Symbolic knowledge extraction from trained neural networks: A sound approach

ARTICLE *in* ARTIFICIAL INTELLIGENCE · JANUARY 2001

Impact Factor: 3.37 · DOI: 10.1016/S0004-3702(00)00077-1 · Source: DBLP

Artur D'Avila Garcez
City University London

K. Broda
Imperial College London

Symbolic Knowledge Extraction from Trained Neural Networks: A New Approach

A. S. d'Avila Garcez[‡] K. Broda[‡] D. M. Gabbay[§]

[‡]Department of Computing
Imperial College, London, SW7 2BZ
{aag,kb}@doc.ic.ac.uk

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

Abstract

Although neural networks have shown very good performance in many application domains, one of their main drawbacks lies in the incapacity to provide an explanation for the underlying reasoning mechanisms.

The "explanation capability" of neural networks can be achieved by the extraction of symbolic knowledge. In this paper, we present a sound new method of extraction which captures nonmonotonic rules encoded in the network.

We start by discussing some of the main problems of knowledge extraction methods. To ameliorate these problems, a partial ordering on the space of input vectors is defined, as well as a number of pruning and simplification rules. The pruning rules are then used in our extraction algorithm to reduce the search space of input vectors during a pedagogical extraction, whereas the simplification rules are used to reduce the size of the extracted set of rules. We show that, in the case of regular networks, the extraction algorithm is sound and complete.

We proceed to extend the extraction algorithm to the class of non-regular networks, the general case. We show that non-regular networks always contain regularities in their subnetworks. As a result, the underlying extraction method for regular networks can be applied, but now in a decompositional fashion. In order to combine the sets of rules extracted from each subnetwork into the final set of rules, we use a method whereby we are able to keep the soundness of the extraction algorithm.

Finally, we present the results of an empirical analysis of the extraction system, using traditional examples and real-world application problems. The results have shown that a very high fidelity between the extracted set of rules and the network can be achieved.

1 Introduction

Human cognition successfully integrates the connectionist and symbolic paradigms of Artificial Intelligence (AI). Yet, the modelling of cognition develops these separately in neural computation and symbolic logic/AI areas. There is now a movement towards a fruitful midway in between these extremes, in which the study of logic is combined with recent insights from connectionism. It is essential that these be integrated [24].

The aim of neural-symbolic integration is to explore the advantages that each paradigm presents. Within the features of artificial neural networks are massive parallelism, inductive learning and generalisation capabilities [5, 15]. On the other hand, symbolic systems can explain their inference process, e.g., through automatic theorem proving, and use powerful declarative languages for knowledge representation [19].

The Connectionist Inductive Learning and Logic Programming (CIL^2P) system [4] is a proposal towards tightly coupled neural-symbolic integration [16]. CIL^2P is a massively parallel computational model based on a feedforward artificial neural network that integrates inductive learning from examples and background knowledge [20] with deductive learning from Logic Programming [22]. Starting with the background knowledge represented by a (propositional) general or extended logic program, a translation algorithm (see Figure 1, (1)) is applied generating a neural network that can be trained with examples (2). Moreover, the neural network computes the stable model (answer set) of the general (extended) program inserted in it or learned by examples, as a parallel system for Logic Programming (3). The final stage of the system (4) consists of the symbolic knowledge extraction from the trained neural network, which provides the explanation for the network's answers. The knowledge extracted then could feed the system again (5), closing the learning cycle[1].

In this paper, we concentrate on the problem of extraction of symbolic knowledge from trained neural networks, that is, the problem of finding "logical representations" for such networks. The extraction allows for the explanation of the decision making process, thus contributing to solve the "knowledge acquisition bottleneck problem". The domain theory extracted, obtained from inductive learning with examples, can be added to an existing knowledge base or used in the solution of analogous domains problems.

Briefly, the problem of extraction lies on the complexity of the extraction algorithm. Holldobler and Kalinke [17] have shown that each logic program is equivalent to a single hidden layer neural network. In one direction of that equivalence relation, a translation algorithm (see Figure 1(1)) derives a neat neural network structure when a logic program is given. The problem arises

[1] For example, in a fault diagnosis system, a neural network can detect a fault quickly, triggering safety procedures, while the knowledge extracted from it can justify the fault later on. If mistaken, that information can be used to fine tune the learning system.

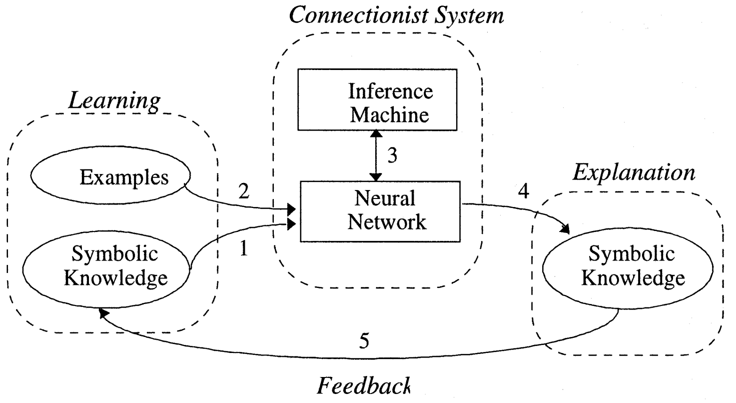


Figure 1: Neural-Symbolic Integration

in the converse direction, i.e., given a trained neural network, how could we find out the equivalent logic program? Unfortunately, it is very unlikely that a neat network will result from the learning process. Furthermore, a typical real-world application network may contain hundreds of input neurons and thousands of connections.

The knowledge acquired by a neural network during its training phase is encoded as: (i) the network's architecture itself; (ii) the activation function associated to it; and (iii) the value of its weights. As pointed out in [1], the task of extracting explanations from trained neural networks is the one of interpreting in a comprehensible form the collective effect of (i), (ii), and (iii). Also in [1], a classification scheme for extraction algorithms is given, based on: (a) the expressive power of the extracted rules; (b) the "translucency" of the network; (c) the quality of the extracted rules; and (d) the algorithmic complexity. The first classification item refers to the symbolic knowledge presented to the user from the extraction process. In general, this knowledge is represented by rules of the form "if then else". The second classification item contains two basic categories: decompositional and pedagogical. In the *decompositional*, the extraction occurs at the level of individual, hidden and output, units within the trained neural network. In the *pedagogical*, the neural network is viewed as a "black box", and the extraction is done by mapping inputs directly into outputs. The next classification item intends to measure how well the task of extracting the rules has been performed, considering the accuracy, consistency and comprehensibility of the set of rules. The last item refers to the requirement for the algorithm to be as effective as possible. In this sense, a crucial issue in developing an extraction algorithm is how to constrain the size of the solution space to be searched.

In [37], Thrun defines the following desirable properties of an extraction algorithm: (i) No architectural requirements: a general extraction mechanism should be able to operate with all types of neural networks; (ii) No training requirements: the algorithm should not make assumptions about the way the network has been built and how its weights and biases have been learned; (iii) Correctness: the extracted rules should describe the underlying network as correctly as possible; (iv) High expressive power: more powerful languages and more compact rule sets are highly desirable.

Intuitively, the extraction task is to find the relations between input and output concepts in a trained network, in the sense that certain inputs *cause* a particular output. We argue that neural networks are nonmonotonic systems, i.e., they jump to conclusions that might be withdrawn when new information is available [23]. Thus, the set of rules extracted may contain default negation ($\sim$). Each neuron can represent a concept or its "classical" negation ($\neg$). Consequently, we expect to extract a set of rules of the form: $L_1, ..., L_n, \sim L_{n+1}, ..., \sim L_m \rightarrow L_{m+1}$, where each L_i is a literal (a propositional variable or its "classical" negation), L_j $(1 \leq j \leq m)$ represents a neuron in the network's input layer, L_{m+1} represents a neuron in the network's output layer, $\sim$ stands for default negation, and $\rightarrow$ means causal implication[2] (see [4] for neural network's nonmonotonic semantics).

In this paper, we present a new approach for knowledge extraction from trained networks that complies with the above perspective. We start by discussing some of the main problems found in the literature. To ameliorate these problems, we identify a partial ordering on the input vectors space, and define a number of pruning rules and simplification rules that interact with such an ordering. These rules are used in our extraction algorithm to reduce the search space of input vectors, as well as the number of rules extracted. We show that, in the case of regular networks, the extraction algorithm is sound and complete[3]. We then extend the extraction algorithm to the general case. By showing that every non regular network contains regularities in its subnetworks, we can still apply the underlying extraction algorithm to the general case network, but now in a decompositional fashion. The only problem we have to tackle, however, is how to combine the sets of rules obtained from each subnetwork into the set of rules of the network. We use a method for assembling the set of rules whereby we are able to preserve soundness of the extraction algorithm, although we have to forego completeness.

In section 2 we discuss the main problems of the task of extracting knowledge from trained networks. In section 3 we recall some useful preliminary concepts and define the extraction problem precisely. In section 4 we present our solution to the extraction problem, culminating with the outline of the extraction algorithm for the class of regular networks. In section 5 we extend

[2] Notice that this is the language of Extended Logic Programming [13].

[3] Following [12], we say that an extraction algorithm is sound and complete if the set of rules is equivalent to the network's generalization set. If, however, the set of rules is a subset of the generalization set, then the extraction is sound but incomplete.

the extraction algorithm to the class of non regular networks, the general case. In section 6 we present the experimental results of applying the extraction system to the Monk's Problems [36], DNA sequence analysis and Power Systems fault diagnosis. Finally, in section 7 we conclude and discuss directions for future work.

2 Related Work

Among the existing extraction methods, the one presented in [17], the "Ruleneg" [30], the "VIAnalysis" algorithm [37], and the "Rule-Extraction-as-Learning" method [7] use "pedagogical" approaches, while the "Subset" [12], the "MofN" [38], the "Rulex" [2] and Setiono's proposal [34, 35] are "decompositional" methods (see [1] for a comprehensive survey).

In the CIL^2P system, after learning takes place, the network N encodes a knowledge P' that contains the background knowledge P complemented or even revised by the knowledge learned with training examples. We want to derive P' from N. At the moment, only pedagogical approaches can guarantee that the knowledge extracted is equivalent to the network, i.e., that the extraction process is sound and complete. In [17], for instance, all possible combinations of the input vector $\mathbf{i}$ of N are taken into account in the process of rule generation. In this way, the method must consider 2^n different input vectors, where n is the number of neurons in the input layer of N. Some pedagogical approaches reduce the input vectors space by extracting rules for the learning set only, excluding the network's generalisation.

Obviously, pedagogical approaches are not effective when the size of the neural network increases, as in real-world problems applications. In order to overcome this limitation, decompositional methods, in general, apply heuristically guided searches to the process of extraction. The "Subset" method [12], for instance, attempts to search for subsets of weights of each neuron in the hidden and output layers of N, such that the neurons' input potential exceeds its threshold. Each subset that satisfies the above condition is written as a rule. One of the most interesting decompositional methods is the "MofN" technique [38]. Based on the Subset method, it reduces the search space of the neural network by clustering and pruning weights. It also generates a smaller number of rules, by using the following representation: *If m of $(A_1, ..., A_n)$ are "true" then A is "true"*, where $m \leq n$. The work by Setiono [34, 35] is another proposal of decompositional extraction. Setiono proposes a penalty function for pruning a feedforward neural network, and then generates rules from the pruned network by considering a small number of activation values at the hidden units.

Decompositional methods, such as [38] and [35], in general use weights pruning mechanisms prior to extraction (notice the difference between pruning the input vectors search space and pruning the networks weights). However, there is no guarantee that a pruned network will be equivalent to the original one. That is the reason why these methods usually require retraining the network. During retraining, some restrictions must be imposed on the learning process - for instance, allowing only the thresholds, but not the weights, to change - in order to the network to keep its "well behaved" pruned structure. At this point, there is no guarantee that retraining will be successful under such restrictions. In addition, methods that use a penalty function are bound to restrict the network's learning capability[4]. Even if we avoid the use of penalty functions and weights' clustering and pruning, the simple task of decomposing the network into smaller subnetworks, from which rules are extracted and then assembled, has to be carried out carefully. That is because, in general, the collective effect of the network is different from the effect of the superposition of its parts [1]. As a result, most decompositional methods are unsound. The following example illustrates this fact.

Example 1 *Consider the network of Figure 2. Let us assume that the weights are such that* $\mathbf{i} = (1, 1)$ *neither activates* n_1 *nor* n_2*, but that the composition of the activations of* n_1 *and* n_2 *activates* x*. For example, suppose that* $a = 1$ *and* $b = 1$ *imply* $n_1 = 0.3$ *and* $n_2 = 0.4$*, and that these activation values imply* $x = 0.99$*. A decompositional method would most probably derive a unique rule,* $n_1, n_2 \rightarrow x$*, not being able to establish the correct relation between* a*,* b *and* x*. Consider, now, the case where* $\mathbf{i} = (1, 1)$

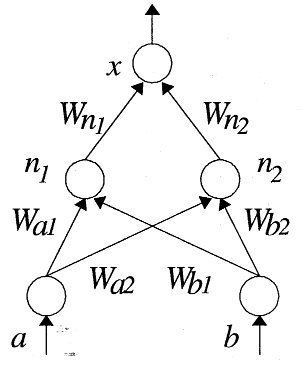


Figure 2: A simple example of unsoundness and incompleteness of some decompositional extraction algorithms.

activates n_1 *and* n_2*, but* n_1 *and* n_2 *do not activate* x*. Moreover, if* n_1 *and* n_2 *are approximated as threshold units, i.e., assumed either totally activated or non activated, then* $n_1 = 1$ *and* $n_2 = 1$ *activate* x*. For example, the weights*

[4] For example, the extraction algorithm can not be applied on a network trained with an "*off the shelf*" learning algorithm.

could be such that $a = 1$ and $b = 1$ imply $n_1 = 0.7$ and $n_2 = 0.8$; $n_1 = 0.7$ and $n_2 = 0.8$ do not imply x (let's say $x < 0.5$), but $n_1 = 1$ and $n_2 = 1$ do imply x (say, $x > 0.5$). As a result, a decompositional method, e.g. [38], would conclude that $ab \rightarrow x$ when actually $ab \nrightarrow x$.

