Extracting Provably Correct Rules 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@cs.uni-bonn.de thrun@cmu.edu
Phone: +49-228-550-260 FAX: +49-228-550-382

Abstract

Although connectionist learning procedures have been applied successfully to a variety of real-world scenarios, artificial neural networks have often been criticized for exhibiting a low degree of comprehensibility. Mechanisms that automatically compile neural networks into symbolic rules offer a promising perspective to overcome this practical shortcoming of neural network representations.

This paper describes an approach to neural network rule extraction based on *Validity Interval Analysis* (VI-Analysis). VI-Analysis is a generic tool for extracting symbolic knowledge from Backpropagation-style artificial neural networks. It does this by propagating whole intervals of activations through the network in both the forward and backward directions. In the context of rule extraction, these intervals are used to prove or disprove the correctness of conjectured rules. We describe techniques for generating and testing rule hypotheses, and demonstrate these using some simple classification tasks including the MONK's benchmark problems. Rules extracted by VI-Analysis are provably correct. No assumptions are made about the topology of the network at hand, as well as the procedure employed for training the network.

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 successfully applied to problems of learning and generalization in a variety of real-world scenarios[1]. While it has frequently been reported that neural network classifiers are able to achieve high generalization accuracy across various application domains, the classification concepts of those networks are usually barely comprehensible to the human users. This is because typical neural network solutions consist of a large number of interacting non-linear elements, characterized by large sets of real-valued parameters that are hard to interpret. Distributed internal representations, which typically emerge during network training, make it even harder to understand *what exactly* a network has learned, and where it will fail to generate the correct answer.

To date, the most common way to characterize the result of neural network learning is statistical in nature. Supervised training procedures, such as the Backpropagation algorithm [Rumelhart *et al.*, 1986], approximate an unknown target function by iteratively minimizing the output error based on a finite set of pre-classified training instances. This iterative process is usually referred to as *(supervised) training* or *learning*.[2] After training, the network is usually evaluated by measuring its generalization accuracy, which is estimated via a separate hold-out set, i.e., the network is "tested" using further instances that were not used during training. Often, by performing several training runs the variance of the generalization rate is estimated as well. Although this statistical method provides a means to foresee future misclassification rates, it requires that sufficient testing data is available and that future queries are drawn from the same distribution as the hold-out set. If either of these conditions is not fulfilled, generalization rates alone are not appropriate to characterize the performance of the network. Hence, in many application domains it is desirable to understand the classification function realized by the neural network in more detail.

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 procedures seek to generate small sets of sparse rules that fit the observed training data. Such rules can usually be much better interpreted by humans and are thus easier to understand. Moreover, symbolic rules allow for interfacing with various knowledge-based systems, such

[1] See [Sejnowski and Rosenberg, 1986], [Waibel, 1989], [Pomerleau, 1989], [LeCun *et al.*, 1990], [Jabri *et al.*, 1992], and [Tesauro, 1992] for few out of many examples.

[2] The primary focus of this paper will be on Backpropagation-style networks which have learned some classification task.

as expert systems or intelligent databases. If neural networks are the learning method of choice (e.g., 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. There is a variety of techniques available which vary in the type of rules, in the requirements they make on the networks, as well as the training procedures employed for learning. Since densely interconnected, real-valued networks seem to be hard to map into rules, these requirements often aim at both lowering the degree of interconnection and discretizing the real-valued parameters of the networks. Consequently, such requirements usually limit the methods at hand to rather special network architectures and training schemes. Many networks that have been successfully applied in practice, however, do not fulfill such requirements, and for those networks the proposed techniques fail in extracting meaningful and sufficiently correct rules. Our goal is to design a more general mechanism which is applicable to a broader class of artificial neural networks. Hence, we desire the following list of properties:

1. **No architectural requirements.** A general rule identification mechanism is able to operate with all types of artificial neural networks, including densely interconnected, unstructured, and recurrent networks.

2. **No training requirements.** Some of the more powerful extraction algorithms proposed in the literature rely on special training procedures which facilitate the extraction of rules. Such procedures have limited applicability, since they usually cannot be applied to networks which have been trained with other learning methods. We desire an algorithm that makes no assumption as to the way the network has been constructed and the weights and biases have been learned.

3. **Correctness.** Rule extraction mechanisms differ in the degree of correctness of the extracted rules. Many approaches to rule extraction generate only approximations to the underlying network. In order to describe the function realized by a neural network accurately, it is desirable that the extracted rules describe the neural network as correctly as possible.

4. **High expressive power.** The language of rules (syntax) characterizes the compactness

of the extracted symbolic knowledge. Powerful languages are generally desirable, since more compact rule sets are often easier to understand.

We are not aware of any rule extraction technique which matches all of the above requirements. Most techniques published thus far yield approximately correct rules only, they rely on a specialized training routine, and/or make major restrictive assumptions about the topology of the network at hand. The rule extraction technique described in this paper generates if-then-type rules that are provably correct, making minimal assumptions concerning the type of network, and no assumption as to the particular training routine employed. This technique is based on a generic tool for analyzing dependencies within neural networks, called Validity Interval Analysis (VI-Analysis or VIA) [Thrun and Linden, 1990]. VI-Analysis iteratively analyzes the input-output functionality of an artificial neural network by propagating sets of intervals through the network. The basic notion of VI-Analysis is presented in Section 2, followed in Section 3 by a demonstration using the simple boolean function XOR. Subsequently, in Section 4, we will show how VI-Analysis can be applied to verify rule-like hypotheses. We will then describe several strategies to generate such hypotheses (Section 5), which completes the description of the rule extracting mechanism. Section 5 also includes empirical evaluations of rule extraction and verification using the XOR problem, as well as the more complex MONK's problems. The paper is concluded by a brief survey of related approaches (Section 6) and a discussion (Section 7).

2 Validity Interval Analysis

Validity Interval Analysis (VIA or VI-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 logically 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.

