Knowledge Extraction from Trained Neural Networks: A Position Paper

A. S. d'Avila Garcez* K. Broda† D. M. Gabbay‡ A. F. de Souza§

Abstract— **It is commonly accepted that one of the main drawbacks of neural networks, the lack of explanation, may be ameliorated by the so called *rules' extraction* methods.**

We argue that neural networks encode *nonmonotonicity*, i.e., they jump to conclusions that might be withdrawn when new information is available.

In this paper, we present a new extraction method that complies with the above perspective. We define a *partial ordering* on the network's input vectors set, and use it to confine the search space for the extraction of rules by querying the network. We then define a number of simplification metarules, show that the extraction is sound and present the results of applying the extraction algorithm to the *Monks' Problems*.

I. Introduction

It is now commonly accepted that one of the main drawbacks of neural networks, the lack of explanation, may be ameliorated by the so called *rules' extraction* methods (see [1] for a comprehensive survey on the subject). The problem lies in the trade-off between the extraction algorithm's *complexity* and the *quality* of the set of rules extracted from the network. So far, decompositional methods have shown a better overall performance than pedagogical ones, when empirically tested in certain application domains. However, such methods as [7] and [9] rely on pruning of weights and retraining, and unfortunately may derive unsound rule sets (see [2]).

We propose a slight shift in perspective. We argue that neural networks are *nonmonotonic* systems, i.e., they jump to conclusions that might be withdrawn when new information is available [6]. In this sense, we derive rules of the form $L_1, ..., L_i, \sim L_{i+1}, ..., \sim L_j \rightarrow L_{j+1}$, where $\sim$ stands for *default negation*. Thus, operationally $a \sim b \rightarrow c$ behaves such that c fires in the presence of a provided that b is not present.

Moreover, one should only be able to derive $a \rightarrow c$ from a network with inputs a and b after it is known that $ab \rightarrow c$ and $a \sim b \rightarrow c$, otherwise the extraction may be unsound. Thus, we see metarules like *subsumption* [4] and *M-of-N* [9], which enhance considerably the readability of the rule set, as *simplifications* of the rules extracted from the network.

In this paper, we present a new extraction method that complies with the above perspective. We define the subnetworks of a network as is done for decompositional techniques. We show that there is a partial ordering on the set of input vectors of each subnetwork w.r.t the corresponding output's activation value. We then use the ordering to guide the input vectors search space during the extraction of rules, which is done by querying each subnetwork. We define the simplification metarules and show that they are strongly related to the ordering on the input vectors. Finally, we describe how to assemble the rule set for the network and show that the extraction algorithm is sound w.r.t exhaustive pedagogical extraction, i.e., that any rule extracted is actually encoded in the network.

In section 2 we describe the extraction algorithm, in section 3 we present the results of applying the algorithm in the Monks' problems [8], and in section 4 we conclude and discuss directions for future work.[1]

*Department of Computing, Imperial College, London SW7 2BZ, UK. aag@doc.ic.ac.uk.

†Department of Computing, Imperial College, London SW7 2BZ, UK. kb@doc.ic.ac.uk.

‡Department of Computer Science, King's College, London WC2R 2LS, UK. dg@dcs.kcl.ac.uk.

§Department of Computer Science, University College London, London WC1E 6BT, UK. A.Souza@cs.ucl.ac.uk.

[1]The extraction algorithm is part of the *Connectionist Inductive Learning and Logic Programming System* (CIL^2P). The interested reader is referred to [3] for a detailed description of the system, including the discussion about neural networks' nonmonotonic semantics, and to [2]

II. The Extraction Algorithm

Throughout this paper the truth-values $True$ and $False$ are represented by 1 and -1, respectively. We assume that each input value $i_i \in \{-1, 1\}$, and each input vector $\mathbf{i}$ is associated with an *interpretation* for the rule set. For example, let $\mathcal{I}$ be the set of input neurons. Suppose $\mathcal{I} = \{a, b, c\}$. We fix a linear ordering on the symbols of $\mathcal{I}$ and represent it as a list, say $[a, b, c]$. Thus if $\mathbf{i} = (1, -1, 1)$ then $\mathbf{i}(a) = \mathbf{i}(c) = True$ and $\mathbf{i}(b) = False$. The set of input vectors $\mathbf{I}$ ($\mathbf{i} \in \mathbf{I}$) is an abstract representation for the power set of the set of input neurons $\mathcal{I}$, i.e., $\mathbf{I} = \wp(\mathcal{I})$. For example, $\mathbf{i} = (1, -1, 1)$ above represents the set $\{a, c\}$. We consider the class of single hidden layer networks without loss of generality [5].