The first case is an example of incompleteness. The second one shows how decompositional methods may turn out to be unsound. Even Fu's extraction [12], which is sound w.r.t each hidden and output neuron, may become unsound w.r.t the whole network.

Clearly, there is a trade-off between the *complexity* of the extraction method and the *quality* of the knowledge extracted from the network. In our view, an alternative is to prune the input vectors space, instead of the weight vectors space. Our goal is to reduce complexity by applying the extraction algorithm on a smaller solution space, yet maintaining the highest possible quality, in particular to maintain soundness.

Differently from the above approaches, we also want to capture nonmonotonic rules encoded in the network. In order to do so, we add negation by default ($\sim$) to the language. We argue that one cannot derive a sensible set of rules from a network without having $\sim$ in the language, as the following example illustrates.

Example 2 *Consider a neural network with two input neurons a and b, one hidden neuron n_1, and one output neuron x, such that $W_{n_1a} = 5$, $W_{n_1b} = -5$ and $W_{xn_1} = 1$. Assume that input vector $\mathbf{i}_1 = (1, 0)$ activates x. One would derive the rule $a \rightarrow x$. As a result, one would be able to conclude that $ab \rightarrow x$, since this rule is subsumed by the rule previously derived. However, as in the network above, it may be the case that input vector $\mathbf{i}_2 = (1, 1)$ does not activate x. In this case, one would conclude that $ab \nrightarrow x$, a contradiction. The correct rule to be extracted in the first place is, therefore, $a \sim b \rightarrow x$, which means that x fires in the presence of a provided that b is not present, and in fact, if b turns out to be true then the conclusion of x is overruled.*[5] *As a result, in order to conclude that $a \rightarrow x$, first we need to ensure that both $ab \rightarrow x$ and $a \sim b \rightarrow x$. "Classical" negation ($\neg$), stronger than $\sim$ in the sense that a literal is proved $false$, should be explicitly represented in the network by a neuron labelled $\neg x$ [3], as we will exemplify later in section 6.*

Summarizing, the novelties on this paper are: we present an eclectic approach whereby we can reduce the complexity of the extraction algorithm in some interesting cases, yet executing a sound extraction; and we capture nonmonotonicity in the set of rules extracted from the network, by adding default negation to the language.

[5] The use of *zero* as input was misleading, and that is one of the reasons why we use $\{-1, 1\}$ as input (see [4]).

3 Preliminaries

3.1 General

We need to assert some basic assumptions that will be used throughout this paper (see [9], [14] and [29]). $\aleph$ and $\Re$ denote the sets of natural and real numbers, respectively.

Definition 3 *A* partial order *is a reflexive, transitive and antisymmetric relation on a set.*

Definition 4 *A binary relation $\leq$ on a set X is* total *if for every $x, y \in X$, either $x \leq y$ or $y \leq x$.*

As usual, $x < y$ abbreviates $x \leq y$ and $y \not\leq x$.

Definition 5 *In a partially ordered set $[X, \leq]$, x is the* immediate predecessor *of y if $x \leq y$ and there is no other element z in X for which $x \leq z \leq y$. The inverse relation is called the* immediate successor.

Definition 6 *Let X be a set and $\leq$ an ordering on X. Let $x \in X$.*

x is minimal *if there is no element $y \in X$ such that $y < x$.*

x is a minimum *if for all elements $y \in X, x \leq y$. If $\leq$ is also antisymmetric and such an x exists, then x is unique and will be denoted by $inf(X)$.*

x is maximal *if there is no element $y \in X$ such that $x < y$.*

x is a maximum *if for all elements $y \in X, y \leq x$. If $\leq$ is also antisymmetric and such an x exists, then x is unique and will be denoted by $sup(X)$.*

A maximum (minimum) element is also maximal (minimal) but is, in addition, comparable to every other element. This property and antisymmetry leads directly to the demonstration of the uniqueness of $inf(X)$ and $sup(X)$.

Definition 7 *Let $[X, \leq]$ be a partially ordered set. For any $x, y \in X$, the* least upper bound *of x and y is the element z such that $x \leq z$ and $y \leq z$, and if there is any element z^* with $x \leq z^*$ and $y \leq z^*$ then $z \leq z^*$. The* greatest lower bound *of x and y is an element w such that $w \leq x$ and $w \leq y$, and if there is any element w^* with $w^* \leq x$ and $w^* \leq y$ then $w^* \leq w$.*

Definition 8 *A* lattice *is a partially ordered set in which every two elements x and y have a least upper bound, denoted by $x + y$, and a greatest lower bound, denoted by $x \cdot y$. A lattice L is* distributive *if $x + (y \cdot z) = (x + y) \cdot (x + z)$ and $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.*

Definition 9 *Let U be a set and $f : U \times U \rightarrow \Re$ a function satisfying the following conditions:*

(I) $f(x,y) \geq 0$,

(II) $f(x,y) = 0 \leftrightarrow x = y$,

(III) $f(x,y) = f(y,x)$,

(IV) $f(x,y) \leq f(x,z) + f(z,y)$.

f is called a *metric* on U[6]. A *metric space* is a tuple $\langle U, f\rangle$. A metric f on U is *bounded* iff for some constant $k \in \Re$, $f(x,y) \leq k$, for all $x, y \in U$.

3.2 Neural Networks

Hornik, Stinchcombe and White [18] have proved that standard feedforward neural networks with a single hidden layer are capable of approximating any (Borel) measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. Thus, we concentrate on single hidden layer networks, without loss of generality.

Given a single hidden layer feedforward network, the following systems of equations describe it.

$$\begin{aligned} n_1 &= h(W^1_{11}i_1 + W^1_{12}i_2 + \cdots + W^1_{1p}i_p - \theta_{n_1}) \\ n_2 &= h(W^1_{21}i_1 + W^1_{22}i_2 + \cdots + W^1_{2p}i_p - \theta_{n_2}) \\ &\vdots \\ n_r &= h(W^1_{r1}i_1 + W^1_{r2}i_2 + \cdots + W^1_{rp}i_p - \theta_{n_r}) \end{aligned} \tag{1}$$

$$\begin{aligned} o_1 &= h(W^2_{11}n_1 + W^2_{12}n_2 + \cdots + W^2_{1r}n_r - \theta_{o_1}) \\ o_2 &= h(W^2_{21}n_1 + W^2_{22}n_2 + \cdots + W^2_{2r}n_r - \theta_{o_2}) \\ &\vdots \\ o_q &= h(W^2_{q1}n_1 + W^2_{q2}n_2 + \cdots + W^2_{qr}n_r - \theta_{o_q}) \end{aligned} \tag{2}$$

where $\mathbf{i} = (i_1, i_2, ..., i_p)$ is the network's input vector ($i_{j(1\leq j\leq p)} \in [-1,1]$), $\mathbf{o} = (o_1, o_2, ..., o_q)$ is the network's output vector ($o_{j(1\leq j\leq q)} \in [-1,1]$), $\mathbf{n} = (n_1, n_2, ..., n_r)$ is the hidden layer vector ($n_{j(1\leq j\leq r)} \in [-1,1]$), $\theta_{n_j(1\leq j\leq r)}$ is the j-th hidden neuron threshold ($\theta_{n_j} \in \Re$), $\theta_{o_j(1\leq j\leq q)}$ is the j-th output neuron threshold ($\theta_{o_j} \in \Re$), $-\theta_{n_j}$ (resp. $-\theta_{o_j}$) is called the bias of the j-th hidden neuron (resp. output neuron), $W^1_{ij(1\leq i\leq r, 1\leq j\leq p)}$ is the weight of the connection from the j-th neuron in the input layer to the i-th neuron in the hidden layer ($W^1_{ij} \in \Re$), $W^2_{ij(1\leq i\leq q, 1\leq j\leq r)}$ is the weight of the connection

[6] f is sometimes called a *distance function*.

from the j-th neuron in the hidden layer to the i-th neuron in the output layer ($W^2_{ij} \in \Re$), and finally $h(x) = \frac{2}{1+e^{-\beta x}} - 1$ is the standard bipolar (semi-linear) activation function. Notice that for each output $o_j (1 \leq j \leq q)$ in $\mathbf{o}$ we have $o_j = h(\sum_{i=1}^{r}(W^2_{ji}.h(\sum_{k=1}^{p}(W^1_{ik}.i_k) - \theta_{n_i})) - \theta_{o_j})$.[7]

We define the extraction problem as follows: *Given a particular set of weights W_{ij} and biases θ_i, resulting from a training process on a neural 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 $A_{min} \in (0,1)$ is a predefined value* (we say that output neuron j is "active" for input vector $\mathbf{i}$ iff $o_j > A_{min}$).

We assume that for each input i_j in the input vector $\mathbf{i}$, either $i_j = 1$ or $i_j = -1$. That is done because we associate each input (and output) neuron with a concept, say a, and $i_j = 1$ means that a is *true* while $i_j = -1$ means that a is *false*. For example, consider a network with input neurons a and b. If $\mathbf{i} = (1, -1)$ activates the output neuron j then we derive the rule $a \sim b \rightarrow j$. As a result, if the input vector $\mathbf{i}$ has length p there are 2^p possible input vectors to be checked.

4 The Extraction Algorithm for Regular Networks

Having identified the problems of knowledge extraction from trained networks, let us now start working towards the outline of their solutions. Given the above extraction problem definition, firstly we realize that each output neuron j has a constraint associated. We want to find $o_j = h(\sum_{i=1}^{r}(W^2_{ji}n_i) - \theta_{o_j})$ *such that* $o_j > A_{min}$. We can equivalently define the extraction problem as follows. Let $\mathbf{I}$ be the set of input vectors and $\mathbf{O}$ be the set of output vectors. We define a binary relation ξ on $\mathbf{I} \times \mathbf{O}$ such that $\mathbf{o}\xi\mathbf{i} \leftrightarrow \mathbf{o} = \delta(\mathbf{i})$, and the extraction problem reduces to: for each o_j in $\mathbf{o} \in \mathbf{O}$, find the set $\mathbf{I}' \subseteq \mathbf{I}$ of input vectors $\mathbf{i}$ such that $o_j > A_{min}$.

Considering the monotonically crescent characteristic of the activation function $h(x)$ and given that $0 < A_{min} < 1$ and $\beta > 0$, we can rewrite $h(x) > A_{min}$ as $x > h^{-1}(A_{min})$. As a result, notice that to satisfy the above constraint $o_j > A_{min}$, it is required that $x = \sum_{i=1}^{r}(W^2_{ji}n_i) - \theta_{o_j} > 0$. Hence, each output o_j is determined by the system of equations 1 and equation 3 below, given in terms of the hidden neurons' activation values.[8]

$$j \textit{ is true iff } W^2_{j1}n_1 + W^2_{j2}n_2 + \cdots + W^2_{jr}n_r > h^{-1}(A_{min}) + \theta_{o_j} \qquad (3)$$

[7] Whenever it is not necessary to differentiate between hidden and output layer, we refer to the weights in the network as W_{ij} only. Similarly, we refer to the network's thresholds in general as θ_i only.

[8] Given $h(x) = \frac{2}{1+e^{-\beta x}} - 1$, we obtain $h^{-1}(x) = -\frac{1}{\beta}\ln\left(\frac{1-x}{1+x}\right)$. We use the bipolar semi-linear activation function for convenience; any monotonically crescent activation function could have been used here.

4.1 Positive Networks

We start by considering a very simple network where all weights are positive real numbers. In other words, each W_{ij} in the system of equations 1 and in equation 3 is positive. Obviously, given two input vectors $\mathbf{i}_m$ and $\mathbf{i}_n$, if $\forall i_{(1 \leq i \leq r)} n_i(\mathbf{i}_m) > n_i(\mathbf{i}_n)$ then $\forall j_{(1 \leq j \leq q)} o_j(n_i(\mathbf{i}_m)) > o_j(n_i(\mathbf{i}_n))$. Moreover, if $\mathbf{i}_m = (1, 1, ..., 1)$ each n_i is maximum and, therefore, each o_j is maximum. Similarly, if $\mathbf{i}_n = (-1, -1, ..., -1)$ then each n_i is minimum and each o_j is minimum. That results also from the monotonically crescent characteristic of the activation function $h(x)$, as we will see in detail later. Let us first present a simple example to help clarify the above ideas.