In the context of rule extraction, initial validity intervals will be generated by a separate search routine (c.f. Section 5). Starting with an initial set of validity intervals, 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 an important 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.

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

If unit i happens to be an input unit, its activation value will simply be the external input value. If not, 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. 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 it 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 intervals are chosen to be $x_1 \in [0; 0.2]$ and $x_2 \in [0.8; 1]$. The output activation is unconstrained. (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* $x_3 \in [0.05732; 0.2314]$ (and thus $x_3 < 0.5$).

---

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.

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

Figure 2: **VI-Analysis with open intervals:** (a) Analogous situation as in the previous figure, 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* $x_3 \in [0.002472; 0.2314]$.

---

2. **Forward phase with unconstrained input values:** Figure 2 demonstrates the same forward phase, but one of the input activations is unconstrained. 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* $x_3 \in [0.002472; 0.2314]$, as shown in Figure 2.

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 on the input activation $x_2 \geq 0.8$ and one on 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 the forward propagation phase, as described above. Note that in Figure 3d only the upper bound b_3 of the output interval is modified—the initial lower 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

[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) Initial intervals constrain both an input activation and an 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* $x_1 \geq 0.8465$.

$net_3 = w_{31}x_1 + w_{32}x_2 + \theta_3$. 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* i *may receive a net-input within their validity interval* $[a'_j; b'_j]$*, given that all 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 the monotonicity of the transfer function ensures that the projections of intervals are again intervals. 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 in the next section.

Figure 4: **VI-Analysis and inconsistencies:** (a) As in the previous figure, there are two initial intervals assigned to an input unit and an 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.

---

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 would 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 be specified by the user. In the case of rule extraction, they are generated automatically by some search mechanism.

- Constraints are propagated in both directions through the network by refining intervals. All refinements made during this process preserve the consistent activation space and can be proven to be correct (VIA is truth-preserving).

Figure 5: **Weight layer:** The set of units in layer $\mathcal{P}$ are connected to the unit in layer $\mathcal{S}$. Each unit j ($j \in \mathcal{P} \cup \mathcal{S}$) is assigned a validity interval $[a_j; b_j]$. 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 first-order constraints on a linear set of equations given by the weights and biases of the layer at hand. Linear programming is now employed to refine interval bounds.

- VI-Analysis converges to a refined set of intervals and might or might not 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. VI-Analysis can easily 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. 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 $[a'_i; b'_i] = \sigma^{-1}([a_i; b_i]) \in \overline{\Re}^2$.[6] The resulting validity intervals in $\mathcal{P}$ and $\mathcal{S}$ form a set of linear constraints on the activation values in $\mathcal{P}$:

$$\begin{array}{lll} \forall k \in \mathcal{P}: & x_k & \geq a_k \\ & x_k & \leq b_k \end{array}$$

$$\begin{array}{lll} \forall i \in \mathcal{S}: \quad net_i = \sum_{k \in \mathcal{P}} w_{ik} x_k + \theta_i & \geq & a_i \\ net_i = \sum_{k \in \mathcal{P}} w_{ik} x_k + \theta_i & \leq & b_i \end{array}$$

Notice that all these constraints are linear in the activation values x_k. Thus, linear programming can be applied to refine lower and upper bounds for validity intervals individually. As in the AND example, constraints are propagated in two phases:

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 $\hat{b}_i$:

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

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

   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 again using linear programming.

$$\begin{array}{lll} \hat{a}_k & = & \min x_k \\ \hat{b}_k & = & \max x_k \end{array}$$

   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 the update rule in both phases ensures that intervals are changed monotonically. This implies the convergence of VI-Analysis.

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

In our experiments, we used the Simplex algorithm. 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 slow in practice. It should be noted that there exist more complex algorithms for linear programming which are worst-case polynomial [Karmarkar, 1984].[7]

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 until convergence is observed, using one of the standard convergence criteria. 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, validity interval 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 clearly neglects dependencies on the activations in $\mathcal{P}$ which may arise from preceding weights and activations beyond the scope of the linear analysis, resulting in an overly careful refinement of validity intervals. If such dependencies are considered, plain linear programming techniques are no longer applicable. In our initial experiments described in this paper, however, we found VI-Analysis to be powerful enough to successfully analyze all networks we applied it to.

3 Experimental Results for the XOR Problem

In an initial experiment we applied VI-Analysis to a neural network which was trained to approximate the boolean function XOR. The network, including its weights and biases, is depicted in Figure 6. It was trained using the standard Backpropagation algorithm. Five runs of VI-Analysis with different initial conditions were performed to evaluate the classification of the network and to demonstrate VI-Analysis. Figure 7 displays the results. Each row summarizes a single experiment, and each diagram shows the refinement of validity intervals over time. Each of the five columns corresponds to a particular unit in the network.

[7] In an earlier paper [Thrun and Linden, 1990] we proposed a more straightforward algorithm which works much faster by sacrificing some solutions to the interval refinement problem.

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

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 correct[8] classification of the network in this interval. It should be noted that a tighter upper bound on the output activation is found by this analysis as well.

2. Same experiment as above, but in this case both inputs are constrained to be low. Consequently, VI-Analysis results in a low output interval.

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

4. Same situation, but this time the output is 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 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. Contradictions play a crucial role in the general rule verification mechanism described in the next section.