We define the extraction problem as follows: given a trained network, find for each input vector $\mathbf{i}$, all the outputs o_j in the corresponding output vector $\mathbf{o}$ such that $o_j > A_{min}$, where $A_{min} \in (0, 1)$ (we say that output neuron j is *active* for $\mathbf{i}$ *iff* $o_j > A_{min}$). For example, given the set of input neurons $\mathcal{I} = [a, b, c]$, if j is *active* for $\mathbf{i} = (1, -1, 1)$ then derive the rule $a \sim bc \rightarrow j$. If $\mathbf{i}$ has length p then there are 2^p possible input vectors to be queried in the network.

Hence, let $\delta_{\mathcal{N}} : \mathbf{i} \rightarrow \mathbf{o}$ be the function computed by network $\mathcal{N}$, if we query the network for all 2^p input vectors and derive rules for each pair $(\mathbf{i}, \mathbf{o})$ then we obtain a sound and complete rule set. Clearly, the problem lies in the fact that computing such a rule set may turn out to be impossible for large input vectors. Moreover, even for the case of computable sets, the rules' readability may be extremely poor. In order to ameliorate the first problem, we try and find a *partial ordering*[2] on the set of input vectors $\mathbf{I}$ such that we can guide the search for rules' extraction. To cope with the second problem, the simplification metarules are brought to bear. A network that computes the $\overline{\text{XOR}}$ function, called N (Figure 1), will be used to exemplify the technique.

Notation: $o_j(\mathbf{i}_m)$ will denote the activation of output neuron j given input vector $\mathbf{i}_m$ and $\delta_{\mathcal{N}} : \mathbf{i} \rightarrow \mathbf{o}$, where $\mathcal{N}$ depends on the context. In Figure 1 for example, for network N, o_j refers to the activation of neuron o, while for (sub)network N_1, o_j refers to the activation of neuron h_0.

We start by defining a *distance function* between input vectors and the *sum* of an input vector.

Definition 1 *Let $\mathbf{i}_m$ and $\mathbf{i}_n$ be two input vectors in $\mathbf{I}$. The distance $dist(\mathbf{i}_m, \mathbf{i}_n)$ between $\mathbf{i}_m$ and $\mathbf{i}_n$ is the number of inputs i_i for which $\mathbf{i}_m(i_i) \neq \mathbf{i}_n(i_i)$.*

for the full version of this paper and for the proofs of the propositions.

[2] A *partial ordering* is a reflexive, transitive and antisymmetric relation on a set.

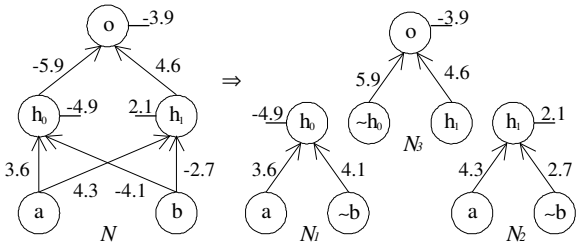

Figure 1 – The network N, having *tanh* as activation function, computes $\overline{XOR}$. We will extract rules for h_0, h_1 and o by querying N_1, N_2 and N_3, respectively, and then assemble the rule set for N.

Definition 2 *Let $\mathbf{i}_m$ be a $p-$ary input vector in $\mathbf{I}$. The sum $\langle \mathbf{i}_m \rangle$ of $\mathbf{i}_m$ is the sum of all input elements i_i in $\mathbf{i}_m$, i.e., $\langle \mathbf{i}_m \rangle = \sum_{i=1}^{p} \mathbf{i}_m(i_i)$.*

Now we define the partial ordering $\leq_{\mathbf{I}}$ on $\mathbf{I} = \wp(\mathcal{I})$ w.r.t set inclusion. We say that $\mathbf{i}_m \subseteq \mathbf{i}_n$ if the set represented by $\mathbf{i}_m$ is a subset of the set represented by $\mathbf{i}_n$.

Definition 3 *Let $\mathbf{i}_m$ and $\mathbf{i}_n$ be input vectors in $\mathbf{I}$. $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_n$ iff $\mathbf{i}_m \subseteq \mathbf{i}_n$.*

Clearly, for a finite set $\mathcal{I}$, $\mathbf{I}$ is a finite partially ordered set w.r.t $\leq_{\mathbf{I}}$ having $\mathcal{I}$ as its maximum element and the empty set $\emptyset$ as its minimum element. The following proposition shows that $\leq_{\mathbf{I}}$ is an ordering of interest w.r.t the network's outputs for networks with positive weights only.