Example 10 *Consider the network and its constraint representation of Figure 3. We know that* $n_1 = h(W_a.a + W_b.b - \theta_{n_1})$. *Since* $W_a, W_b > 0$, *it is*

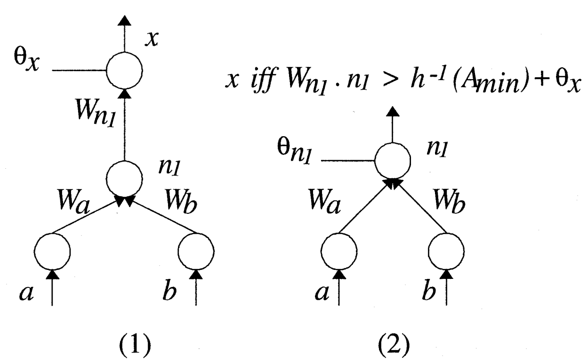


Figure 3: A single hidden neuron network (1) and its constraint representation (2) w.r.t. output x. $W_a, W_b, W_{n_1} \in \Re^+$.

easy to verify that the ordering of Figure 4 on the set of input vectors $\mathbf{I}$ *holds w.r.t the output* (x). *The ordering says, for instance, that the activation of*

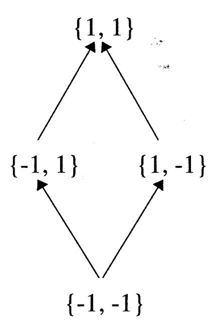


Figure 4: Ordering on the input vectors set $\mathbf{I}$ of the network (1) of figure 6.

n_1 is maximum if $\mathbf{i} = (1,1)$, that $n_1(1,1) \geq n_1(1,-1)$, and that n_1 is minimum if $\mathbf{i} = (-1,-1)$. Since $W_{n_1} > 0$, the activation of o_x is also maximum if $\mathbf{i} = (1,1)$, $o_x(1,1) \geq o_x(1,-1)$, and o_x is minimum if $\mathbf{i} = (-1,-1)$. The output x is therefore governed by the ordering.

Given the ordering, we can draw some conclusions. If the minimum element is given as the network's input (representing $\sim a \wedge \sim b$) and it satisfies the constraint over x (that is, $\sim a \wedge \sim b \rightarrow x$) then any other element in the ordering will satisfy it as well. In this case, since all possible input vectors are in the ordering, we can conclude that x is a fact ($\rightarrow x$). If, on the other hand, the maximum element ($a \wedge b$) does not satisfy x then no other element in the ordering will satisfy it. Notice that if it is the case that both $(1,1)$ (representing $a \wedge b$) and $(1,-1)$ (representing $a \wedge \sim b$) satisfy x but no other element in the ordering does, we can conclude that $a \rightarrow x$. Similarly, if $(1,1)$ and $(-1,1)$ are the only elements satisfying x we conclude that $b \rightarrow x$, regardless of a.

We have identified, therefore, that if $\forall ij, W_{ij} \in \Re^+$ it is easy to find an ordering on the set of input vectors $\mathbf{I}$ w.r.t the set of output vectors $\mathbf{O}$. Such information can be very useful to guide a pedagogical extraction procedure of symbolic knowledge from the network. The ordering can help prune the search space of input vectors, so that we avoid checking irrelevant input vectors safely, in the sense that those vectors that are not checked would not generate new rules. Moreover, each rule obtained is sound because the extraction is done by querying the actual network.

Notice that in the worst case we still have to check 2^n input vectors, and in the best case we only need to check one input vector (either the minimum or the maximum element in the ordering). Note also that there is, actually, a linear order on the set of input vectors, which however may be impossible to be found without querying each input vector for a particular set of weights. Thus, we will focus initially on the analysis of a group of networks where an ordering can be easily found. The following example illustrates what we mean by "an ordering easily found".

Example 11 *Consider the network of Figure 5. If we know, for instance, that $W_{a1} > W_{b1}$ then we can derive the ordering of Figure 6(1) on the set of input vectors w.r.t the activation value of neuron n_1. In the same way, if we know that $W_{a2} = W_{b2}$ then we can derive the ordering 6(2) w.r.t. n_2. If $W_{a1} > W_{b1}$ and $W_{a2} > W_{b2}$ then we can derive a linear order on the set of input vectors w.r.t the output x as well.*

However, if $W_{a1} > W_{b1}$ (see Figure 7(1)) and $W_{a2} < W_{b2}$ (7(2)) then we can only derive a partial order 7(3) w.r.t the output x. Notice that the ordering 7(3) is equal to the ordering given in example 10.

If a particular set of weights is given, for example if $W_{a1} = 10, W_{b1} = 5, W_{a2} = 2$ and $W_{b2} = 8$, then we can check that $(-1,1) > (1,-1)$. For the

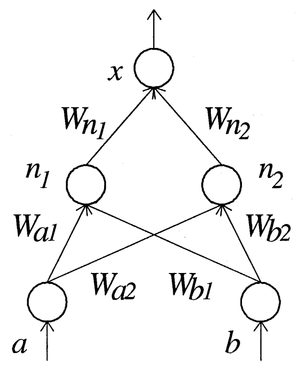


Figure 5: A two hidden neurons network. $W_{ij} \in \Re^+$.

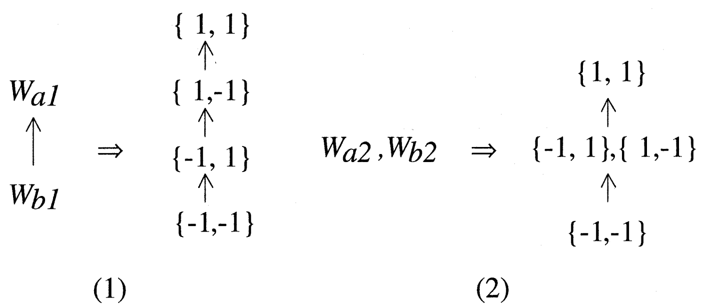


Figure 6: Linear ordering on the set of input vectors.

time being, however, we use the partial ordering of Figure 7(3) because it is "easily found", regardless of the network's weights values.

Examples 10 and 11 indicate that the partial ordering on the set of input vectors is the same for a network with two hidden neurons and for a network with only one hidden neuron. Actually, we will see later that if $W_{ij} \in \Re^+$ then the partial ordering on the set of input vectors is not affected by the number of hidden neurons. Although the weights are different, if a given input occurs in n_1 then the same input has to occur in n_2 as well (where "to occur" means to be responsible for its activation value). That results from the fact that the network's recall process is synchronous, that is, at each time step a unique input vector is presented to the network and is used to compute the activation values of all hidden and output neurons. Hence, given $W_{ij} \in \Re^+$, input $(1, 1)$, for instance, provides the maximum activation of both n_1 and n_2 at the same time.

Let us now try and see if we can find an ordering easily in the case where there are three inputs $\{a, b, c\}$, but still with $W_{ij} \in \Re^+$. It seems reasonable to consider the ordering of Figure 8 since we do not have any

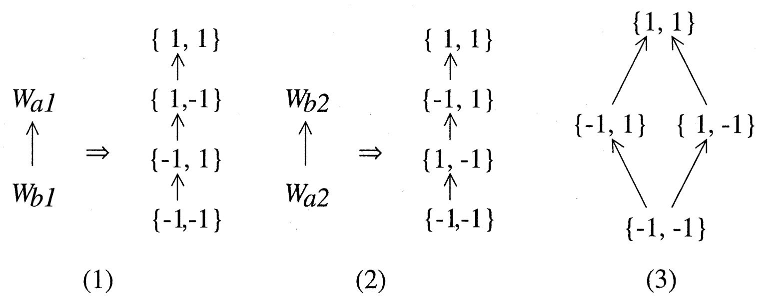


Figure 7: Partial ordering on the set of input vectors.

extra information regarding the network's weights. The ordering is built starting from element $(-1,-1,-1)$ and then flipping each input at a time from -1 to 1 until $(1,1,1)$ is obtained.

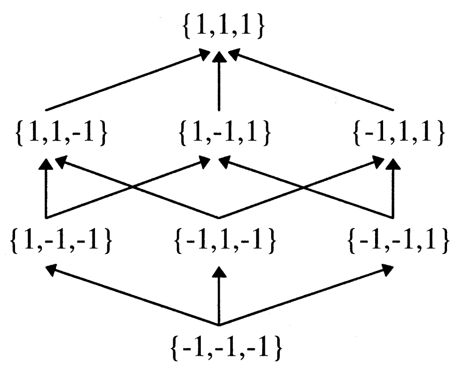


Figure 8: Partial ordering w.r.t set inclusion on the network's input vectors set $(p = 3)$.

It seems that, for an arbitrary number of input and hidden neurons, if $W_{ij} \in \Re^+$, then there exists a unique minimal element $(-1,-1,...,-1)$ and a unique maximum element $(1,1,...,1)$ in the ordering on the set of input vectors w.r.t the output neurons' activations. It seems that $W_{ij} \in \Re^+$ is a sufficient condition for the existence of an easily found ordering on the input vectors space. Let us see if we can confirm this.

We assume the following conventions. A *literal* is a propositional variable or the negation of a propositional variable. Let $\mathbf{P}$ be a finite set of literals. An *interpretation* is a function from $\mathbf{P}$ to $\{tt, ff\}$. That is, an interpretation maps each literal to either *true* or *false*. Given a neural network, we associate each input and output neuron with a unique literal in $\mathbf{P}$. Let $\mathcal{I}$ be the set of input neurons and $\mathcal{O}$ the set of output neurons. Then, each input vector

$\mathbf{i}$ can be seen as an interpretation. Suppose $\mathcal{I} = \{p, q, r\}$. We fix a linear ordering on the symbols of $\mathcal{I}$ and represent it as a list, say $[p, q, r]$. This will allow us to refer to interpretations and input vectors interchangeably in the following way. We represent $\mathbf{i}$ as a string of 1's and -1's, where the value 1 in a particular position in the string means that the literal at the corresponding position in the list of symbols is assigned tt, and the value -1 means that it is assigned ff. For example, if $\mathbf{i} = (1, -1, 1)$ then $\mathbf{i}(p) = \mathbf{i}(r) = tt$ and $\mathbf{i}(q) = ff$.

Each input vector $\mathbf{i}$ can be seen as an abstract representation of a subset of the set of input neurons, with 1's denoting the presence and -1's denoting the absence of a neuron in the set. For example, given the set of input neurons $\mathcal{I}$ as the list $[p, q, r]$, if $\mathbf{i} = (1, -1, 1)$ it represents the set $\{p, r\}$, if $\mathbf{i} = (-1, -1, -1)$ it represents $\{\emptyset\}$, if $\mathbf{i} = (1, 1, 1)$ it represents $\{p, q, r\}$, and so on. We conclude that the set of input vectors $\mathbf{I}$ is an abstract representation of the power set of the set of input neurons $\mathcal{I}$. We write it as $\mathbf{I} = \wp(\mathcal{I})$.

We are now in position to formalize the above concepts. We start by defining a distance function between input vectors. The distance between two input vectors is the number of neurons assigned different inputs by each vector. In terms of the above analogy between input vectors and interpretations, the same distance function can be defined as the number of propositional variables with different truth-values.

Definition 12 *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_j for which $\mathbf{i}_m(i_j) \neq \mathbf{i}_n(i_j)$. ($dist : \mathbf{I} \times \mathbf{I} \rightarrow \aleph$)*

For example, the distance between $\mathbf{i}_1 = (-1, -1, 1)$ and $\mathbf{i}_2 = (1, 1, -1)$ is $dist(\mathbf{i}_1, \mathbf{i}_2) = 3$. The distance between $\mathbf{i}_3 = (-1, 1, -1)$ and $\mathbf{i}_4 = (1, -1, -1)$ is $dist(\mathbf{i}_3, \mathbf{i}_4) = 2$.

Proposition 13 *[31] The function dist is a metric on $\mathbf{I}$.*

Clearly, the function *dist* is also a bounded metric on $\mathbf{I}$. That is, $dist(\mathbf{i}_m, \mathbf{i}_n) \leq p$ for all $\mathbf{i}_m, \mathbf{i}_n \in \mathbf{I}$, where p is the length of the input vectors $\mathbf{i}_m$ and $\mathbf{i}_n$.

Another concept that will prove to be important is the sum of the input elements in a input vector. We define it as follows.

Definition 14 *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_j in $\mathbf{i}_m$, that is $\langle \mathbf{i}_m \rangle = \sum_{j=1}^{p} \mathbf{i}_m(i_j)$. ($\langle\rangle : \mathbf{I} \rightarrow \mathbf{Z}$)*

For example, the sum of $\mathbf{i}_1 = (-1, -1, 1)$ is $\langle \mathbf{i}_1 \rangle = -1$. The sum of $\mathbf{i}_2 = (1, 1, -1)$ is $\langle \mathbf{i}_2 \rangle = 1$.

Now we define the ordering $\leq_{\mathbf{I}}$ on $\mathbf{I} = \wp(\mathcal{I})$ w.r.t set inclusion. Recall that $\mathbf{i}_m \in \mathbf{I}$ is an abstract representation of a subset of $\mathcal{I}$. 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 15 *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. In other words, $sup(\mathbf{I}) = \{1, 1, ..., 1\}$ and $inf(\mathbf{I}) = \{-1, -1, ..., -1\}$. Actually, $[\mathbf{I}, \leq_{\mathbf{I}}]$ is not only that.

Proposition 16 *[29] The partially ordered set $[\mathbf{I}, \leq_{\mathbf{I}}]$ is a distributive lattice.*

Note that $\mathbf{I}$ is actually the n-cube in the Cartesian n-dimensional space of coordinates $x_1, x_2, ..., x_n$ where the generic $x_j (1 \leq j \leq n)$ is either -1 or 1. $\mathbf{I} = \{\mathbf{i}_k \mid \mathbf{i}_k = (i_1, ..., i_p), i_{j(1 \leq j \leq p)} \in \{-1, 1\}\}$.

The following Proposition 17 shows that $\leq_{\mathbf{I}}$ is actually the ordering of our interest w.r.t the network's output.