[8] correct in the sense of a to real-valued input extended XOR

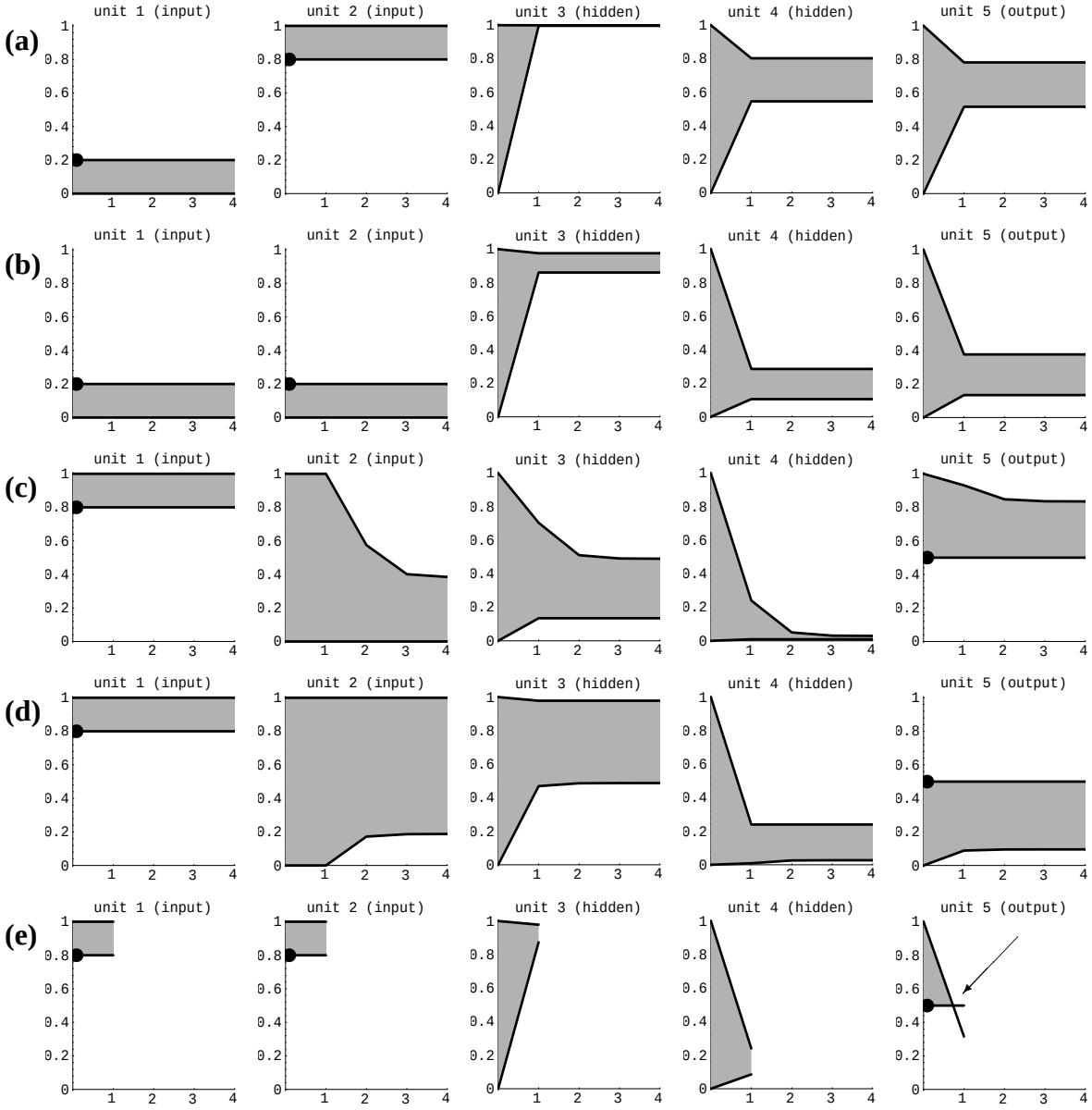


Figure 7: **VI-Analysis on XOR:** Each row summarizes one experiment. The gray area indicates the development of validity intervals over the iterations of VI-refinement for each of the 5 units (columns) in the XOR network. Dots indicate the initial setting of the intervals. (a) Forward propagation: If 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) Contradiction: 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.

4 Rule Verification

VI-Analysis can be applied to verify the correctness of hypotheses on the network classification. Generally speaking, rule hypotheses are represented by a set of validity intervals on (a subset of) the input units of the trained network, as well as a target output class. Thus, they correspond to rules of the type:

If input $\in$ *some hypercube* $\mathcal{I}$ *then class is* C.

If input $\in$ *some hypercube* $\mathcal{I}$ *then class is not* C.

for some target class C. Here $\mathcal{I}$ abbreviates the antecedent 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

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

Rules are verified by contradictions. More specifically, let $via_{\mathcal{I}}[x_j < x'_j]$ denote the truth value of running validity interval analysis when the (linear) output constraint $x_j < x'_j$ is added to the initial interval constraints $\mathcal{I} = [a_i, b_i]^m$. If $via_{\mathcal{I}}[x_j < x'_j] = false$, a contradiction has occurred during VI-Analysis. Hence the initial input intervals are inconsistent with the conjecture $x_j < x'_j$, and thus $x_j \geq x'_j$. If $via_{\mathcal{I}}[x_j < x'_j] = true$ VI-Analysis converged without contradiction, and 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 $j = 1, \ldots, c$. Then the rule

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

is provably correct if

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

This is to say, 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'. Consequently, output unit j will always take the largest activation value among all output units. 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

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

can be verified by showing that

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

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 proves the conjectured rule.

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 powerful tool for verifying the correctness of conjectured rules. It does this by (a) inverting the rule and (b) detecting logical contradictions. 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.

5 Rule Extraction Strategies

The previous section describes how to apply VI-Analysis to verify the correctness of conjectured rules. In order to *extract* rules, mechanisms have to be invoked that systematically

generate hypotheses which are subsequently verified by VI-Analysis. We will now describe search heuristics that allow for systematically generating and testing rules in order to find (a) the most general and (b) the most relevant rules. We will describe both a search technique for finite (discrete) domains and one for real-valued domains. These heuristics by no means establish the uniquely best technique for generating rules—depending on the classification domain and potential background knowledge, rule extraction strategies may vary among different application domains. In our experiments, however, they turned out to be appropriate for extracting reasonably general rules.