Proposition 4 *Let $\mathbf{o}$ be a $r-$ary vector. If $\forall kl, W_{lk} \in \Re^+$ then $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_n$ implies $o_j(\mathbf{i}_m) \leq o_j(\mathbf{i}_n)$, for all $1 \leq j \leq r$.*

By Proposition 4, we know for instance that if output j is active for $\mathbf{i}_m$ then it is also active for $\mathbf{i}_n$. However, $W_{lk} \in \Re^+$ is a very strong condition. In order to relax it, we need to split the network into *subnetworks*, similarly to [4], such that a variation of Proposition 4 will hold for $W_{lk} \in \Re$. A network with p input neurons, q hidden neurons and r output neurons contains q *input-to-hidden subnetworks*, each with p inputs and a single output, and r *hidden-to-output subnetworks*, each with q inputs and a single output (see Figure 1). To each subnetwork we apply a transformation whereby we rename input neurons x_k linked through negative weights to the output, by $\sim x_k$ and replace each weight $W_{lk} \in \Re$ by its modulus. We call the result the *positive form* of the subnetwork. For example, in Figure 1, N_1 and N_2 are the positive forms of the input-to-hidden subnetworks of N, and N_3 is the positive form of the hidden-to-output subnetwork of N. More precisely, we define the function σ mapping input vectors of the positive form into input vectors of the subnetwork as follows. Let $x_k \in \mathcal{I}$,

$1 \leq k \leq s$. $\sigma([x_1,...,x_s](i_1,...,i_s)) = (i'_1,...,i'_s)$, where $i'_k = i_k$ if x_k is a positive literal and $i'_k = -i_k$ if x_k is a negative literal. For example, for N_1 $\sigma([a, \sim b](1,1)) = (1,-1)$.

The following proposition shows that $\leq_{\mathbf{I}}$ is still valid for subnetworks and $W_{lk} \in \Re$.

Proposition 5 *For each subnetwork of a network,* $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_n$ *implies* $o_j(\sigma(\mathcal{I}, \mathbf{i}_m)) \leq o_j(\sigma(\mathcal{I}, \mathbf{i}_n))$.

Figure 2 shows $\leq_{\mathbf{I}}$ on the input vectors of N_1, where $(1,1) = [a, \sim b]$, and the mapping σ to the input vectors of the corresponding subnetwork of N.

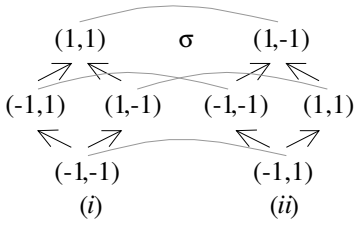

Figure 2 – (i) $\leq_{\mathbf{I}}$ on the input vectors of N_1 and (ii) the ordering on the input vectors of the corresponding subnetwork of N.

If now, in addition, we consider the weights values of each positive form, we can decide whether $o_j(\mathbf{i}_m) \leq o_j(\mathbf{i}_n)$ when $\langle \mathbf{i}_m \rangle = \langle \mathbf{i}_n \rangle$ [3]. Taking for instance $\mathbf{i}_m = (1,-1)$ and $\mathbf{i}_n = (-1,1)$ for N_1, since $W_{h_0 a} \leq W_{h_0 \sim b}$ it is not difficult to see that $h_0(\mathbf{i}_m) \leq h_0(\mathbf{i}_n)$. Let us formalize this idea.

Definition 6 *Let* $\mathbf{i}_m, \mathbf{i}_n$ *and* $\mathbf{i}_o$ *be three different input vectors in* $\mathbf{I}$ *such that* $dist(\mathbf{i}_m, \mathbf{i}_o) = 1$, $dist(\mathbf{i}_n, \mathbf{i}_o) = 1$ *and* $\langle \mathbf{i}_o \rangle < \langle \mathbf{i}_m \rangle, \langle \mathbf{i}_n \rangle$, *i.e.,* $\mathbf{i}_m$ *and* $\mathbf{i}_n$ *are immediate* successors *of* $\mathbf{i}_o$. *Let* $\mathbf{i}_m$ *be obtained from* $\mathbf{i}_o$ *by flipping the i-th input from -1 to 1, while* $\mathbf{i}_n$ *is obtained from* $\mathbf{i}_o$ *by flipping the k-th input from -1 to 1. We write* $\mathbf{i}_m \leq_{\langle\rangle} \mathbf{i}_n$ *iff* $W_{ji} \leq W_{jk}$.

Proposition 7 *For each subnetwork of a network,* $\mathbf{i}_m \leq_{\langle\rangle} \mathbf{i}_n$ *implies* $o_j(\sigma(\mathcal{I}, \mathbf{i}_m)) \leq o_j(\sigma(\mathcal{I}, \mathbf{i}_n))$.