Proposition 17 *If $W_{ji} \in \Re^+$ then $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_n$ implies $(\mathbf{o}_m(o_j) = \delta(\mathbf{i}_m)) \leq (\mathbf{o}_n(o_j) = \delta(\mathbf{i}_n))$, for all $1 \leq j \leq q$.*

Proof. *Let $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_n$ and $dist(\mathbf{i}_m, \mathbf{i}_n) = 1$, then $\mathbf{i}_m(i_i) = -1$ and $\mathbf{i}_n(i_i) = 1$ for some input i_i. Let r be the number of hidden neurons in the network. Firstly, we have to show that:*

$$
\begin{aligned}
&h(\textstyle\sum_{i=1}^{p}(W^1_{1i}\mathbf{i}_m(i_i) - \theta_{n_1})) + h(\sum_{i=1}^{p}(W^1_{2i}\mathbf{i}_m(i_i) - \theta_{n_2})) + \cdots + \\
&h(\textstyle\sum_{i=1}^{p}(W^1_{ri}\mathbf{i}_m(i_i) - \theta_{n_r})) \leq h(\sum_{i=1}^{p}(W^1_{1i}\mathbf{i}_n(i_i) - \theta_{n_1})) + \\
&h(\textstyle\sum_{i=1}^{p}(W^1_{2i}\mathbf{i}_n(i_i) - \theta_{n_2})) + \cdots + h(\sum_{i=1}^{p}(W^1_{ri}\mathbf{i}_n(i_i) - \theta_{n_r})).
\end{aligned}
$$

By the definition of $\leq_{\mathbf{I}}$ and since $W_{ji} \in \Re^+$ we derive immediately that for all $j(1 \leq j \leq r)$ $\sum_{i=1}^{p}(W^1_{ji}\mathbf{i}_m(i_i) - \theta_{n_j}) \leq \sum_{i=1}^{p}(W^1_{ji}\mathbf{i}_n(i_i) - \theta_{n_j})$, and by the monotonically crescent characteristic of $h(x)$ we obtain $\forall j(1 \leq j \leq r)$ $h(\sum_{i=1}^{p}(W^1_{ji}\mathbf{i}_m(i_i) - \theta_{n_j})) \leq h(\sum_{i=1}^{p}(W^1_{ji}\mathbf{i}_n(i_i) - \theta_{n_j}))$. This proves that if $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_n$ and $dist(\mathbf{i}_m, \mathbf{i}_n) = 1$ then $(\mathbf{n}_m(n_j) = \delta(\mathbf{i}_m)) \leq (\mathbf{n}_n(n_j) = \delta(\mathbf{i}_n))$ for all $1 \leq j \leq r$. In the same way, we obtain that $h(\sum_{i=1}^{r}(W^2_{ji}\mathbf{n}_m(n_i) - \theta_{o_j})) \leq h(\sum_{i=1}^{r}(W^2_{ji}\mathbf{n}_n(n_i) - \theta_{o_j}))$ and therefore that $(\mathbf{o}_m(o_j) = \delta(\mathbf{i}_m)) \leq (\mathbf{o}_n(o_j) = \delta(\mathbf{i}_n))$ for all $1 \leq j \leq q$ (1).

Now, let $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_n$ and $dist(\mathbf{i}_m, \mathbf{i}_n) = k$ $(1 < k \leq p)$. There are $k - 1$ vectors $\mathbf{i}_\xi, ..., \mathbf{i}_\zeta$ such that $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_\xi \leq_{\mathbf{I}} ... \leq_{\mathbf{I}} \mathbf{i}_\zeta \leq_{\mathbf{I}} \mathbf{i}_n$. From (1) above and since $\leq$ is transitive, it follows that if $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_n$ then $(\mathbf{o}_m(o_j) = \delta(\mathbf{i}_m)) \leq (\mathbf{o}_n(o_j) = \delta(\mathbf{i}_n))$ for all $1 \leq j \leq q$.$\square$

4.2 Regular Networks

Let us see now if we can relax the condition $W_{ji} \in \Re^+$ and still find easily an ordering in the network's input vectors set. We start by giving an example.

Example 18 *Consider the network given at example 11 (Figure 5), but now assume W_{b1} and $W_{b2} < 0$. Although some weights are negative, we can find a "regularity" in the network. For example, the input neuron b contributes negatively for both n_1 and n_2, and there are no negative connections from the hidden to the output layer. Following [12], we can transform the network of Figure 5 into the network of Figure 9, where all weights are positive and the input neuron b is negated.*

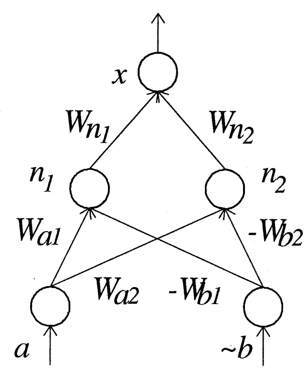


Figure 9: The positive form of a (regular) network.

Given the network of Figure 9, we can find an ordering on the set of input vectors in the same way as before. The only difference is that now $\mathcal{I} = \{a, \sim b\}$. We will see later that, if we account for the fact that $\mathcal{I}$ may now have negated literals (default negation), then the networks of Figures 5 and 9 are equivalent.

Let us analyse what we have done in the above example. If the connections from the hidden layer to any one neuron in the output layer of a network are either all positive or all negative, we do the following for each input neuron y:

1. if y is linked to the hidden layer through connections with negative weights W_{jy} only:
   (a) change each W_{jy} to $-W_{jy}$ and rename y by $\sim y$.
2. If y is linked to the hidden layer through positive and negative connections:
   (a) add a neuron named $\sim y$ to the input layer, and
   (b) for each negative connection with weight W_{jy} from y to n_j:
      i. add a new connection with weight $-W_{jy}$ from $\sim y$ to n_j, and

ii. delete the connection with weight W_{jy} from y to n_j.

3. If y is linked to the hidden layer through connections with positive weights only:

   (a) do nothing.

We call the above procedure the *Transformation Algorithm.*

Example 19 *Consider again the network given at example 11 (Figure 5), but now assume that only* $W_{a2} < 0$. *Applying the transformation algorithm we obtain the network of Figure 10.*

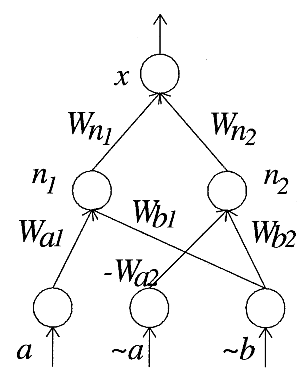


Figure 10: The positive form of a (non regular) network.

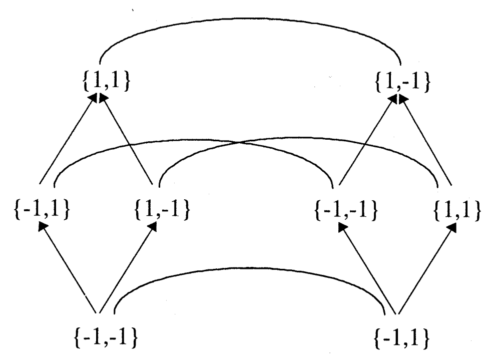
Although the network of Figure 10 has positive weights only, it is clearly not equivalent to the original network (Figure 5). In this case, the combination of n_1 *and* n_2 *is not straightforward. Note that,* $\mathbf{i} = (1, 1)$ *in the original network provides the maximum activation of* n_1*, but not the maximum activation of* n_2 *that is given by* $\mathbf{i} = (-1, 1)$. *We can not affirm anymore that* $(1, 1)$ *is bigger than* $(-1, 1)$ *w.r.t the output* x*, without having to check them by querying the network.*

The above examples 18 and 19 indicate that if the transformation algorithm generates a network where complementary literals (say, a and $\sim a$) appear in the input layer (see the network of Figure 10) then the ordering $\leq_{\mathbf{I}}$ on $\mathbf{I}$ is not applicable. On the other hand, if it does not, it seems that $\leq_{\mathbf{I}}$ is still valid for networks that have "well-behaved" negative weights. This motivates the following definition.

Definition 20 *A single hidden layer neural network is said to be* regular *if its connections from the hidden layer to each output neuron have either all positive or all negative weights, and if the above transformation algorithm generates on it a network without complementary literals in the input layer.*

Back to example 18, we have seen that the positive form N_+ of a regular network N may have negated literals in its input set (e.g. $\mathcal{I}_+ = \{a, \sim b\}$). In this case, if we represent $\mathcal{I}_+$ as a list, say $[a, \sim b]$, and refer to an input vector $\mathbf{i} = (-1, 1)$ w.r.t $\mathcal{I}_+$ then we consider $\mathbf{i}$ as the abstract representation of the set $\{\sim b\}$. In the same way, $\mathbf{i} = (1, -1)$ represents $\{a\}$, and so on. In this sense, the set of input vectors of N_+ can be ordered w.r.t set inclusion exactly as before, using Definition 15. The following example illustrates that.

Example 21 *Consider the network N_+ of Figure 9. Given $\mathcal{I}_+ = [a, \sim b]$ we obtain the ordering (1) of Figure 11 w.r.t set inclusion. The ordering 11(2) on the set of input vectors of the original network N is obtained by mapping each element of (1) into (2) using $\sim b = 1 \Rightarrow b = -1$ and $\sim b = -1 \Rightarrow b = 1$.*



$\mathcal{I}_+ = [a, \sim b]$

(1)

$\mathcal{I} = [a, b]$

(2)

Figure 11: The ordering w.r.t set inclusion on the positive form of a network (1) and the ordering on the original network (2).

As a result, querying the network N_+ with $\mathbf{i} = (1, 1)$ is equivalent to querying the network N with $\mathbf{i} = (1, -1)$, querying N_+ with $\mathbf{i} = (-1, 1)$ is equivalent to querying N with $\mathbf{i} = (-1, -1)$, and so on.

More precisely, we define the function σ mapping input vectors of the positive form into input vectors of the subnetwork as follows. Let $\mathbf{I}$ be the set of input vectors of s tuples. Given $\mathcal{I}_+$ and an abstract representation $\mathbf{I}_+$ of $\wp(\mathcal{I}_+)$, each element $x_i \in \mathcal{I}_+$, $1 \leq i \leq s$, is mapped to the set $\{-1, 1\}$ such that $\sigma_{[x_1,...,x_s]}(i_1, ..., i_s) = (i'_1, ..., i'_s)$, where $i'_i = i_i$ if x_i is a positive literal and $i'_i = -i_i$ if x_i is a negative literal. For example $\sigma_{[a,\sim b,c,\sim d]}(1, 1, -1, -1) = (1, -1, -1, 1)$.

Note that the correspondence between input vectors and interpretations is still valid. We only need to define $\mathbf{i}(\sim p) = ff$ iff $\mathbf{i}(p) = tt$ and $\sim\sim p = p$. For example, for $\mathcal{I}_+ = [a, \sim b]$, if $\mathbf{i} = (-1, -1)$ then $\mathbf{i}(a) = ff$ and $\mathbf{i}(b) = tt$.

Proposition 22 *If a network is regular with input $\mathcal{I}_+$ then $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_n$ implies $(\mathbf{o}_m(o_j) = \delta(\sigma_{[\mathcal{I}_+]}(\mathbf{i}_m)) \leq (\mathbf{o}_n(o_j) = \delta(\sigma_{[\mathcal{I}_+]}(\mathbf{i}_n))$, for all $1 \leq j \leq q$.*
Proof. *Straightforward by Proposition 17 and by the above definition of the mapping function σ.*$\square$

Proposition 22 establishes the correlation between regular networks and their positive counterpart. As a result, the extraction procedure can either use the set inclusion ordering, and query directly the positive form of the network, or use the mapping function σ to obtain the ordering on the regular, original network, and query the original network. We will adopt the first policy. Note that if the network is already positive then σ is the identity function.

We have seen briefly that if we can find an ordering in a network's input vectors set easily, as a result there are some properties that can help pruning the input search space during a pedagogical extraction of rules. Let us now define precisely these properties.

Proposition 23 *(Search Space Pruning Rule 1) Let $\mathbf{i}_m$ and $\mathbf{i}_n$ be input vectors of a regular neural network N, such that $dist(\mathbf{i}_m, \mathbf{i}_n) = 1$ and $\langle \mathbf{i}_m \rangle < \langle \mathbf{i}_n \rangle$. If $\mathbf{i}_n$ does not satisfy the constraint $\mathbf{C}o_j$ on the j-th output neuron of N, then $\mathbf{i}_m$ does not satisfy $\mathbf{C}o_j$ either.*
Proof. *Directly by Definitions 12, 14 and 15, if $dist(\mathbf{i}_m, \mathbf{i}_n) = 1$ and $\langle \mathbf{i}_m \rangle < \langle \mathbf{i}_n \rangle$ then $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_n$. By Proposition 17, $o_j(\mathbf{i}_m) \leq o_j(\mathbf{i}_n)$. That completes the proof.* $\square$

Proposition 24 *(Search Space Pruning Rule 2) Let $\mathbf{i}_m$ and $\mathbf{i}_n$ be input vectors of a regular neural network N, such that $dist(\mathbf{i}_m, \mathbf{i}_n) = 1$ and $\langle \mathbf{i}_m \rangle < \langle \mathbf{i}_n \rangle$. If $\mathbf{i}_m$ satisfies the constraint $\mathbf{C}o_j$ on the j-th output neuron of N, then $\mathbf{i}_n$ also satisfies $\mathbf{C}o_j$.*
Proof. *This is the contrapositive of Proposition 23.*$\square$

Propositions 23 and 24 say that for any $\mathbf{i} \in \mathbf{I}$, starting from $sup(\mathbf{I})$ (resp. $inf(\mathbf{I})$), if $\mathbf{i}$ does not activate (resp. activates) the j-th output neuron, then the immediate predecessors (resp. successors) of $\mathbf{i}$ does not activate (resp. activates) it as well.

We have seen very briefly in example 10 that simplifications, like $ab \rightarrow x$ and $a \sim b \rightarrow x \Rightarrow a \rightarrow x$, can be done in the set of rules extracted. Moreover, they can be identified in the input vectors ordering, prior to the actual extraction of rules. We define, therefore, the following "*simplification rules*" that will help in the extraction of a smaller and clearer set of rules.