5.1 Discrete Rules and Graph Search

If the domain of the networks is *finite*[9], i.e., if there are finitely many instances that exhaustively characterize the target function, there is a simple but expensive strategy to extract rules from arbitrary trained 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 a set of expressions in disjunct normal form, one for each of the c classes). Common examples for finite input domains are classification tasks over binary feature vectors. While this straightforward rule extraction technique will always generate a set of rules that correctly describes the function represented by the network, the process of testing each input pattern in the domain of the network is expensive and usually computationally intractable for larger domains.

In finite domains VI-Analysis can be applied to cut whole regions in the search space by evaluating more general rules. To see this, consider the space of possible rules. This space can be described by c directed acyclic graphs (DAGs) in which nodes correspond to rules, which are ordered from the most general to the most specific. The root of such a graph forms the most general rule and is of the type *everything is in class* C. The leaves correspond to a single instance in the input space of the network. Figure 8 shows such a rule search graph. Each rule in the graph spans a subtree of more specific rules which logically follow from this rule. VI-Analysis is now applied in breadth-first manner: As soon as VI-Analysis proves the correctness of a general rule, the whole subgraph spanned by the corresponding rule node belongs to the same class and needs no further investigation. The breadth-first search scheme may decrease the number of tests significantly. This reduction of the search space, however,

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

Figure 8: **Example rule-DAG:** A 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 subtree spanned by this rule is removed from the search list.

---

comes at the expense of running VI-Analysis instead of single forward propagations, as is required for the pure classification of an instance.

5.2 Results Obtained for the MONK's Problems

We applied DAG breadth-first search to the MONK's problems. The three MONK's problems constitute a set of benchmark problems for inductive machine learning algorithms [Thrun *et al.*, 1991]. They are discrete classification problems defined in an artificial "robot" domain, in which robots are described by six different attributes, adopted from [Wnek *et al.*, 1990]:

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:

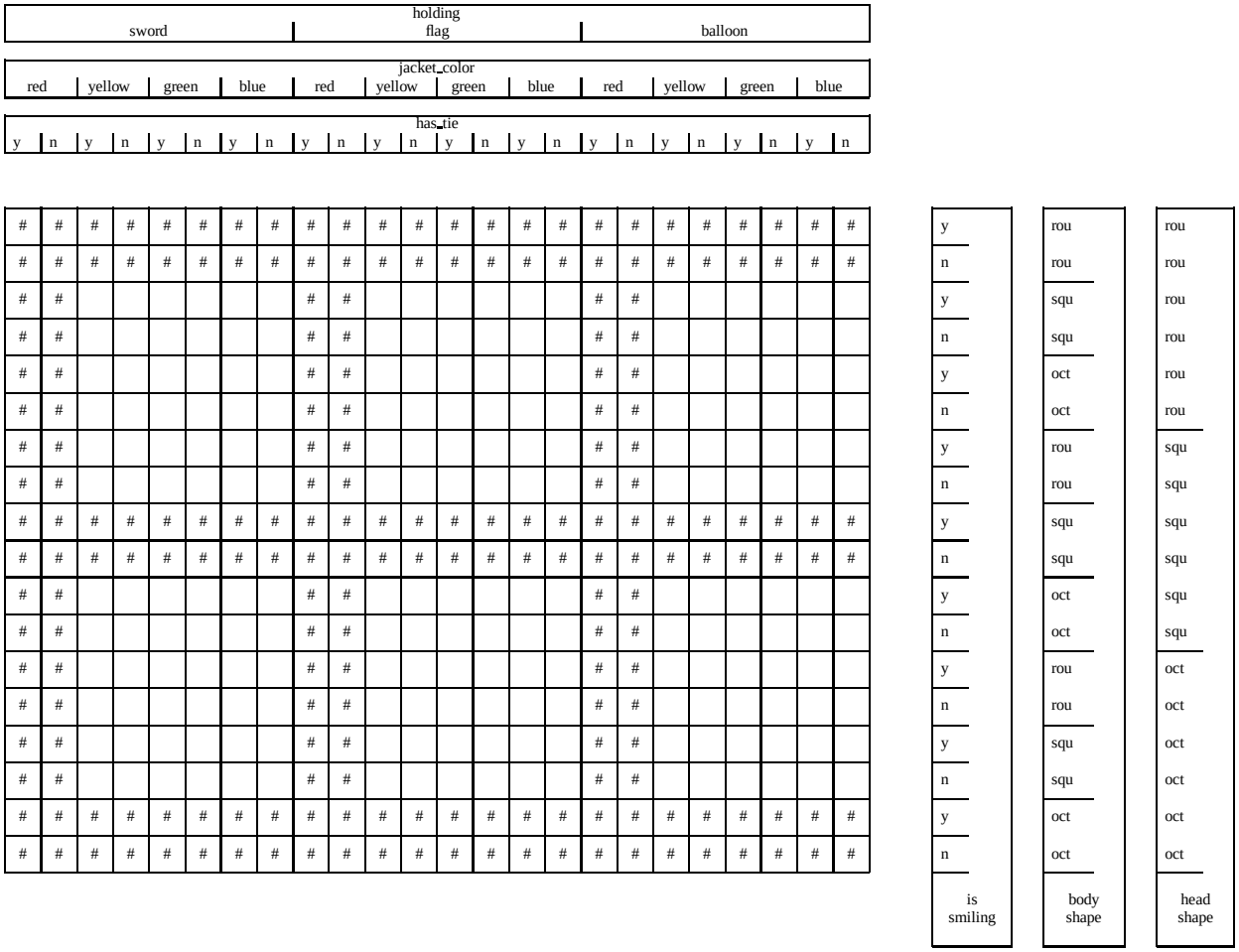
- **Problem M_1:**
  (`head_shape = body_shape`**) or (**`jacket_color = red`**)**
  Figure 9 shows the resulting classification and the weights and biases of the trained network.

- **Problem $\mathbf{M}_2$:**
  Exactly two of the six attributes have their *first* value.
  Figure 10 shows the resulting classification and the weights and biases of the trained network.