We may now define the ordering $\preceq$ on the input vectors of a subnetwork's positive form w.r.t $\leq_{\mathbf{I}}$ and $\leq_{\langle\rangle}$ as follows.

Definition 8 *Let* $\preceq$ *be a partial ordering on* $\mathbf{I}$. *For all* $\mathbf{i}_m, \mathbf{i}_n \in \mathbf{I}$, $\mathbf{i}_m \preceq \mathbf{i}_n$ *iff* $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_n$ *or* $\mathbf{i}_m \leq_{\langle\rangle} \mathbf{i}_n$.

Proposition 9 *For each subnetwork of a network,* $\mathbf{i}_m \preceq \mathbf{i}_n$ *implies* $o_j(\sigma(\mathcal{I}, \mathbf{i}_m)) \leq o_j(\sigma(\mathcal{I}, \mathbf{i}_n))$.

[3] Recall that, previously, two input vectors $\mathbf{i}_m$ and $\mathbf{i}_n$ such that $\langle \mathbf{i}_m \rangle = \langle \mathbf{i}_n \rangle$ were incomparable, e.g., $(-1,1)$ and $(1,-1)$ at Figure 2(i).

Corollary 10 *(*Search Space Pruning Rule*) Let* $\mathbf{i}_m$ *and* $\mathbf{i}_n$ *be input vectors of the positive form of a subnetwork with output neuron* j, *such that* $\mathbf{i}_m \preceq \mathbf{i}_n$. *If* $\mathbf{i}_n$ *does not activate* j *then* $\mathbf{i}_m$ *does not activate* j *either. By contraposition, if* $\mathbf{i}_m$ *activates* j *then* $\mathbf{i}_n$ *also does.*

Definition 8 ($\preceq$), together with Corollary 10, provides a systematic way of searching the input vectors space (see [2] for implementation details).

Example 11 *Figure 3 shows* $\preceq$ *on the positive form of a subnetwork with three input neurons, say* $\{a, b, c\}$, *and output neuron* j. *Let* $W_{ja} = 5$, $W_{jb} = 2$, *and* $W_{jc} = 1$ *and* $(1,1,1) = [\sim a, b, c]$. *By Corollary 10, if for instance* $(1,1,-1)$ *activates* j *then* $(1,1,1)$ *also does (see Figure 3), and we can derive the rules* $\sim ab \sim c \rightarrow j$ *and* $\sim abc \rightarrow j$, *respectively. Similarly, if in addition* $(1,-1,1)$ *does not activate* j *then we can stop the search.*

Basically, the extraction algorithm queries each subnetwork's positive form, alternating from both the ordering's maximum and minimum elements and following the partial ordering $\preceq$ *on the input vectors, it generates rules accordingly (by using* σ*) until it reaches the frontier between vectors that activate the output and vectors that do not do so, according to the Search Space Pruning Rule.*

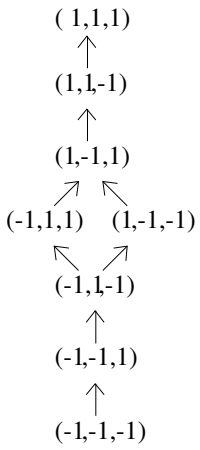

Figure 3 – The ordering $\preceq$ on *3-ary* input vectors [x_1,x_2,x_3], where $W_{jx_1} \geq W_{jx_2} \geq W_{jx_3}$.

Note that for $2-ary$ input vectors, $\preceq$ is a linear ordering. For N_1 $(-1,-1) \preceq (1,-1) \preceq (-1,1) \preceq (1,1)$ and for N_2 $(-1,-1) \preceq (-1,1) \preceq (1,-1) \preceq (1,1)$, where $(1,1) = [a, \sim b]$ for both. For N_1, h_0 is active for $(1,1)$ only. Thus, by applying σ we derive $a \sim b \rightarrow h_0$. For N_2, h_1 is not active for $(-1,-1)$ only. Similarly, we derive $ab \rightarrow h_1$, $\sim a \sim b \rightarrow h_1$ and $a \sim b \rightarrow h_1$. The last two rules can be simplified to obtain $\sim b \rightarrow h_1$, since $\sim b$ implies h_1 given either a or $\sim a$. Similarly, from

$ab \rightarrow \mathrm{h}_1$ and $a \sim b \rightarrow \mathrm{h}_1$ we obtain $a \rightarrow \mathrm{h}_1$. Let us then define the simplification metarules of the extraction algorithm.