Definition 25 *(*Subsumption*) A rule r_1 subsumes a rule r_2 iff r_1 and r_2 have the same conclusion and the set of premises of r_1 is a subset of the set of premises of r_2.*

For example, $a \rightarrow x$ subsumes $ab \rightarrow x$ and $a \sim b \rightarrow x$.

Definition 26 *(*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, where $j \leq |\mathcal{I}|$. 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.*

For example, if $\mathcal{I} = \{a, b, c\}$ and we write $a \sim b \rightarrow x$, then it simplifies $a \sim bc \rightarrow x$ and $a \sim b \sim c \rightarrow x$. Note that, considering the ordering on $\mathbf{I}$, the above property requires that two adjacent ($dist = 1$) input vectors $\mathbf{i}_m = (1, -1, 1)$ and $\mathbf{i}_n = (1, -1, -1)$ satisfy x.

Definition 27 *(*Fact*) If a literal L_{j+1} holds in the presence of any combination of the truth values of literals $L_1, ..., L_j$ in $\mathcal{I}$ then we derive a rule of the form $\rightarrow L_{j+1}$ (L_{j+1} is a fact).*

Definition 27 is a important special case of Definition 26. Considering the ordering on $\mathbf{I}$, an output neuron x is a fact iff $inf(\mathbf{I})$ satisfies the constraint on x. Note that, by Proposition 24, if $inf(\mathbf{I})$ satisfies x then any other input vector in $\mathbf{I}$ satisfies x as well. Another interesting special case occurs when $sup(\mathbf{I})$ does not satisfy x. In that case, by Proposition 23, any other input vector in $\mathbf{I}$ does not satisfy x either, and we can stop the search process deriving no rules with conclusion x.

Definition 28 *(*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 $m(\mathcal{I}\prime) \rightarrow L_{j+1}$.*

The above Definition 28 may be very useful in helping to reduce the number of rules extracted. It states that, for example, $2(abc) \rightarrow x$ represents $ab \rightarrow x, ac \rightarrow x$,and $bc \rightarrow x$. In this way, if for example we write $3(abcdef) \rightarrow x$ then this rule is a short representation of at least $C_3^6 = 20$ rules [9]. There is a rather intricate relation between each rule of the form *M of N* and the ordering on the set of input vectors $\mathbf{I}$, in the sense that each valid *M of N* rule represents a subset of $\mathbf{I}$. Here is a flavour of that relation in a example where it is easy to identify it. Suppose $\mathcal{I} = \{a, b, c\}$ and assume that $\mathcal{I}\prime = \mathcal{I}$. Let us say that the output neuron in question is x and that constraint $\mathbf{C}_{o_x}$

[9] Note that if $\mathcal{I} = \{a, b, c\}$ and we write $1(ab) \rightarrow x$, then it is a simplification of $C_1^2 = 2$ rules: $a \rightarrow x$ and $b \rightarrow x$. However, by definition 26, $a \rightarrow x$ and $b \rightarrow x$ are already simplifications of $abc \rightarrow x$, $ab \sim c \rightarrow x$, $a \sim bc \rightarrow x$, $a \sim b \sim c \rightarrow x$, $\sim abc \rightarrow x$, *and* $\sim ab \sim c \rightarrow x$.

is satisfied by at least one input vector in $\mathbf{I}$. If only $sup(\mathbf{I})$ satisfies $\mathbf{C}_{o_x}$, we derive the rule $abc \rightarrow x$. Clearly, this rule is equivalent to $3(abc) \rightarrow x$. If all immediate predecessor of $sup(\mathbf{I})$ also satisfy $\mathbf{C}_{o_x}$, it is not difficult to verify that the four rules obtained ($r_1 = abc \rightarrow x$, $r_2 = ab \sim c \rightarrow x$, $r_3 = a \sim bc \rightarrow x$, $r_4 = \sim abc \rightarrow x$) can be represented by $2(abc) \rightarrow x$. That is because, by Definition 26, each rule r_2, r_3 and r_4 can be simplified together with r_1, deriving $abc \rightarrow x$, $ab \rightarrow x$, $ac \rightarrow x$ and $bc \rightarrow x$. Since, by Definition 25, $abc \rightarrow x$ is subsumed by any of the other three rules, we obtain $2(abc) \rightarrow x$. Moreover, $2(abc) \rightarrow x$ subsumes $3(abc) \rightarrow x$. This motivates the definition of yet another simplification rule, as follows.

Definition 29 *(*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 $m < p$.*

Back to the illustration about the relation between *M of N* rules and subsets of $\mathbf{I}$, let us see what happens if the elements at distance 2 from $sup(\mathbf{I})$ all satisfy $\mathbf{C}_{o_x}$. We expect that the set of rules obtained from $\mathbf{I}$ could be represented by $1(abc) \rightarrow x$, and in fact it is. Let $r_1 = a \sim b \sim c \rightarrow x, r_2 = \sim ab \sim c \rightarrow x$, and $r_3 = \sim a \sim bc \rightarrow x$. By Proposition 24, from any two rules in $\{r_1, r_2, r_3\}$ we have $r_4 = ab \sim c \rightarrow x, r_5 = a \sim bc \rightarrow x$, and $r_6 = \sim abc \rightarrow x$. Again by Proposition 24, from any rule in $\{r_4, r_5, r_6\}$ we have $r_7 = abc \rightarrow x$. By Definition 26, from $\{r_1, r_4\}$ we have $r_8 = a \sim c \rightarrow x$, from $\{r_5, r_7\}$ we have $r_9 = ac \rightarrow x$, and from r_8 and r_9 we derive $r_a = a \rightarrow x$. Similarly, from $\{r_2, r_4, r_6, r_7\}$ we derive $r_b = b \rightarrow x$, and from $\{r_3, r_5, r_6, r_7\}$ we derive $r_c = c \rightarrow x$. Finally, since r_a, r_b and r_c together subsume any rule previously obtained, by Definition 28 we may derive the single *M of N* rule $1(abc) \rightarrow x$. We have identified, therefore, a pattern in the ordering on $\mathbf{I}$ w.r.t a group of *M of N* rules, the ones where $\mathcal{I}\prime = \mathcal{I}$. More generally, given $|\mathcal{I}| = k$, if all the elements in $\mathbf{I}$ that are at distance d from $sup(\mathbf{I})$ satisfy a constraint $\mathbf{C}_{o_x}$, then derive the rule $(k-d)(\mathcal{I}) \rightarrow x$. Note that there are C^k_{k-d} elements at distance d from $sup(\mathbf{I})$ and that, as a result of Proposition 24, if all the elements in $\mathbf{I}$ at distance d from $sup(\mathbf{I})$ satisfy $\mathbf{C}_{o_x}$, then any other element at distance d' from $sup(\mathbf{I})$, $0 \leq d' < d$ also satisfies $\mathbf{C}_{o_x}$.

Remark 1 *We have defined regular networks (see Definition 20) either with all the weights from the hidden layer to each output neuron positive or with all of them negative. We have, although, considered in the above examples and definitions only the ones where all the weights are positive. However, it is not difficult to verify that the constraint $\mathbf{C}_{o_j}$ on the j-th output of a regular network with negative weights from hidden to output layer is $W^2_{j1}n_1 + W^2_{j2}n_2 + \cdots + W^2_{jr}n_r < h^{-1}(A_{min}) + \theta_{o_j}$. As a result, the only difference now is on the sign ($<$) of the constraint. In other words, in this case we only need to invert the signs at Propositions 23 and 24. All remaining definitions and propositions are still valid.*

We referred to soundness and completeness of the extraction algorithm in a somewhat vague manner. Let us define these concepts precisely.

Definition 30 *(*Extraction Algorithm Soundness*) A rules' extraction algorithm from a neural network N is* sound *iff for each rule r_i extracted, whenever the premise of r_i is presented to N as input vector, in the presence of any combination of the input values of literals not referenced by rule r_i, the conclusion of r_i presents activation greater than A_{min} in the output vector of N.*

Definition 31 *(*Extraction Algorithm Completeness*) A rules' extraction algorithm from a neural network N is* complete *iff each rule extracted by exhaustively verifying all the combinations of the input vector of N either belongs to or is subsumed by a rule in the set of rules generated by the extraction algorithm.*

We are finally in position to present the extraction algorithm for regular networks, which will be refined in section 5 for the general case extraction.

- *Knowledge Extraction Algorithm for Regular Networks*[10]

1. Apply the *Transformation Algorithm* over N, obtaining its positive form N_+;
2. Find $inf(\mathbf{I})$ and $sup(\mathbf{I})$ w.r.t N_+ using σ;
3. For each neuron o_j in the output layer of N_+ do:
   (a) Query N_+ with input vector $inf(\mathbf{I})$. If $o_j > A_{min}$, apply the Simplification Rule *Fact* and stop.
   (b) Query N_+ with input vector $sup(\mathbf{I})$. If $o_j < A_{min}$, stop.
   /* *Search the input vectors space* $\mathbf{I}$.
   (c) $\mathbf{i}_\perp := inf(\mathbf{I})$; $\mathbf{i}_\top := sup(\mathbf{I})$;
   (d) While $dist(\mathbf{i}_\perp, inf(\mathbf{I})) \leq n\text{DIV}2$ or $dist(\mathbf{i}_\top, sup(\mathbf{I})) \leq n\text{DIV}2 + n\text{MOD}2$, where n is the number of input neurons of N_+, and still generating new $\mathbf{i}_\perp$ or $\mathbf{i}_\top$, do:
   /* *Generate the successors of* $\mathbf{i}_\perp$ *and query the network*
      i. set new $\mathbf{i}_\perp :=$ old $\mathbf{i}_\perp$ flipped according to the ordering on $\mathbf{I}$;[11]
      ii. Query N_+ with input vector $\mathbf{i}_\perp$;
      iii. If Search Space *Pruning Rule 2* is applicable, stop generating new $\mathbf{i}_\perp$;

[10] The algorithm is kept simple for clarity, and is not necessarily the most efficient.
[11] From $inf(\mathbf{I})$, we generate new $\mathbf{i}_\perp$ from right to left.

iv. Apply the Simplification Rule *Complementary Literals*, and Add the rules derived accordingly to the rule set.

/* *Generate the predecessors of* $\mathbf{i}_\top$ *and query the network*

v. set new $\mathbf{i}_\top :=$ old $\mathbf{i}_\top$ flipped according to the ordering on $\mathbf{I}$;[12]

vi. Query N_+ with input vector $\mathbf{i}_\top$;

vii. If Search Space *Pruning Rule 1* is applicable, stop generating new $\mathbf{i}_\top$;

viii. Apply the Simplification Rule *M of N*, and Add the rules derived accordingly to the rule set.

(e) Apply the Simplification Rules *Subsumption* and *M of N Subsumption* on the rule set regarding o_j.

Note that if the weights from the hidden to the output layer of N are negative, we simply substitute $inf(\mathbf{I})$ by $sup(\mathbf{I})$ and vice-versa. In a given application, the above extraction algorithm can be halted if a desired degree of accuracy is achieved in the set of rules. The algorithm is such that the exact symbolic representation of the network is being approximated at each cycle.

Example 32 *Suppose* $\mathcal{I} = \{a, b, c\}$ *and let* $\mathbf{I} = \wp(\mathcal{I})$ *be ordered w.r.t set inclusion. We start by checking* $inf(\mathbf{I})$ *w.r.t an output neuron* x*. If* $inf(\mathbf{I})$ *activates* x*, i.e.,* $inf(\mathbf{I})$ *satisfies constraint* C_{o_x}*, then by Proposition 24 any other input vector activates* x *and by Definition 27 we can extract* $\rightarrow x$ *and stop. If, on the other hand,* $inf(\mathbf{I})$ *does not activate* x*, then we may need to query the network with the immediate successors of* $inf(\mathbf{I})$*. Let us call these input vectors* $\mathbf{I}^*$*, where* $dist(inf(\mathbf{I}), \mathbf{I}^*) = 1$*.*

We proceed to check the element $sup(\mathbf{I})$*. If* $sup(\mathbf{I})$ *does not satisfy* C_{o_x}*, by Proposition 23 we can stop, extracting no rules with conclusion* x*. If* $sup(\mathbf{I})$ *activates* x*, we conclude that* $abc \rightarrow x$*, but we still have to check the input vectors* $\mathbf{I}^{**}$ *at distance 1 from* $sup(\mathbf{I})$*. We may also later apply some simplification on* $abc \rightarrow x$*, if at least one of the input vectors in* $\mathbf{I}^{**}$ *activates* x*. Hence, we keep* $abc \rightarrow x$ *in stand by and proceed.*

Let us say that we choose to start by checking $\mathbf{i}_1 = (-1, -1, 1)$ *in* $\mathbf{I}^*$*. If* $\mathbf{i}_1$ *does not satisfy* C_{o_x}*, we have to check the remaining inputs in* $\mathbf{I}^*$*. However, if* $\mathbf{i}_1$ *activates* x *then, again by Proposition 24, we know that* $(-1, 1, 1)$ *and* $(1, -1, 1)$ *also do. This tells us that not all the inputs in* $\mathbf{I}^{**}$ *need to be checked. Moreover, if all the elements in* $\mathbf{I}^*$ *activate* x *then we can use Definition 28 to derive* $1(abc) \rightarrow x$ *and stop the search.*

Analogously, when checking $\mathbf{I}^{**}$ *we can obtain information about* $\mathbf{I}^*$*. If, for instance,* $\mathbf{i}_2 = (1, 1, -1)$ *does not activate* x *then* $(-1, 1, -1)$ *and* $(1, -1, -1)$ *in* $\mathbf{I}^*$ *do not either, now by Proposition 23. If, on the contrary,*

[12] From $sup(\mathbf{I})$, we generate new $\mathbf{i}_\top$ from left to right.

