describe the underlying network. Sacrificing the high accuracy goal, however, can be appropriate if the resulting rule verification algorithms turns out to give better and more compact results.

One of the key features of the approach taken is that no architectural requirements regarding the network at hand are made. Indeed, by analyzing the network as a whole rather than compiling networks unit-by-unit, rules can be extracted from densely interconnected networks with distributed representations and arbitrary real-valued weights and biases. This includes recurrent networks[10] not described here. Since VI-Analysis analyses networks after training, the rule extraction mechanism described in this paper does not require any special training procedure either. Many neural network applications that have proven to be successful in practice have not been trained to facilitate the extraction of rules. VI-Analysis is thus generally applicable to a rather broad class of artificial neural networks.

Two important assumption of VI-Analysis in its current form, however, are the monotonicity and the continuity of the transfer functions in the network. These assumptions can be relaxed to piecewise monotonic and piecewise continuous transfer function, resulting in assigning whole sets of intervals to each unit. Such transfer functions include, for example, radial-basis functions [Moody and Darken, 1989] which have been frequently used in artificial neural networks. An extended VI-Analysis which propagates sets of intervals—rather than single intervals—is straightforward. Such an algorithm will come, however, at the price of drastically increased computational complexity. It remains to be shown whether the VI-Analysis of sets of intervals will be efficiently applicable in practice.

The goal of this paper is to attack one of the major shortcomings of artificial neural network representations, namely their lack of human expert comprehensibility. This paper demonstrates how to extract symbolic rules from a broad class of Backpropagation-style neural networks. In addition, an empirical study using the MONK's problems and a robot arm kinematics problem as a testbed is described in order to characterize how the proposed rule extraction scheme works in practice. While it is clear that these mechanism do not estab-

[10] See for example [Jordan, 1986], [Elman, 1988], and [Williams and Zipser, 1989] for literature on recurrent networks

lish a final solution to the problem of extracting rules, we are confident that rule extraction via VI-Analysis can shed light into a variety of real-world applications of artificial neural networks.

Acknowledgment

I wish to thank Tom Dietterich, Clayton McMillan, and Tom Mitchell for their invaluable feedback that has influenced this research. I thank Mark Craven and Clayton McMillan for thoughtful comments on an earlier draft of this paper.

References

[Craven and Shavlik, 1993] Mark W. Craven and Jude W. Shavlik. Learning symbolic rules using artificial neural networks. In Paul E. Utgoff, editor, *Proceedings of the Tenth International Conference on Machine Learning*, San Mateo, CA, 1993. Morgan Kaufmann. to appear.

[DeJong and Mooney, 1986] Gerald DeJong and Raymond Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176, 1986.

[Elman, 1988] Jeffrey L. Elman. Finding structure in time. Technical Report CRL Technical Report 8801, Center for Research in Language, University of California, San Diego, 1988.

[Fu, 1989] Li-Min Fu. Integration of neural heuristics into knowledge-based inference. *Connection Science*, 1(3):325–339, 1989.

[Giles and Omlin, 1993] C. Lee Giles and Christian W. Omlin. Rule refinement with recurrent neural networks. In *Proceedings of the IEEE International Conference on Neural Network*, pages 801–806, San Francisco, CA, March 1993. IEEE Neural Network Council.

[Goh and Wong, 1993] T. G. Goh and Francis Wong. Semantic extraction using neural network modelling ans sensitivity analysis. To be found in the neurprose archive (anonymous ftp from archive.cis.ohio-state.edu:pub/neuroprose/thgoh.sense.ps.Z), 1993.

[Goldberg, 1989] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[Holland, 1984] John H. Holland. Genetic algorithms and adaptation. In *Proceedings of Ill-Defined Systems*, England, 1984.

[Hsu and Simmons, 1991] Goang-Tay Hsu and Reid Simmons. Learning footfall evaluation for a walking robot. In *Proceedings of the Eigth International Workshop on Machine Learning*, pages 303–307, Evanston, IL, June 1991.

[Jabri *et al.*, 1992] M. Jabri, S. Pickard, P. Leong, Z. Chi, B. Flower, and Y. Xie. ANN based classification for heart defibrillators. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 637–644, San Mateo, CA, 1992. Morgan Kaufmann.

[Jordan, 1986] Michael I. Jordan. Serial order: A parallel distributed processing approach. Technical Report ICS Report 8604, Institute for Cognitive Science, University of California, 1986.

[Karmarkar, 1984] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[LeCun *et al.*, 1990] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1990.