Definition 12 *(*Subsumption*) A rule r_1 subsumes a rule r_2 iff they have the same conclusion and the set of premises of r_1 is a subset of the set of premises of r_2.*

Definition 13 *(*Complementary Literals*) Let $r_1 = L_1, ..., L_i, ..., L_j \rightarrow L_{j+1}$ and $r_2 = L_1, ..., \sim L_i, ..., L_j \rightarrow L_{j+1}$ be derived rules. Then, $r_3 = L_1, ..., L_{i-1}, L_{i+1}, ..., L_j \rightarrow L_{j+1}$ is also a derived rule. Note that r_3 subsumes r_1 and r_2.*

Definition 14 *(*M of N*) Let $m, n \in \aleph, \mathcal{I}\prime \subseteq \mathcal{I}, |\mathcal{I}\prime| = n, m \leq n$. Then, if any combination of m elements chosen from $\mathcal{I}\prime$ implies L_{j+1} we derive a rule of the form $m(\mathcal{I}\prime) \rightarrow L_{j+1}$.*

Definition 15 *(*M of N Subsumption*) Let $m, p \in \aleph, \mathcal{I}\prime \subseteq \mathcal{I}, m(\mathcal{I}\prime) \rightarrow L_{j+1}$ subsumes $p(\mathcal{I}\prime) \rightarrow L_{j+1}$ iff $m < p$.*

Each simplification metarule is tied to the ordering $\preceq$ on the input vectors, e.g., *Complementary Literals* are applied over vectors with distance one between themselves, and valid *M of N* rules are applied over *up-sets*[4]. Taking, for the sake of simplicity, $(1, 1, 1) = [a, b, c]$ in the ordering of Figure 3, if $(1, -1, -1)$ activates j then, by Corollary 10 and Definitions 12 and 13, we obtain the rule $a \rightarrow j$. Similarly, if $(-1, 1, 1)$ activates j then, by using Definition 14 as well, we derive the rule $2(abc) \rightarrow j$. Finally, by Definition 15, $1(abc) \rightarrow j$, obtained when $(-1, -1, 1)$ activates j, subsumes $2(abc) \rightarrow j$, and so on. The following proposition shows that the simplification metarules preserve equivalence of rule sets.

Proposition 16 *Let S and S' be rule sets. If S' is obtained from S by applying Definitions 12, 13, 14 or 15 then S and S' are logically equivalent.*

So far, we have deliberately not mentioned the extraction from *hidden-to-output subnetworks*. Briefly, all the propositions above hold for such subnetworks, if we assume that the network's hidden neurons present discrete values activations such as $\{-1, 1\}$. We know however that this is not the case, and therefore accuracy problems may arise from such assumption (see [1]). At this point we need to compromise. Either we assume that the hidden neurons activations are in $\{-1, A_{min}\}$, and then are able to show that the extraction is sound but incomplete, or we assume that it is in $\{-A_{min}, 1\}$, obtaining an unsound but complete extraction. We have chosen the first approach[5].

Back to the $\overline{\mathrm{XOR}}$ example, for N_3 we have $(-1, -1) \preceq (-1, A_{min}) \preceq (A_{min}, -1) \preceq (A_{min}, A_{min})$, where $(A_{min}, A_{min}) = [\sim \mathrm{h}_0, \mathrm{h}_1]$ and $A_{min} = 0.5$. Only (A_{min}, A_{min}) activates o, and we derive the rule $\sim \mathrm{h}_0 \mathrm{h}_1 \rightarrow \mathrm{o}$.

In what follows, we outline the extraction algorithm for a network $\mathcal{N}$ and present a simple but illustrative example. The interested reader is referred to [2] for the detailed presentation of the extraction algorithm. For each subnetwork $\mathcal{N}_i$ of $\mathcal{N}$ we do:

1. Apply the *Transformation* σ on $\mathcal{N}_i$ and obtain $\mathcal{N}_i$'s *positive form* ($\mathcal{N}_i^+$).
2. Fix a linear ordering on $\mathcal{I}$ according to the weights vector of $\mathcal{N}_i^+$.
3. Query $\mathcal{N}_i^+$ with input vector $inf(\mathbf{I})$, where $inf(\mathbf{I})$ is the minimum element of the ordering $\preceq$ on $\mathbf{I}$. If $o_j > A_{min}$, derive the rule $\rightarrow j$ and stop (j is a $fact$, by Definition 13).
4. Query $\mathcal{N}_i^+$ with input vector $sup(\mathbf{I})$, where $sup(\mathbf{I})$ is the maximum element of the ordering $\preceq$ on $\mathbf{I}$. If $o_j < -A_{min}$, stop.
5. *Search the input vectors space* $\mathbf{I}$: Set $\mathbf{i}_\perp := inf(\mathbf{I})$, $\mathbf{i}_\top := sup(\mathbf{I})$. While $dist(\mathbf{i}_\perp, inf(\mathbf{I})) \leq n\mathrm{DIV}2$ or $dist(\mathbf{i}_\top, sup(\mathbf{I})) \leq n\mathrm{DIV}2 + n\mathrm{MOD}2$, where n is the number of input neurons of $\mathcal{N}_i^+$, do:

   Generate new $\mathbf{i}_\perp$ and $\mathbf{i}_\top$ from old $\mathbf{i}_\perp$ and $\mathbf{i}_\top$, respectively, following the ordering $\preceq$ on $\mathbf{I}$, and query the network

   (a) set new $\mathbf{i}_\perp$:= old $\mathbf{i}_\perp$, according to $\preceq$. Query $\mathcal{N}_i^+$ with new $\mathbf{i}_\perp$.