- **Problem $\mathbf{M}_3$:**
  (`jacket_color = green` **and** `holding = sword`**)**
  or (`jacket_color = not blue` **and** `body_shape = not octagon`**)**
  Figure 11 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 9, 10, and 11). 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. Hence, we applied VI-Analysis and breadth-first rule extraction to these networks. Note that these networks were not trained with the intention of easing their analysis—rather they represent ad hoc solutions to the MONK's classification problems.

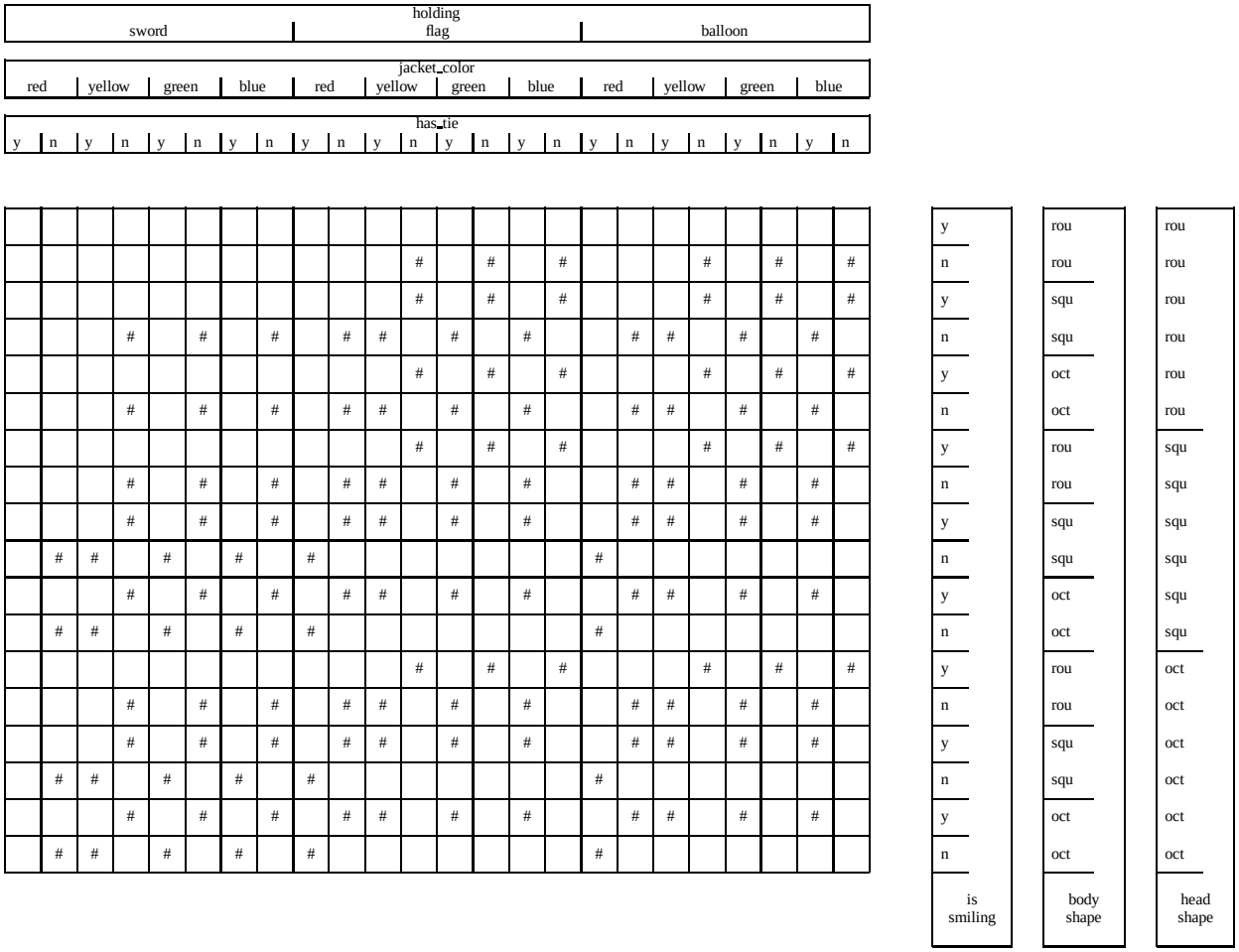
The following observation turned out to be essential for increasing the rule verification power of VI-Analysis in the context of the MONK's problems: The input coding of all networks was local, i.e., to each feature value we assigned a separate unit which was set to 1, if the corresponding feature was present, and 0 otherwise. Since different values of one and the same feature are mutually exclusive, exactly one (out of 2 to 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 $\mathcal{I}$. It is surprising that