[Mahoney and Mooney, 1992] J. Jeffrey Mahoney and Raymond J. Mooney. Combining symbolic and neural learning to revise probabilistic theories. In *Proceedings of the 1992 Machine Learning Workshop on Integrated Learning in Real Domains*, Aberdeen Scotland, July 1992.

[Mahoney and Mooney, 1993] J. Jeffrey Mahoney and Raymond J. Mooney. Combining neural and symbolic learning to revise probabilistic rule bases. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 5*, San Mateo, CA, 1993. Morgan Kaufmann. (to appear).

[McMillan *et al.*, 1992] Clayton McMillan, Michael C. Mozer, and Paul Smolensky. Rule induction through integrated symbolic and subsymbolic processing. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 969–976, San Mateo, CA, 1992. Morgan Kaufmann.

[McMillan, 1992] Clayton McMillan. *Rule Induction in a Neural Network through Integrated Symbolic and Subsymbolic Processing*. PhD thesis, University of Colorado, Department of Computer Science, Boulder, 1992.

[Minton, 1988] Steven Minton. *Learning Search Control Knowledge: An Explanation-Based Approach*. Kluwer Academic Publishers, 1988.

[Mitchell *et al.*, 1986] Tom M. Mitchell, Rich Keller, and Smadar Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80, 1986.

[Moody and Darken, 1989] John Moody and Chris Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[Pomerleau, 1989] D. A. Pomerleau. ALVINN: an autonomous land vehicle in a neural network. Technical Report CMU-CS-89-107, Computer Science Dept. Carnegie Mellon University, Pittsburgh PA, 1989.

[Press, 1988] William H. Press. *Numerical recipes in C : the art of scientific computing*. Cambridge University Press, Cambridge [Cambridgeshire]; New York, 1988.

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

[Rumelhart *et al.*, 1986] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing. Vol. I + II.* MIT Press, 1986.

[Sejnowski and Rosenberg, 1986] T. J. Sejnowski and C. R. Rosenberg. Nettalk: A parallel network that learns to read aloud. Technical Report JHU/EECS-86/01, John Hopkins University, 1986.

[Tesauro, 1992] Gerald J. Tesauro. Practical issues in temporal difference learning. *Machine Learning Journal*, 8, 1992.

[Thrun and Linden, 1990] S. Thrun and A. Linden. Inversion in time. In *Proceedings of the EURASIP Workshop on Neural Networks*, Sesimbra, Portugal, Feb 1990. EURASIP.

[Thrun *et al.*, 1991] Sebastian B. Thrun, Jerzy Bala, Eric Bloedorn, Ivan Bratko, Bojan Cestnik, John Cheng, Kenneth De Jong, Saso Džeroski, Douglas Fisher, Scott E. Fahlman, Rainer Hamann, Kenneth Kaufman, Stefan Keller, Igor Kononenko, Juergen Kreuziger, Ryszard S. Michalski, Tom Mitchell, Peter Pachowicz, Yoram Reich, Haleh Vafaie, Walter Van de Welde, Walter Wenzel, Janusz Wnek, and Jianping Zhang. The MONK's problems - a performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, Pittsburgh, PA, December 1991.

[Thrun, 1993] Sebastian B. Thrun. Extracting provably correct rules from artificial neural networks. Technical Report IAI-TR-93-5, University of Bonn, Institut für Informatik III, D-53117 Bonn, May 1993.

[Towell and Shavlik, 1992] Geoffrey Towell and Jude W. Shavlik. Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 977–984, San Mateo, CA, 1992. Morgan Kaufmann.

[Towell, 1991] Geoffrey G. Towell. *Symbolic Knowledge and Neural Networks: Insertion, Refinement and Extraction*. PhD thesis, University of Wisconsin–Madison, 1991.

[Tresp and Hollatz, 1993] Volker Tresp and Jürgen Hollatz. Network structuring and training using rule-based knowledge. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 5*, San Mateo, CA, 1993. Morgan Kaufmann. (to appear).

[Veloso, 1992] Manuela M. Veloso. *Learning by Analogical Reasoning in General Problem Solving*. PhD thesis, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA, August 1992.

[Waibel, 1989] A. H. Waibel. Modular construction of time-delay neural networks for speech recognition. *Neural Computation*, 1:39–46, 1989.

[Williams and Zipser, 1989] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989. also appeared as: Technical Report ICS Report 8805, Institute for Cognitive Science, University of California, San Diego, CA, 1988.

[Wnek *et al.*, 1990] J. Wnek, J. Sarma, A. Wahab, and R. Michalski. Comparison learning paradigms via diagrammatic visualization: A case study in single concept learning using symbolic, neural net and genetic algorithm methods. Technical report, George Mason University, Computer Science Department, 1990.