   (b) If the *Search Space Pruning Rule* is applicable: stop generating the successors of $\mathbf{i}_\perp$, apply *Complementary Literals* and add the rules derived accordingly to the rule set.

   (c) set new $\mathbf{i}_\top$:= old $\mathbf{i}_\top$, according to $\preceq$. Query $\mathcal{N}_i^+$ with new $\mathbf{i}_\top$.

   (d) If the *Search Space Pruning Rule* is applicable: stop generating the predecessors of $\mathbf{i}_\top$.
6. Apply *Subsumption* to the rule set.

[4] Let P be an ordered set and $Q \subseteq P$. Q is an *up-set* if, whenever $x \in Q$, $y \in P$ and $x \preceq y$, then $y \in Q$.

[5] Here, we perform a kind of worst case analysis. By choosing activations in $\{-1, A_{min}\}$, misclassifications occur because of the absence of a rule (incompleteness). Analogously, by choosing $\{-A_{min}, 1\}$, misclassifications are due to the inappropriate presence of rules in the rule set (unsoundness). In this context, the choice of $\{-1, 1\}$ yields unsound and incomplete rule sets.

7. Apply *M of N* and *M of N Subsumption* to the rule set.

Example 17 *(*Exactly 1 of 5*) We train a network with five input neurons* $\{a, b, c, d, e\}$*, two hidden neurons* $\{h_0, h_1\}$ *and one output neuron* $\{o\}$*, on all the 32 possible input vectors. The network's output fires iff* exactly *1 of its inputs fires. Although this is a very simple network, it is not very easy to verify by inspecting its weights that it computes* exactly *1 out of* $\{a, b, c, d, e\}$*. Assume that* $|W_{h_0 d}| \leq |W_{h_0 e}| \leq |W_{h_0 c}| \leq |W_{h_0 a}| \leq |W_{h_0 b}|$ *and* $|W_{h_1 d}| \leq |W_{h_1 e}| \leq |W_{h_1 a}| \leq |W_{h_1 c}| \leq |W_{h_1 b}|$*. We split the network into its subnetworks and apply the extraction algorithm, i.e., we query each positive form, by following the ordering* $\preceq$ *on* $\mathbf{I}$*, until we reach the frontier of activations. Taking* $\mathcal{I} = [a, b, c, d, e]$ *for the subnetwork with output* h_0*, suppose we find that input* $(-1, -1, -1, 1, -1)$ *activates* h_0*. Since* $|W_{h_0 d}|$ *is the smallest weight, from the ordering* $\preceq$ *on* $\mathbf{I}$ *and by applying Definitions 13 and 14, we derive the rule* $1(abcde) \rightarrow h_0$*. Note that, by Definition 12, this rule subsumes* $m(abcde) \rightarrow h_0$*, for* $m > 1$*. Taking again* $\mathcal{I} = [a, b, c, d, e]$ *for the subnetwork with output* h_1*, suppose we find that input* $(-1, -1, -1, 1, 1)$ *activates* h_1*. Similarly, from the ordering* $\preceq$ *on* $\mathbf{I}$ *and by applying Definitions 13 and 14, we derive the rule* $2(abcde) \rightarrow h_1$*. Finally, for the hidden-to-output subnetwork,* $\mathcal{I} = [h_0, \sim h_1]$*. Taking* $A_{min} = 0.5$*, o is only activated by* (A_{min}, A_{min}) *and we derive the rule* $h_0 \sim h_1 \rightarrow o$*. Therefore,* exactly 1 *out of* $\{a, b, c, d, e\}$ *is obtained by computing* $1(abcde) \wedge \sim 2(abcde) \rightarrow o$*, i.e.,* at least 1 *out of* $\{a, b, c, d, e\}$ AND at most 1 *out of* $\{a, b, c, d, e\}$ *implies o.*