$\mathbf{i}_2$ *activates* x*, we can derive* $ab \to x$*, using Proposition 24 and Definition 26. If not only* $\mathbf{i}_2$ *but also the other inputs in* $\mathbf{I}^{**}$ *activate* x *then we obtain* $2(abc) \to x$*, which subsumes* $abc \to x$ *by Definitions 28 and 25. In this case, we still need to query the network with inputs* $\mathbf{i}$ *at distance 1 from* $\mathbf{i}_2$ *such that* $\langle \mathbf{i} \rangle < \langle \mathbf{i}_2 \rangle$*, but those inputs are already the ones in* $\mathbf{I}^{**}$ *and therefore we can stop. Note that the stopping criteria are the following: either all elements in the ordering are visited or, if not, for each element not visited, Propositions 23 and 24 guarantee that it is safe not to consider it, in the sense that it is either already represented in the set of rules or irrelevant and will not give rise to any new rule.*

Theorem 33 *(Soundness) The extraction algorithm for regular networks is sound (satisfies Definition 30).*

Proof. *We have to show that, whether a rule* r *is extracted by querying the network (*Case 1*) or by a simplification of rules (*Case 2*), any rule* r' *that is subsumed by* r*, including* r *itself, can be obtained by querying the network. We prove this by contradiction. Consider a set* $\mathbf{I}$ *of* p*-ary input vectors. Assume that there exist rules* r *and* r' *such that* r' *is subsumed by* r*, and* r' *is not obtainable by querying the network. Assume also that* r *contains the largest number of premisses of such a rule. Let* X_i *denote* L_i *or* $\sim L_i$ $(1 \leq i \leq p)$*.*

Case 1*: If* r *is itself obtained by querying the network, then the only possible subsumed rule is* r*, and obviously this yields a contradiction.*

Case 2*:* r *is either a simplification by Complementary Literals, or a Fact, or a M of N rule. It is shown that each assumption yields a contradiction.*

Let $r = L_1, ..., L_q \to L_j$ $(1 \leq q < p)$ *be a simplification by Complementary Literals. Then,* r *is derived from two rules* $r'_1 = L_1, ..., L_s, ..., L_q \to L_j$ *and* $r'_2 = L_1, ..., \sim L_s, ..., L_q \to L_j$*,* $(1 \leq s \leq q)$*. Each of these has more premisses than* r*. So, by assumption, all rules subsumed by* r'_1 *and* r'_2 *are obtainable by querying the network. By Proposition 24,* r *is also obtained by querying the network. Since, by Definition 25, any other rule subsumed by* r *is also subsumed by either* r'_1 *or by* r'_2*, this leads to a contradiction.*

Let $r = \to L_j$ *be a simplification by Fact. Then,* r *must have been obtained by querying the network with* $inf(\mathbf{I})$*. By Proposition 24, any rule of the form* $X_1, ..., X_p \to L_j$ *is also obtainable by querying the network, contradicting the assumption about* r'*.*

Finally, if a further simplification is made, to obtain $r = m(L_1, ..., L_n) \to L_j$ $(1 \leq m < n \leq p)$ *by M of N simplification, then* r *is obtained from a set of rules of the form* $L_1, ..., L_m \to L_j$*, where* $L_1, ..., L_m$ *are* m *elements chosen from* $\{L_1, ..., L_n\}$*. By the previous cases, all subsumed rules are obtainable by querying the network.* $\square$

Theorem 34 *(Completeness) The extraction algorithm for regular networks is complete (satisfies Definition 31).*

Proof. *We have to show that the extraction algorithm terminates either when all possible combinations of the input vector have been queried in the network (*Case 1*) or the set of rules extracted subsumes any rule derived from elements not queried (*Case 2*).* Case 1 *is trivial. In* Case 2*, we have to show that any element not queried either would not generate a rule (*Case 2(*i*)*) or would generate a rule that is subsumed by some rule extracted (*Case 2(*ii*)*).*

Consider a set $\mathbf{I}$ *of p-ary input vectors.*

Case 2(i)*: Let* $\mathbf{i}_m, \mathbf{i}_n \in \mathbf{I}$, $dist(\mathbf{i}_m, \mathbf{i}_n) = q$ $(1 \leq q \leq p)$ *and* $\langle \mathbf{i}_m \rangle < \langle \mathbf{i}_n \rangle$. *Assume that* $\mathbf{i}_n$ *is queried in the network and that* $\mathbf{i}_n$ *does not generate a rule. By Proposition 23 q times,* $\mathbf{i}_m$ *would not generate a rule either.*

Case 2(ii)*: Let* $\mathbf{i}_k, \mathbf{i}_o \in \mathbf{I}$, $dist(\mathbf{i}_k, \mathbf{i}_o) = t$ $(1 \leq t \leq p)$ *and* $\langle \mathbf{i}_k \rangle < \langle \mathbf{i}_o \rangle$. *Assume that* $\mathbf{i}_k$ *is queried in the network and that* $\mathbf{i}_k$ *derives a rule* r_k. *Let* $S = \{L_1, ..., L_s\}$ *be the set of positive literals in the body of* r_k*, where* $s \in [1, p]$. *By Definition 26, the rule* $r = L_1, ..., L_s \rightarrow L_j$ *can be obtained from* r_k. *Clearly, r subsumes* r_k. *Now, by Proposition 24 t times,* $\mathbf{i}_o$ *would also derive a rule* r_o. *Let* $U = \{L_1, ..., L_u\}$ *be the set of positive literals in the body of* r_o*, where* $u \in [1, p]$. *Since* $\langle \mathbf{i}_k \rangle < \langle \mathbf{i}_o \rangle$ *then* $S \subset U$ *and, by Definition 25, r also subsumes* r_o.

That completes the proof since all the stopping criteria of the extraction algorithm have been covered. □

5 The Extraction Algorithm for Non-Regular Networks (The General Case)

So far, we have seen that for the case of regular networks it is possible to apply an ordering on the set of input vectors, and use a sound and complete pedagogical extraction algorithm that searches for relevant input vectors in this ordering. Furthermore, the neural network and its set of rules can be shown equivalent (that results directly from the proofs of soundness and completeness of the extraction algorithm).

Despite the above results being highly desirable, it is much more likely that a non-regular network will result from an unbiased training process. In order to overcome this limitation, in the sequel we present the extension of our extraction algorithm to the general case, the case of non-regular networks. The idea is to investigate fragments of the non-regular network in order to find regularities over which the above described extraction algorithm could be applied. We would then split a non-regular network into regular subnetworks, extract the symbolic knowledge from each subnetwork, and finally assemble the rule set of the original non-regular network. That, however, is a decompositional approach, and we need to bear in mind that the collective behavior of a network is not equivalent to the behavior of its parts grouped together. We will need, therefore, to be specially careful when assembling the network's final set of rules.

The problem with non-regular networks is that it is difficult to find the ordering on the set of input vectors without having to actually check each input. In this case, the gain obtained in terms of complexity could be lost. By considering its regular subnetworks, the main problem we have to tackle is how to combine the information obtained into the network's rule set. That problem is due mainly to the non discrete nature of the network's hidden neurons. As we have seen in Example 1, that is the reason why a decompositional approach may be unsound (see section 2). In order to solve this problem, we will assume that hidden neurons present four possible activations $(-1, A_{max}, A_{min}, 1)$. Performing a kind of worst case analysis, we will be able to show that the general case extraction is sound, although we will have to exchange completeness for efficiency.

5.1 Regular Subnetworks

We start by defining precisely the above intuitive concept of a subnetwork.

Definition 35 *(subnetworks) Let N be a neural network with p input neurons $\{i_1, ..., i_p\}$, r hidden neurons $\{n_1, ..., n_r\}$ and q output neurons $\{o_1, ..., o_q\}$. Let $N\prime$ be a neural network with p' input neurons $\{i'_1, ..., i'_{p'}\}$, r' hidden neurons $\{n'_1, ..., n'_{r'}\}$ and q' output neurons $\{o'_1, ..., o'_{q'}\}$. $N\prime$ is a* subnetwork *of N iff $0 \leq p' \leq p$, $0 \leq r' \leq r$, $0 \leq q' \leq q$, and for all i'_i, n'_j, o'_k in N', $W_{n'_j i'_i} = W_{n_j i_i}$, $W_{o'_k n'_j} = W_{o_k n_j}$, $\theta_{n'_j} = \theta_{n_j}$ and $\theta_{o'_k} = \theta_{o_k}$.*

Our first task is to find the regular subnetworks of a non-regular network. Indeed, any single hidden layer network can be split into exactly r regular subnetworks, where r is the number of hidden neurons. It is not difficult to check that any network containing a single hidden neuron is regular. As a result, we could be tempted to split a non-regular network into r subnetworks, each containing the same input and output neurons as the original network plus only one of its hidden neurons.

However, let us briefly analyse what could happen if we were to extract rules from each of the above subnetworks. Suppose that, for a given output neuron x, from the subnetwork containing the hidden neuron n_1, the extraction algorithm obtains the rules $ab \rightarrow_{n_1} x$ and $cd \rightarrow_{n_1} x$; from the subnetwork containing the hidden neuron n_2, it obtains the rule $cd \rightarrow_{n_2} x$; and so on. The problem is that the information that ab implies x through n_1 is not very useful. It may be the case that the same input ab has no effect on the activation of x through n_2, or that it actually blocks the activation of x through n_2. It may also be the case that, for instance, $ad \rightarrow x$ as a result of the combination of the activations of n_1 and n_2 together, but not through each one of them individually. If, therefore, we take the intersection of the rules derived from each subnetwork, we would be extracting only the rules that are encoded in every hidden neuron individually, but not the rules derived from each hidden neuron or from the collective effect of the hidden neurons' activations. If, on the other hand, we take the union of the rules derived from each subnetwork, then the extraction could clearly be unsound.

It seems that we need to analyse a non-regular network first from the input layer to each of the hidden neurons, and then from the hidden layer to each of the output neurons. That motivates the following definition of "*Basic Neural Structures*".

Definition 36 *(Basic Neural Structures) Let N be a neural network with p input neurons $\{i_1, ..., i_p\}$, r hidden neurons $\{n_1, ..., n_r\}$ and q output neurons $\{o_1, ..., o_q\}$. A subnetwork N' of N is a* Basic Neural Structure *(*BNS*) iff either N' contains exactly p input neurons, 1 hidden neuron and 0 output neurons of N, or N' contains exactly 0 input neurons, r hidden neurons and 1 output neuron of N.*

Note that a *BNS* is a neural network with no hidden neurons and a single neuron in its output layer. Note also that a network N with r hidden neurons and q output neurons contains $r + q$ *BNSs.* We call a *BNS* containing no output neurons of N, an *Input to Hidden BNS*; and a *BNS* containing no input neurons of N, a *Hidden to Output BNS.*

Proposition 37 *Any* BNS *is (vacuously) regular.*
Proof. *Directly by Definition 36, by applying the Transformation Algorithm on a* BNS, *a network without complementary literals in the input layer is obtained. By Definition 20, since a* BNS *does not contain hidden neurons, it is (vacuously) regular.* □

Proposition 37 shows that the Transformation Algorithm applied over a *BNS* will derive a positive network ($W_{ji} \in \Re^+$), the *BNS's* positive form, which will not contain pairs of neurons labelled as complementary literals in its input layer. The above result indicates that *BNSs*, which can be easily obtained from a network N, are suitable subnetworks for applying the extraction algorithm when N is a non-regular network.

5.2 Knowledge Extraction from BNSs

We have seen that, if we split a non-regular network into *BNSs*, there is always an ordering easily found in each subnetwork. The problem, now, is that *Hidden to Output BNSs* do not present discrete activations $\{-1, 1\}$ in their input layer. Instead, each input neuron may present activations in the ranges $(-1, A_{max})$ or $(A_{min}, 1)$, where $A_{max} \in (-1, 0)$ is a predefined value, and we will need to consider this during the extraction from *Hidden to Output BNSs.* For the time being, let us simply assume that each neuron in the input layer of a *Hidden to Output BNS* is labeled n_i, and if n_i is connected to the neuron in the output layer of the *BNS* through a negative weight, then we rename it $\sim n_i$ when applying the Transformation Algorithm, as done for regular networks. Moreover, let us assume that neurons in the input layer of the positive form of *Hidden to Output BNSs* present activations in $\{-1, A_{min}\}$ only. This results from the above mentioned worst case analysis, as we will see later in this section.

We need to rewrite Search Space Pruning Rules 1 and 2 for *BNSs*. Now, given a *BNS* with i input neurons $\{i_1, ..., i_i\}$ and the output neuron o_j, the constraint $\mathbf{C}_{o_j}$ on the output neuron's activation is simply given by: *j is true iff* $W_{o_j i_1} i_1 + W_{o_j i_2} i_2 + ... + W_{o_j i_i} i_i > h^{-1}(A_{min}) + \theta_{o_j}$.