MONK's problem $MONK_1$: 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

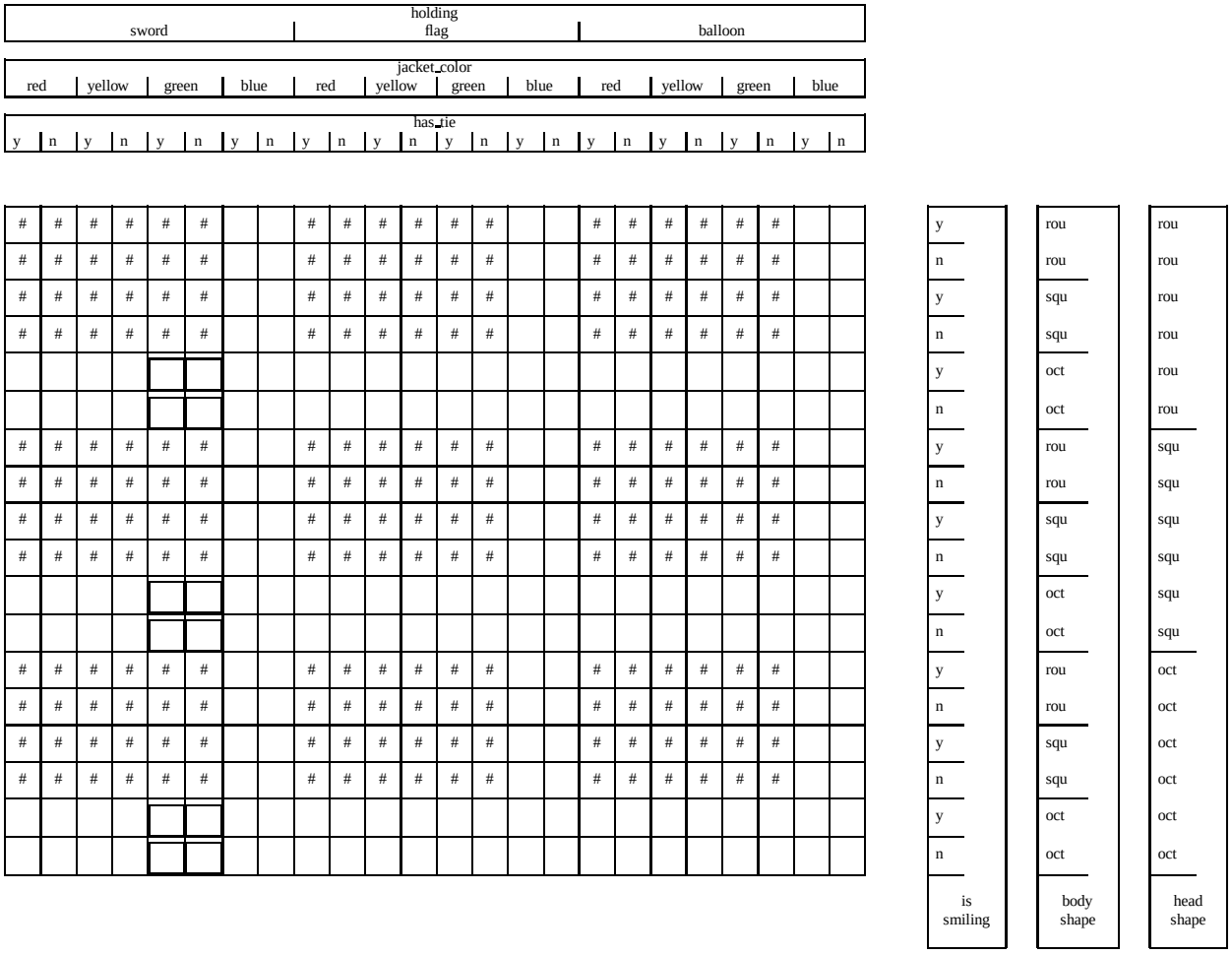
Figure 9: **MONK's problem $MONK_1$:** The network learned and generalized successfully.



MONK's problem $MONK_2$: weights and biases

	to-node		
from-node	hidden_1	hidden_2	output
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 10: **MONK's problem $MONK_2$:** The network learned and generalized successfully.



MONK's problem $MONK_3$: 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 11: **MONK's problem $MONK_3$:** Due to noise in the training set, the network had a small error in the classification, indicated by the boxes in the classification diagram.

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 $\mathbf{M}_1$

Figure 12: Rules extracted from the network trained on the MONK's problem $\mathbf{M}_1$.

without these additional input constraints VI-Analysis was unable to find reasonable rules at all.

The results of VI-Analysis for each of the three MONK's problems, respectively, are summarized here.

- **$\mathbf{M}_1$:** (c.f. Figure 9) The most general rules found by VI-Analysis are shown in Figure 12. As can be seen from this figure, VI-Analysis found a number of general rules for the target concept. It failed, however, in detecting the most general rules possible. For example, the attribute `body_shape` may only take on the values `round`, `square`, or `octagon`, and thus the first three rules depicted in Figure 12 can be combined yielding

  if `jacket_color` is `red` then $\mathbf{M}_1$.

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 $\mathbf{M}_3$

Figure 13: Rules extracted from the network trained on the MONK's problem $\mathbf{M}_3$.

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

(`head_shape` = `body_shape`) or (`jacket_color` = `red`)

- **$\mathbf{M}_3$:** (c.f. Figure 11). The rules generated by VI-Analysis for the $\mathbf{M}_3$ problem are listed in Figure 13. Here VI-Analysis succeeded in extracting the most general rules. The two negative rules depicted in this Figure suffice to describe the classification completely.

- **$\mathbf{M}_2$:** (c.f. Figure 10) As might be seen from the concept description of the problem $\mathbf{M}_2$, this problem is not representable in a compact set of hypercube-type rules at all. 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:

  - *The sum of all first feature values is ≤ 1.*
  - *The sum of all first feature values is 2.*
  - *The sum of all first feature values is ≥ 3.*

  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 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, it seems questionable as to whether this can be done efficiently. Such mechanisms are beyond the scope of this paper.

5.3 Growing Rules

We will now draw out attention to real-valued domains. While at least in principle finite domains allow for exhaustively testing all input patterns, real-valued domains are infinite and exhaustive search techniques cannot be applied. Therefore, one needs other techniques to form and verify rules.

In this section we will describe an approach to rule extraction that is based upon iteratively growing training instances. Assume a data point is given which is classified by the network as belonging to some class C. Obviously, VI-Analysis will easily prove the correctness of this classification, simply by constraining the input intervals to exactly this very data point. This is because VI-Analysis degrades to the standard forward propagation of activation values if each input interval contains a single point only, and the "proof" in this case is trivial.

Starting with this point, the input intervals can be iteratively extended to more general rules by adding small random values to the bounds of the input intervals. Each time one of the input intervals is enlarged by some random amount, VI-Analysis is applied to verify if this more general rule is still provably correct. If VI-Analysis succeeds, the larger input interval is maintained, otherwise it will be set to its previous values. This iterative growing procedure allows the gradual approximation of more general rules, based on a single input pattern which acts as a seed. If the network transfer functions are continuous, the almost-certain (i.e., with probability 1) existence and VIA-provability of small non-point rules can be shown. In our case we applied the technique of growing intervals to the XOR problem, and enlarged input intervals completely randomly. Therefore, for each starting point there might be a whole set of resulting rules, depending on the order in which the input intervals are modified. Figure 14 summarizes the result for the XOR example. Figure 14a displays the output of the XOR network over the two-dimensional hypercube spanned by the input units, and Figure 14b-d displays some rules found by random rule growing starting with the starting points $(1;0)$, $(0;1)$, $(0;0)$, $(1;1)$, and $(.5;.5)$.