The last step of the extraction algorithm is to assemble the rule set of $\mathcal{N}$. In the above example, in order to obtain the rule mapping inputs $\{a, b, c, d, e\}$ directly into the output $\{o\}$, we assumed that $1(abcde) \leftrightarrow h_1$ and $2(abcde) \leftrightarrow h_2$. Lemma 18 below guarantees that we can do so, i.e., we take the *completion* of each rule extracted from input-to-hidden subnetworks. For example, for network N (Figure 1) we had $a \sim b \rightarrow \mathrm{h_0}$, $a \rightarrow \mathrm{h_1}$, $\sim b \rightarrow \mathrm{h_1}$ and $\sim \mathrm{h_0 h_1} \rightarrow \mathrm{o}$. And from $a \sim b \leftrightarrow \mathrm{h_0}$ and $a \vee \sim b \leftrightarrow \mathrm{h_1}$ we finally obtain $(\sim a \vee b) \wedge (a \vee \sim b) \rightarrow \mathrm{o}$; the $\overline{\mathrm{XOR}}$ function.

Lemma 18 *The extraction of rules from input-to-hidden subnetworks is sound and complete.*

Lemma 19 *The extraction of rules from hidden-to-output subnetworks is sound.*

Finally, the proof that the extraction algorithm is sound follows from Lemmas 18 and 19.

Theorem 20 *The extraction algorithm is sound.*

III. Experimental Results

As a point of departure for testing, we applied the extraction algorithm[6] to the Monks' problems [8], which have been used as benchmark for performance comparison between a range of symbolic and connectionist machine learning systems. Briefly, in the Monks' problems, robots in an artificial domain are described by six attributes with the following possible values: *head-shape*{round, square, octagon}, *body_shape*{round, square, octagon}, *is_smiling*{yes, no}, *holding*{sword, balloon, flag}, *jacket_color*{red, yellow, green, blue} and *has_tie*{yes, no}. Problem 1 trains a network with 124 examples, selected from 432, where $head\_shape = body\_shape \vee jacket\_color = red$. Problem 2 trains a network with 169 examples, selected from 432, where *exactly two of the six attributes have their first value.* Problem 3 trains a network with 122 examples with 5% noise, selected from 432, where $(jacket\_color = green \wedge holding = sword) \vee (jacket\_color \neq blue \wedge body\_shape \neq octagon)$. The remaining examples are used in the respective test sets.

We use the same architectures as Thrun [8], i.e., single hidden layer networks with three, two and four hidden neurons, for Problems 1, 2 and 3, respectively; 17 input neurons, one for each attribute value, and a single output neuron, for the binary classification task. We use the standard backpropagation learning algorithm. Differently from Thrun, we use bipolar activation function, inputs in the set $\{-1, 1\}$, and $A_{min} = 0$ (See [3] for the motivation behind this).

For Problems 1, 2 and 3, the networks' test set performance was 100%, 100% and 93.1%, and the rule sets performance in the same test sets was 100%, 99.2% and 91.9%, respectively. In general, less than 30% of the input vectors set is queried and, among them, less than 50% generate rules. The tables below present, for Problems 1, 2, and 3, the number of input vectors queried during extraction and the number of rules obtained before and after simplifications *Complementary Literals* and *Subsumption* are applied. For example, for hidden neuron $\mathrm{h_0}$ in Monk's Problem 1, 18,724 input vectors are queried generating 9,455 rules that after simplification are reduced to 2,633 rules.

Monks1	Input Vectors	Queried	Extracted	Simplified
$\mathrm{h_0}$	131072	18724	9455	2633
$\mathrm{h_1}$	131072	18598	9385	536
$\mathrm{h_2}$	131072	42776	21526	1793
o	8	8	2	1

[6]The extraction algorithm was implemented in ANSI C.

Monks2	Input Vectors	Queried	Extracted	Simplified
h_0	131072	131070	58317	18521
h_1	131072	43246	21769	5171
o	4	4	1	1

Monks3	Input Vectors	Queried	Extracted	Simplified
h_0	131072	18780	9240	3311
h_1	131072	18618	9498	794
h_2	131072	43278	21282	3989
h_3	131072	18466	9544	1026
o	16	14	8	2

In general, *Complementary Literal* and *Subsumption* reduce the rule set by 80%. *M of N* and *M of N Subsumption* further enhance the rule set readability. In particular, the rule set for Problem 1 is presented in the Appendix[7]. For short, we name each attribute value with a letter from a to q in the sequence presented above, s.t. $a = (head\_shape = round), b = (head\_shape = square)$, and so on.

It is interesting that because the rule obtained for the hidden-to-output subnetwork of Problem 1 was $\sim h_1 \sim h_2 \rightarrow o$ and since the rule set presents 100% of accuracy, hidden neuron h_0 is not necessary at all, i.e., the problem could have been solved by a network with two hidden neurons only, obtaining the same results. Another interesting exercise is to try and see what the network has generalized, given the rule set and the classification task learned.