Proposition 38 *Let* $\mathbf{i}_m$ *and* $\mathbf{i}_n$ *be input vectors of the positive form of a* BNS *with output neuron* o_j*. If* $\mathbf{i}_m \leq_\mathbf{I} \mathbf{i}_n$ *then* $o_j(\mathbf{i}_m) \leq o_j(\mathbf{i}_n)$.
Proof. *Case 1* (Input to Hidden BNSs)*: Directly, by Proposition 37 and Proposition 17 we obtain* $o_j(\mathbf{i}_m) \leq o_j(\mathbf{i}_n)$*. Case 2* (Hidden to Output BNSs)*: Assume* $\mathbf{i}_m(i_k) = -1$ *and* $\mathbf{i}_n(i_k) = A_{min}$*. Since* $W_{ji} \in \Re^+$ *and* $A_{min} > 0$*, we have* $(W_{o_j i_k}(-1) - \theta_{o_j}) \leq (W_{o_j i_k}(A_{min}) - \theta_{o_j})$*. Since* $\mathbf{i}_m \leq_\mathbf{I} \mathbf{i}_n$*, we have* $(\sum_{i=1}^{p}(W_{o_j i_i}\mathbf{i}_m(i_i) - \theta_{o_j})) \leq (\sum_{i=1}^{p}(W_{o_j i_i}\mathbf{i}_n(i_i) - \theta_{o_j}))$*, and by the monotonically crescent characteristic of* $h(x)$ *we obtain* $h(\sum_{i=1}^{p}(W_{o_j i_i}\mathbf{i}_m(i_i) - \theta_{o_j})) \leq h(\sum_{i=1}^{p}(W_{o_j i_i}\mathbf{i}_n(i_i) - \theta_{o_j}))$*, i.e.,* $o_j(\mathbf{i}_m) \leq o_j(\mathbf{i}_n)$*. That completes the proof.*
□

Corollary 39 *(*BNS *Pruning Rule 1) Let* $\mathbf{i}_m \leq_\mathbf{I} \mathbf{i}_n$*. If* $\mathbf{i}_n$ *does not satisfy the constraint* $\mathbf{C}o_j$ *on the* BNSs *output neuron, then* $\mathbf{i}_m$ *does not satisfy* $\mathbf{C}o_j$ *either.*
Proof. *Directly from Proposition 38.*

Corollary 40 *(*BNS *Pruning Rule 2) Let* $\mathbf{i}_m \leq_\mathbf{I} \mathbf{i}_n$*. If* $\mathbf{i}_m$ *satisfies the constraint* $\mathbf{C}o_j$ *on the* BNSs *output neuron, then* $\mathbf{i}_n$ *also satisfies* $\mathbf{C}o_j$*.*
Proof. *Directly from Proposition 38.*

The particular characteristic of *BNSs*, specifically because they have no hidden neurons, allows us to define a new ordering that can be very useful in helping to reduce the *BNS's* input vectors search space. Briefly, if now, in addition, we consider the *BNS* weights' values, we may be able to assess, given two input vectors $\mathbf{i}_n$ and $\mathbf{i}_m$ such that $\langle \mathbf{i}_n \rangle = \langle \mathbf{i}_m \rangle$, whether $o_j(\mathbf{i}_m) \leq o_j(\mathbf{i}_n)$ or not[13]. Assume, for instance, that $\mathbf{i}_m$ and $\mathbf{i}_n$ differ only on inputs i_i and i_k, where $i_i = 1$ in $\mathbf{i}_n$ and $i_k = 1$ in $\mathbf{i}_m$. Thus, if $|W_{o_j i_i}| \leq |W_{o_j i_k}|$, it is not difficult to see that $o_j(\mathbf{i}_n) \leq o_j(\mathbf{i}_m)$. Let us formalize this idea.

Proposition 41 *(*BNS *Pruning Rule 3) 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* $\langle \mathbf{i}_m \rangle, \langle \mathbf{i}_n \rangle <$

[13]Recall that, previously, two input vectors $\mathbf{i}_n$ and $\mathbf{i}_m$ such that $\langle \mathbf{i}_n \rangle = \langle \mathbf{i}_m \rangle$ were incomparable.

$\langle \mathbf{i}_o \rangle$, *that is, both* $\mathbf{i}_m$ *and* $\mathbf{i}_n$ *are immediate predecessors of* $\mathbf{i}_o$. *Let* $\mathbf{i}_m$ *be obtained from* $\mathbf{i}_o$ *by flipping the i-th input from 1 (resp.* A_{min} *for* Hidden to Output BNSs*) to -1, while* $\mathbf{i}_n$ *is obtained from* $\mathbf{i}_o$ *by flipping the k-th input from 1 (resp.* A_{min} *for* Hidden to Output BNSs*) to -1. If* $|W_{o_j i_k}| \leq |W_{o_j i_i}|$ *then* $o_j(\mathbf{i}_m) \leq o_j(\mathbf{i}_n)$. *In this case, we write* $\mathbf{i}_m \leq_{\langle\rangle} \mathbf{i}_n$.
Proof. *We know that both* $\mathbf{i}_m$ *and* $\mathbf{i}_n$ *are obtained from* $\mathbf{i}_o$ *by flipping, respectively, inputs* $\mathbf{i}_o(i)$ *and* $\mathbf{i}_o(k)$ *from 1 (resp.* A_{min}*) to -1. We also know that* $o_j(\mathbf{i}_o) = h(W_{o_j i_i}\mathbf{i}_o(i) + W_{o_j i_k}\mathbf{i}_o(k) + \Delta + \theta_{o_j})$, *where* $W_{ji} \in \Re^+$ *and* $A_{min} > 0$. *For* Input to Hidden BNSs, $o_j(\mathbf{i}_m) = h(-W_{o_j i_i} + W_{o_j i_k} + \Delta + \theta_{o_j})$ *and* $o_j(\mathbf{i}_n) = h(W_{o_j i_i} - W_{o_j i_k} + \Delta + \theta_{o_j})$. *For* Hidden to Output BNSs, $o_j(\mathbf{i}_m) = h(-W_{o_j i_i} + A_{min}W_{o_j i_k} + \Delta + \theta_{o_j})$ *and* $o_j(\mathbf{i}_n) = h(A_{min}W_{o_j i_i} - W_{o_j i_k} + \Delta + \theta_{o_j})$. *Since* $|W_{o_j i_k}| \leq |W_{o_j i_i}|$, *and from the monotonically crescent characteristic of* $h(x)$, *we obtain* $o_j(\mathbf{i}_m) \leq o_j(\mathbf{i}_n)$ *in both cases.* □

As before, a direct result of Proposition 41 is that: if $\mathbf{i}_m$ satisfies the constraint $\mathbf{C}_{o_j}$ on the *BNS* output neuron, then $\mathbf{i}_n$ also satisfies $\mathbf{C}_{o_j}$. By contraposition, if $\mathbf{i}_n$ does not satisfy $\mathbf{C}_{o_j}$ then $\mathbf{i}_m$ does not satisfy $\mathbf{C}_{o_j}$ either.

Proposition 42 *(*BNS *Pruning Rule 4) 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* $\langle \mathbf{i}_o \rangle < \langle \mathbf{i}_m \rangle, \langle \mathbf{i}_n \rangle$, *that is, both* $\mathbf{i}_m$ *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 (resp.* A_{min} *for* Hidden to Output BNSs*), while* $\mathbf{i}_n$ *is obtained from* $\mathbf{i}_o$ *by flipping the k-th input from -1 to 1 (resp.* A_{min} *for* Hidden to Output BNSs*). If* $|W_{o_j i_k}| \leq |W_{o_j i_i}|$, *then* $o_j(\mathbf{i}_n) \leq o_j(\mathbf{i}_m)$. *In this case, we write* $\mathbf{i}_n \leq_{\langle\rangle} \mathbf{i}_m$.
Proof. *This is the contrapositive of Proposition 41.* □

Example 43 *Consider the network of Figure 12(1) and its positive form at Figure 12(2). The network contains three* BNSs*; two* Input to Hidden BNSs, *having inputs* $\{a, b, c\}$ *and* $\{a, b, \sim c\}$, *and outputs* n_1 *and* n_2, *respectively, and one* Hidden to Output BNS, *having input* $\{n_1, \sim n_2\}$ *and output* x.

Applying the Transformation Algorithm on each BNS *and considering the ordering on set inclusion, we verify that* (abc) *is the maximum element of the* BNS *with output* n_1, $(ab \sim c)$ *is the maximum element of the* BNS *with output* n_2, *and* $(n_1 \sim n_2)$ *is the maximum element of the* BNS *with output* x.

If now we add information about the weights, we can apply Pruning Rules 3 and 4 as well. Take, for example, the positive form of the BNS *with output* n_1, *where* $W_{n_1 b} \leq W_{n_1 c} \leq W_{n_1 a}$. *Using Pruning Rules 3 and 4, we can obtain a new ordering on input vectors* $\mathbf{i}_m$ *and* $\mathbf{i}_n$ *when* $\langle \mathbf{i}_m \rangle = \langle \mathbf{i}_n \rangle$.[14] *We obtain* $(-1, 1, 1) \leq_{\langle\rangle} (1, 1, -1) \leq_{\langle\rangle} (1, -1, 1)$ *and* $(-1, 1, -1) \leq_{\langle\rangle} (-1, -1, 1) \leq_{\langle\rangle} (1, -1, -1)$. *Similarly, given* $W_{x \sim n_2} \leq W_{x n_1}$, *we obtain* $(\sim$

[14] Recall that such input vectors are incomparable under the set inclusion ordering.

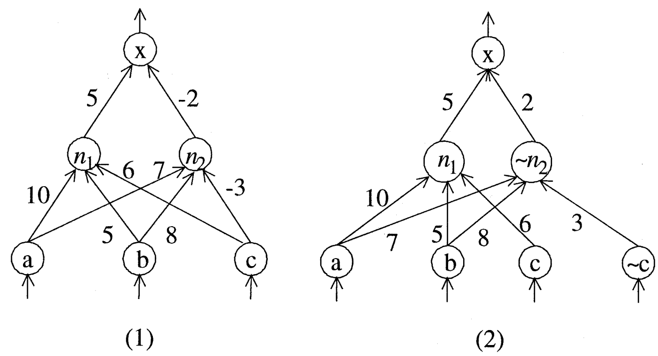


Figure 12: A non-regular network (1) and its positive form (2) obtained by applying the Transformation Algorithm on its *BNSs.*

$n_1, \sim n_2) \leq_{\langle\rangle} (n_1, n_2)$ *for the* Hidden to Output BNS[15]. *Figure 13 contains two diagrams in which this new ordering is superimposed on the previous set inclusion ordering for the* BNSs *with outputs* n_1*(1) and* x*(2).*

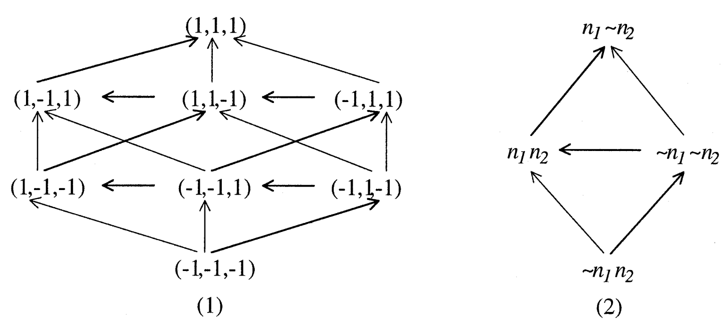


Figure 13: Adding information about the weights of the *BNSs* with output n_1 (1) and x (2).

The above example illustrates the ordering $\preceq$ on the set of input vectors $\mathbf{I}$ of *BNSs.* The ordering results from the superimposition of the ordering $\leq_{\langle\rangle}$, obtained from Pruning Rules 3 and 4, on the set inclusion ordering $\leq_{\mathbf{I}}$, obtained from Pruning Rules 1 and 2. Let us define $\preceq$ more precisely.

Definition 44 *Let* $\preceq$ *be a partial order on a* BNS's *input vectors set* $\mathbf{I}$. *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* $\mathbf{i}_m \leq_{\langle\rangle} \mathbf{i}_n$.

Back to Example 43 above, it is not difficult to see that the ordering $\preceq$ on the *BNS* with output n_1 is given by the diagram in Figure 14 below (see

[15]Here, we have deliberately used $\{n_i, \sim n_i\}$, instead of $\{1, -1\}$, to stress the fact that hidden neurons do not present discrete activations.

also Figure 13(1)). Incomparable elements in $\preceq$, as $\mathbf{i}_1 = (1,-1,-1)$ and $\mathbf{i}_2 = (-1,1,1)$ at Figure 14, indicate that it is not easy to establish whether $\mathbf{i}_1 \preceq \mathbf{i}_2$ without actually querying the *BNS* with both inputs. Note also that $\preceq$ is a chain for the *BNS* with output x, i.e., $(\sim n_1, n_2) \preceq (\sim n_1, \sim n_2) \preceq (n_1, n_2) \preceq (n_1, \sim n_2)$.

$$\begin{tikzcd}
 & (1,1,1) & \\
 & (1,\text{-}1,1) \arrow[u] & \\
 & (1,1,\text{-}1) \arrow[u] & \\
(1,\text{-}1,\text{-}1) \arrow[ur] & & (\text{-}1,1,1) \arrow[ul] \\
 & (\text{-}1,\text{-}1,1) \arrow[ul] \arrow[ur] & \\
 & (\text{-}1,1,\text{-}1) \arrow[u] & \\
 & (\text{-}1,\text{-}1,\text{-}1) \arrow[u] &
\end{tikzcd}$$

Figure 14: The ordering $\preceq$ on the input vectors set of the *BNS* with output n_1.

Figure 15 displays $\preceq$ on $\mathbf{I} = \wp(\mathcal{I})$ for $\mathcal{I} = \{a,b,c,d\}$, given $(1,1,1,1) = [a,b,c,d]$ and $|W_d| \leq |W_c| \leq |W_b| \leq |W_a|$. Note that $\preceq$ follows the ordering on $|W_a| + |W_b| + |W_c| + |W_d|$.

$\preceq$ provides a systematic way of searching the input vectors space. Let us illustrate this with the following example, which also gives a glance about the implementation of the extraction search process.