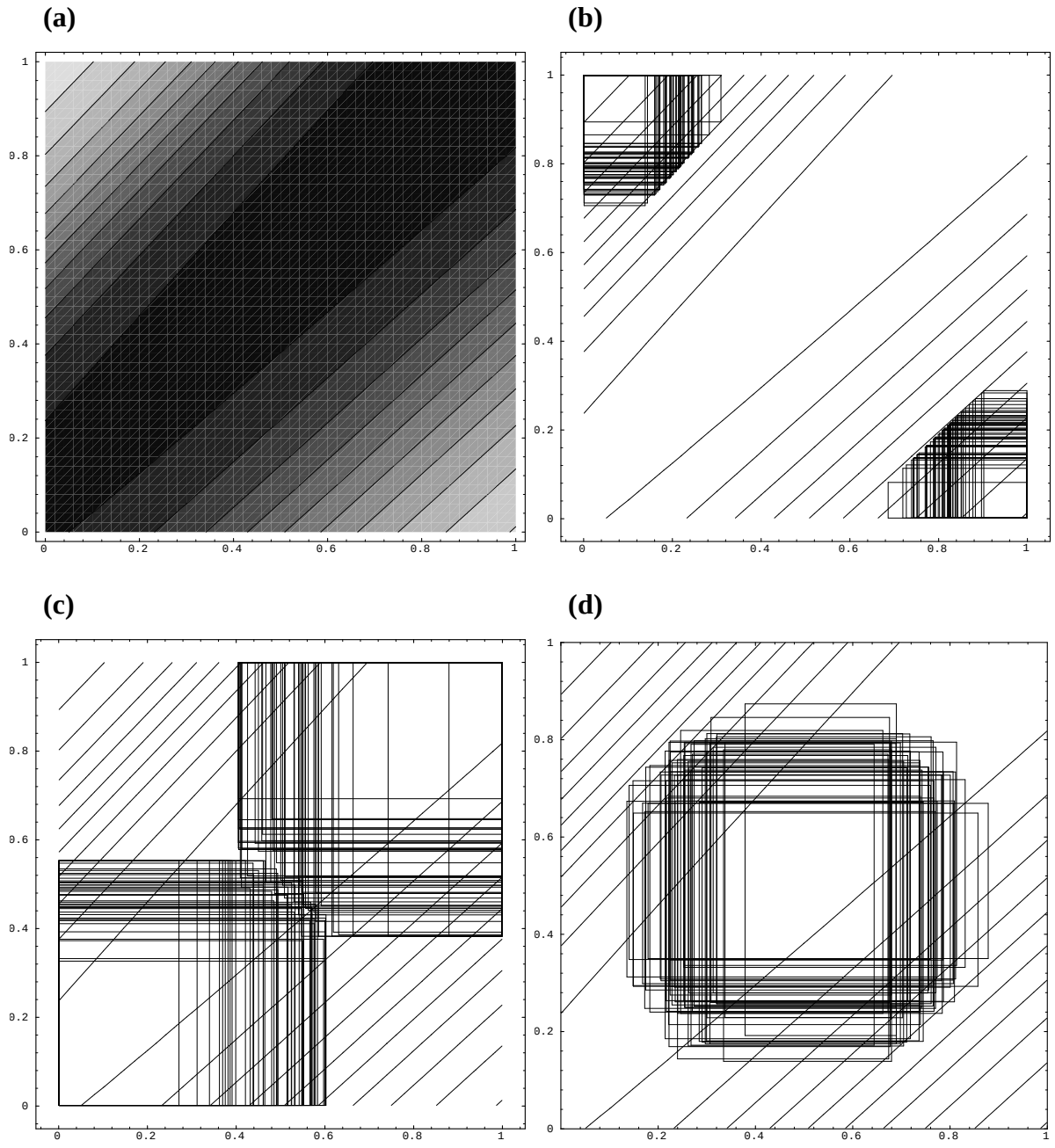


Figure 14: **Rules for XOR:** (a) shows the function generated by the network. The vertical and the horizontal axis measure the two input variables, and the gray-scale measures the network output (white=1, black=0). (b) Rules found from growing the two patterns $(0,0)$ and $(1,1)$. For these hypercubes the output of the network will always be greater than 0.5. (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.

Growing training data results in *legitimate preconditions* that ensure the same classification as the classification obtained by the training instance at hand. Hence the resulting preimage generalizes the training instance in the input space. It is worth mentioning that VI-Analysis bears some resemblance to symbolic explanation-based learning techniques (EBL) [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 feature 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 counterpart to weakest preconditions in symbolic EBL.

6 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. 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 layer-by-layer. Moreover, the rules found by VI-Analysis are provably correct, making VI-Analysis a promising candidate for larger networks with multiple hidden layers.

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. Note that the weight regularization term replaces 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.[10]

It should 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

[10]It seems feasible, although not necessarily computationally tractable, 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.

as certain structure on the network topology, might significantly improve the generalization rate, given that the target concept can be easily described by rules. Since we are interested in rule extraction from arbitrary networks we make no assumption about the training procedure at hand. Thus, 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.

7 Discussion

In this paper we have proposed a generic technique for extracting provably correct rules from arbitrary pre-trained artificial neural networks. The rule extraction mechanism relies on VI-Analysis, which is a tool for analyzing trained neural networks. VI-Analysis verifies the correctness of conjectured rules by searching inconsistencies. It does this by propagating and refining rule-knowledge (validity intervals) through the network in both forward and backward direction. We have demonstrated how VI-Analysis can be employed as a powerful proving-engine for the verification of symbolic rules. Two systematic rule search schemes, one for discrete domains and one for real-valued domains, are proposed and empirically evaluated on the XOR and the MONK's problems.