IV. Work in Progress and Conclusion

The next step is to test the extraction algorithm on real world applications such as Computational Biology, where the number of input neurons make an exhaustive pedagogical extraction impossible, and compare results with those obtained in [4, 7, 9].

We have formally presented a new algorithm that extracts (sound) nonmonotonic rules from neural networks. Its application in the Monks' Problems shows very promising results.

References

[1] R. Andrews, J. Diederich and A. B. Tickle, "Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks", Knowledge-based Systems, 8(6):373-389, 1995.

[2] A. S. d'Avila Garcez, K. Broda and D. Gabbay, "Symbolic Knowledge Extraction from Trained Neural Networks: A New Approach", Technical Report TR-98-014, Department of Computing, Imperial College, London, 1998.

[3] A. S. d'Avila Garcez and G. Zaverucha, "The Connectionist Inductive Learning and Logic Programming System", In F. Kurfess (ed.) Applied Intelligence Journal, Special Issue on Neural Networks and Structured Knowledge (to appear), 1999.

[4] L. Fu, "Neural Networks in Computer Intelligence", McGraw Hill, 1994.

[5] K. Hornik, M. Stinchcombe and H. White, "Multilayer Feedforward Networks are Universal Approximators", Neural Networks, 2:359-366, 1989.

[6] W. Marek and M. Truszczynski, "Nonmonotonic Logic: Context Dependent Reasoning", Springer-Verlag, 1993.

[7] R. Setiono, "Extracting Rules from Neural Networks by Pruning and Hidden-unit Splitting", Neural Computation 9:205-225, 1997.

[8] S. B. Thrun et al., "The MONK's Problem: A Performance Comparison of Different Learning Algorithms", Technical Report, Carnegie Mellon University, CMU-CS-91-197, 1991.

[9] G. G. Towell and J. W. Shavlik, "The Extraction of Refined Rules From Knowledge Based Neural Networks", Machine Learning, 13(1):71-101, 1993.

Appendix

Rules extracted for the Monk's Problem 1:

$\sim h_1 \sim h_2 \rightarrow o$

$\sim abcd \sim e \rightarrow h_1$

$bd \sim e \sim l \rightarrow h_1$

$b \sim i \sim lmn \rightarrow h_1$

$bcd(\sim l \vee \sim ef) \rightarrow h_1$

$b \sim ef(mn \vee mo) \rightarrow h_1$

$\sim abdf(\sim l \vee m \vee n) \rightarrow h_1$

$mno(\sim l \vee b \sim e \vee d \sim e \vee bc \vee cd \vee \sim ab \vee bf) \rightarrow h_1$

$1(mno) \wedge (bd \sim e \vee bd \sim l \vee bcdf \vee b \sim ef \sim l \vee \sim abcd \vee \sim ab \sim e \sim l \vee bc \sim e \sim l \vee cd \sim e \sim l) \rightarrow h_1$

$a \sim b \sim dek \sim l \rightarrow h_2$

$ac \sim dem \sim q \rightarrow h_2$

$a \sim b \sim def \sim l \rightarrow h_2$

$ae \sim gjm(n \vee o) \rightarrow h_2$

$\sim be \sim g \sim ln(a \vee \sim d) \rightarrow h_2$

$a \sim b \sim de \sim l(c \vee \sim h) \rightarrow h_2$

$\sim b \sim de \sim g \sim l(m \vee o) \rightarrow h_2$

$a \sim b \sim de \sim l(j \vee p \vee i) \rightarrow h_2$

$a \sim be \sim l \sim q(\sim d \vee m) \rightarrow h_2$

$a \sim be \sim g \sim l(\sim d \vee m \vee o) \rightarrow h_2$

$aem(\sim gn \sim p \vee \sim go \sim p \vee \sim hkn \vee \sim hko) \rightarrow h_2$

$1(mno) \wedge (a \sim de \sim h \vee a \sim de \sim g \vee a \sim de \sim l \vee a \sim b \sim de \vee ac \sim def \vee a \sim b \sim df \sim l \vee \sim b \sim def \sim l \vee a \sim bef \sim l \vee a \sim b \sim d \sim g \sim l \vee a \sim be \sim h \sim l \vee a \sim b \sim d \sim h \sim l \vee \sim b \sim de \sim h \sim l \vee a \sim bce \sim l \vee a \sim bc \sim d \sim l \vee \sim bc \sim de \sim l) \rightarrow h_2$

[7] We use the *Integrity Constraints* of the Monks' Problems in order to present the rule set here. For example, we do not present derived rules where $has\_tie = yes$ and $has\_tie = no$ simultaneously.