In the beginning of this paper we outlined four desired properties for a general rule extraction mechanism. VI-Analysis fulfills most of these demands:

1. **No architectural requirements.** Rule extraction using VI-Analysis does not make any architectural commitments whatsoever regarding the network to be analyzed. Indeed, by analyzing the network as a whole rather than compiling networks unit-by-unit, rules can be extracted from densely interconnected networks with arbitrary real-valued weights and biases. This includes recurrent networks not described here.[11] In its current form VI-Analysis relies exclusively on the assumption that all transfer functions are monotonic and continuous, as it the case in Backpropagation networks. As described below, VI-Analysis can be extended to piecewise monotonic transfer functions, which includes for example Radial-Basis functions [Moody and Darken, 1989].

2. **No training requirements.** Since VI-Analysis analyses trained networks, the rule

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

extraction mechanism described in this paper does not require any special training procedure. Consequently, VI-Analysis is applicable to a variety of networks. For example, many neural network applications that have proven to be successful in practice have not been trained to facilitate the extraction of rules. Examples include speech recognition [Waibel, 1989], speech synthesis [Sejnowski and Rosenberg, 1986], robot navigation [Pomerleau, 1989], handwritten digit recognition [LeCun *et al.*, 1990], medical diagnostics [Jabri *et al.*, 1992], and game playing [Tesauro, 1992]. Unlike rule extraction mechanisms which require a special training routine, VI-Analysis is generally applicable to a broad variety of artificial neural networks, including those listed above.

3. **Correctness.** The extracted rules are provably correct, i.e. rule extraction based on VI-Analysis generates rules that correctly describe the target network. The correctness of rules is a direct implication of the truth-preservation property of VI-Analysis.

4. **Expressive power.** Formally, the expressive power of the rule language of VI-Analysis is the set of linear constraints on the activation patterns for the input and the output layer. We have demonstrated that this language is sufficient for expressing hypercube 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 applied across several layers. VI-Analysis also excludes higher-order rules[12], 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. We conclude that we have partially met our goal of a powerful rule language. More general rule languages are clearly desirable.

There are several limitations and open questions that warrant future research:

- While VI-Analysis is truth-preserving, it might fail to prove the correctness of correct rules. This is because each weight layer is evaluated separately, neglecting dependencies that arise across multiple weight layers. By evaluating weight layer separately,

[12] rules with products of variables

techniques of linear programming become applicable. On the other hand, VI-Analysis might be too careful when refining intervals. It might miss existing contradictions. Non-linear optimization techniques which optimize the network as a whole are prospective candidates for overcoming this limitation. Since non-linear optimization usually suffers from local minima, it is generally unclear whether the resulting rules would still correctly describe the underlying network. Sacrificing the correctness goal, however, might be appropriate if the resulting rule verification algorithms turns out to be superior to VI-Analysis.

- So far, the space of strategies for generating rule conjectures has not yet been fully explored. In this paper we described two basic approaches, one for discrete and one for real-valued domains. The latter mechanism, for example, used a random search strategy to grow antecedents of a rule. There are a wide variety of more efficient strategies to grow input intervals in real-values domains. For example, one could start with identifying irrelevant features by removing whole input interval constraints. If the total volume of the rule is to be maximized, parallel search techniques such as Genetic Algorithms [Holland, 1984], [Goldberg, 1989] become applicable. 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.

  As Tom Dietterich [personal communication] pointed out, symbolic learning algorithms such as decision tree learning [Quinlan, 1986] may be used to generate rule hypotheses as well. 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 an artificial neural network trained on the same data.

- Rule verification based on VI-Analysis is complex, if the network at hand is large. This is because VI-Analysis refines the validity intervals iteratively, which involves many complete runs of the Simplex algorithm. In this paper we did not address computational efficiency at all. While for the simple cases we analyzed thus far, VI-Analysis was consistently found to terminate quickly, we suspect that VI-Analysis might be awfully slow when applied to networks two or more orders of magnitude larger. Research on speeding up VI-Analysis and other faster rule provers will then be warranted.

- One of the major strengths of the approach described in this paper is also one of

its weaknesses, namely the correctness of rules. While non-correct rule extraction mechanisms are less likely to scale to large networks with many hidden layers, the correctness of the extracted rules may lead to very specific rules. Generally speaking, if the function learned by the neural network is complex and highly cluttered, any correct rule extraction mechanism will be forced to generate a large set of highly specific rules. In such cases it might be more desirable to invoke mechanisms which trade off correctness versus generality and produce overly general rules. For such a mechanisms it is desirable that the human user has explicit control over this trade-off.

- Two important assumption of VI-Analysis in its current form 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. Algorithms for evaluating and propagating sets of intervals—rather than single intervals—are straightforward, but they come at the price of increased computational complexity. It remains to be shown whether such algorithms will be efficiently applicable in practice.

- VI-Analysis is a generic tool for analyzing dependencies in artificial neural networks. As such it can be applied to problems other than rule extraction. In this paper, for example, we exclusively focussed on constraining input and output activations. VI-Analysis allows hidden activations to be constrained as well. This might be useful for figuring out the role of hidden unit activations in the computation, in order to assign semantic meaning to hidden units. Satinder Singh [personal communication] has pointed out that validity intervals can also be assigned to weights and biases, characterizing the dependence of the network's output on the weights and biases of the network.

  These directions are currently completely unexplored, since our primary interest in VI-Analysis has been the automated extraction of preimages and rules.

Acknowledgment

I wish to thank Tom Dietterich, Clayton McMillan, and Tom Mitchell for their invaluable feedback that has influenced this research. I thank Clayton McMillan for his comments on an earlier draft of this paper. I also thank Armin Cremers for his steady support during the

course of this research. The "Deutsche Formschungsgemeinschaft" has in part supported this work by a travel grant.

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.

[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.

[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.

[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.

[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 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).

[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.