Example 45 *Consider the* Input to Hidden BNS *of Figure 16(1), and its positive form 16(2). The ordering's maximum element is input vector* $\mathbf{i}_\top = (1,1,1,1) = (a,b,\sim c,\sim d)$. *Taking the* BNS *of Figure 16(2), if* $\mathbf{i}_\top$ *does not activate* n_i *then we proceed to generate the elements* $\mathbf{i}_m$ *such that* $dist(\mathbf{i}_m, \mathbf{i}_\top) = 1$. *However, Pruning Rule 3 says that there is an ordering among elements* $\mathbf{i}_m$. *For example, it says that* $(1,1,1,-1) = (a,b,\sim c,d)$ *provides a smaller activation value to* n_i *than* $(1,1,-1,1) = (a,b,c,\sim d)$.

Therefore, given $W_{n_i \sim c} \leq W_{n_i a} \leq W_{n_i \sim d} \leq W_{n_i b}$, *we start from* $\mathbf{i}_\top$ *by flipping from 1 to -1 the input* $\sim c$ *with the smallest weight* $W_{n_i \sim c}$, *and obtain the input vector* $\mathbf{i}_1 = (1,1,-1,1)$. *By Pruning Rule 3, the activation of* n_i *given* $\mathbf{i}_1$ *is greater than the activation of* n_i *given any other element* $\mathbf{i}_m$ *such that* $\langle \mathbf{i}_m \rangle = \langle \mathbf{i}_1 \rangle$. *Thus, if* $n_i(\mathbf{i}_1) < A_{max}$ *then* $n_i(\mathbf{i}_m) < A_{max}$. *In this case, we could stop the search. Otherwise, we derive the rule* $abc \sim d \to n_i$,

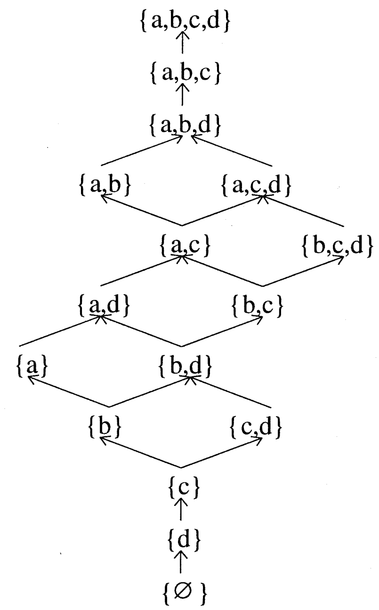


Figure 15: $\preceq$ on $\wp(\mathcal{I})$, given $\mathcal{I} = \{a, b, c, d\}$ and $(1, 1, 1, 1) = [a, b, c, d]$.

but we still need to generate and test the next element $\mathbf{i}_m$*, which is obtained from* $\mathbf{i}_\top$ *by flipping the input a with the next smallest weight.*

Similarly, let $\mathbf{i}_\perp = (-1, -1, -1, -1) = (\sim a, \sim b, c, d)$*. If* $\mathbf{i}_\perp$ *does not activate* n_i *then we flip from -1 to 1 the input* $\sim c$ *with the smallest weight* $W_{n_i \sim c}$*, and obtain the input vector* $\mathbf{i}_2 = (-1, -1, 1, -1)$*. By Pruning Rule 4, if* $n_i(\mathbf{i}_2) > A_{min}$ *then* $n_i(\mathbf{i}_n) > A_{min}$ *for all* $\mathbf{i}_n$ *such that* $\langle \mathbf{i}_n \rangle = \langle \mathbf{i}_2 \rangle$*. In this case, we could derive the rule* $1(a, b, \sim c, \sim d) \rightarrow n_i$*, and stop the search. Otherwise, we need to generate and test the next element* $\mathbf{i}_n$*, obtained from* $\mathbf{i}_\perp$ *by flipping the input a with the next smallest weight.*

A systematic way of searching the input vectors space is obtained as follows. Given the maximum element, we order it from left to right w.r.t the weights associated with each input, such that inputs with greater weights are on the left of inputs with smaller weights. In Example 45, we rearrange $(a, b, \sim c, \sim d)$ and obtain $(1, 1, 1, 1) = [b, \sim d, a, \sim c]$. The search proceeds by flipping the right most input, then the second right most input and so on. At distance 2 from $sup(\mathbf{I})$ and beyond, we only flip the inputs on the left of the left most -1 input. In this way, we avoid repeating input vectors. Figure 17 illustrates this process for the *BNS* of Example 45.

Similarly, starting from the minimum element, we rearrange $(\sim a, \sim b, c, d)$ and obtain $(-1, -1, -1, -1) = [\sim b, d, \sim a, c]$. Figure 18 illustrates the process for the *BNS* of Example 45. Now, at distance 2 from $inf(\mathbf{I})$ and beyond, we only flip the inputs on the left of the left most 1 input.

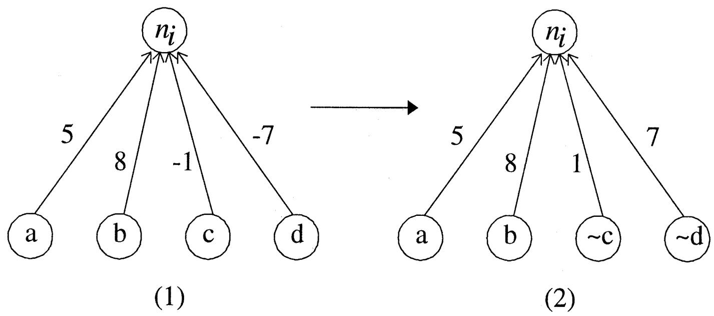


Figure 16: An *Input to Hidden BNS* (1), and its positive form (2).

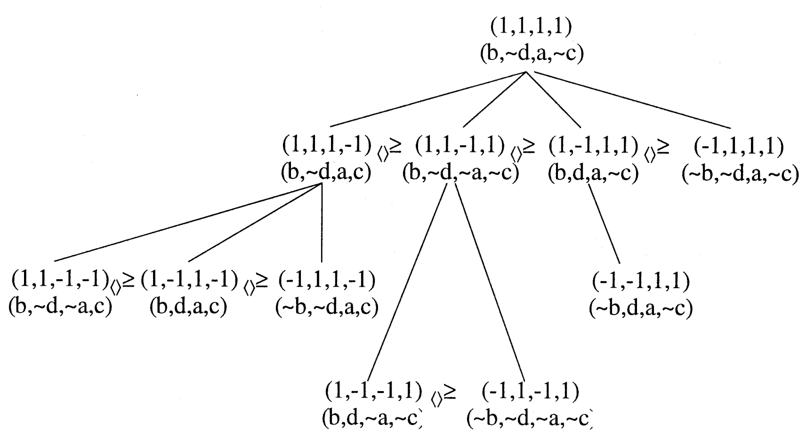


Figure 17: Systematically deriving input vectors from $\mathbf{i}_{\top}$ without repetitions.

Note the symmetry between Figures 17 and 18, reflecting, respectively, the use of Pruning Rules 3 and 4. Starting from $sup(\mathbf{I})$, flipping the input with the smallest weight results in the next greatest input, while from $inf(\mathbf{I})$, flipping the input with the smallest weight results in the next smallest input.

Let us now focus on the problem of knowledge extraction from *Hidden to Output BNSs*. The problem lies on the fact that hidden neurons do not present discrete activations $\{-1, 1\}$. Instead, they are said to be active if their activation values lie on the interval $(A_{min}, 1)$, or non-active if their activation values lie on the interval $(-1, A_{max})$. We need to provide, therefore, a special treatment for the knowledge extraction procedure from *Hidden to Output BNSs*.

We have seen that if we simply assume that hidden neurons are either fully active or non-active, then the extraction algorithm looses soundness. We are left with the option of trying to find an ordering on the hidden neurons ranges of activations $(-1, A_{max})$ and $(A_{min}, 1)$. But we realize that

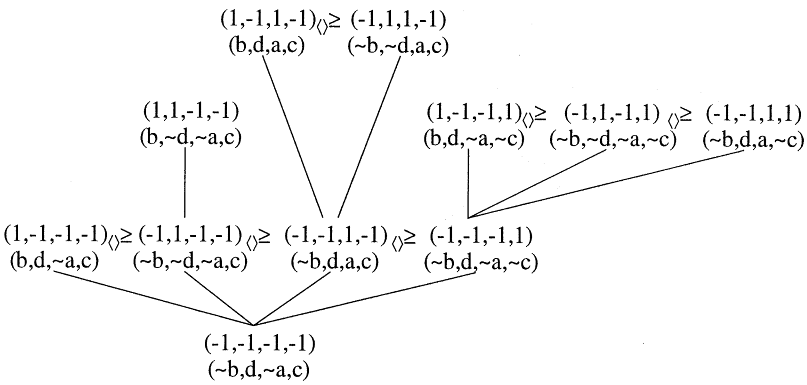


Figure 18: Systematically deriving input vectors from $\mathbf{i}_\perp$ without repetitions.

we can not define such an ordering easily. For example, we can not say that $(n_1 < A_{max})$ *and* $(n_2 < A_{max}) \preceq (n_1 < A_{max})$ *and* $(n_2 > A_{min})$. As a counter-example, simply take $A_{max} = -A_{min} = -0.2$ and note that $n_1 = -0.3$ *and* $n_2 = -0.3$ may provide a greater activation to an output neuron than $n_1 = -0.95$ *and* $n_2 = 0.25$.

At this stage, we need to compromise in order to keep soundness. Roughly, we have to analyse the activation values of the hidden neurons in the "worst cases". Those activations are given by -1 and A_{min} in the case of a hidden neuron connected through a positive weight to the output, and by A_{max} and 1 in the case of a hidden neuron connected through a negative weight to the output.

Example 46 *Consider the* Hidden to Output BNS *of Figure 19. The intuition behind its corresponding ordering is as follows: either both n_1 and n_2 present activations greater than A_{min}, or one of them presents activation greater than A_{min} while the other presents activation smaller than A_{max}, or both of them present activations smaller than A_{max}.*

Considering the worst cases activations, since the weights from n_1 and n_2 to x are both positive, if the activation of n_i is smaller than A_{max}, then we assume that it is -1. On the other hand, if the activation of n_i is greater than A_{min}, then we analyse the case where it is equal to A_{min}. In this way, we can derive the ordering of Figure 19 safely, as we show in the sequel. Similarly, if the weight from n_i to x is negative, then we take activation values A_{max} and 1.

Given $W_{xn_2} \leq W_{xn_1}$, we also obtain $(-1, A_{min}) \preceq (A_{min}, -1)$. As before, in this case $\preceq$ is a chain.

The recipe for performing a sound extraction from non-regular networks, concerning *Hidden to Output BNSs*, is: *If the weight from n_i to o_j is positive*

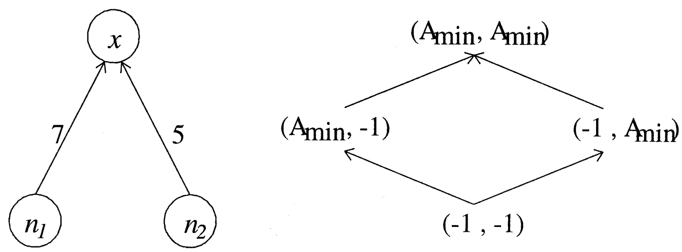


Figure 19: A *Hidden to Output BNS* and the corresponding set inclusion ordering on the hidden neurons activations in the worst case.

then assume $n_i = A_{min}$ *and* $\sim n_i = -1$. *If the weight from* n_i *to* o_j *is negative then assume* $n_i = 1$ *and* $\sim n_i = A_{max}$. These are the worst cases analyses, what means that we consider the minimal contribution of each hidden neuron to the activation of an output neuron.

Remark 2 *Note that when we consider that the activation values of hidden neurons are either positive in the interval* $(A_{min}, 1)$ *or negative in the interval* $(-1, A_{max})$, *we assume, without loss of generality, that the network's learning algorithm is such that no hidden neuron presents activation in the range* $[A_{max}, A_{min}]$ *(see [4]). Note that one can always assume* $A_{max} = A_{min} \simeq 0$.

In the sequel, we exemplify how to obtain the ordering on a *Hidden to Output BNS* with two input neurons n_1 and n_2, connected to an output neuron x with positive and negative weights.

We start by applying the Transformation Algorithm. We obtain the *BNS's* positive form and check the labels of its input neurons (the network's hidden neurons). If they are labeled n_1 and n_2 ($sup(\mathbf{I}) = (n_1, n_2)$) then the weights from both of them to x are positive. Thus, we assume that $\sim n_i = -1$ and $n_i = A_{min}$ for $i = \{1, 2\}$. As a result, we derive the ordering of Figure 20($Case$ 1). If, however, the Transformation Algorithm tells us that $sup(\mathbf{I}) = (n_1, \sim n_2)$ then we consider $\sim n_1 = -1$ and $n_1 = A_{min}$ for the activation values of n_1, and $\sim n_2 = A_{max}$ and $n_2 = 1$ for the activation values of n_2. Figure 20($Case$ 2) shows the ordering obtained if $sup(\mathbf{I}) = (n_1, \sim n_2)$. Finally, if $sup(\mathbf{I}) = (\sim n_1, \sim n_2)$, we assume that $\sim n_i = A_{max}$ and $n_i = 1$ for $i = \{1, 2\}$, as shown in Figure 20($Case$ 3). If, in addition, we have $\left|W_{o_j n_2}\right| \leq \left|W_{o_j n_1}\right|$, we also obtain $(A_{min}, -1) \leq_{\langle\rangle} (-1, A_{min})$ in Figure 20(Case 1), $(A_{min}, 1) \leq_{\langle\rangle} (-1, A_{max})$ in 20(Case 2), and $(A_{max}, 1) \leq_{\langle\rangle} (1, A_{max})$ in 20(Case 3). Thus, the resulting orders $\preceq$ are chains, as expected. Note that the orders of Figure 20 are valid for the original *BNSs*, and not for their positive forms.

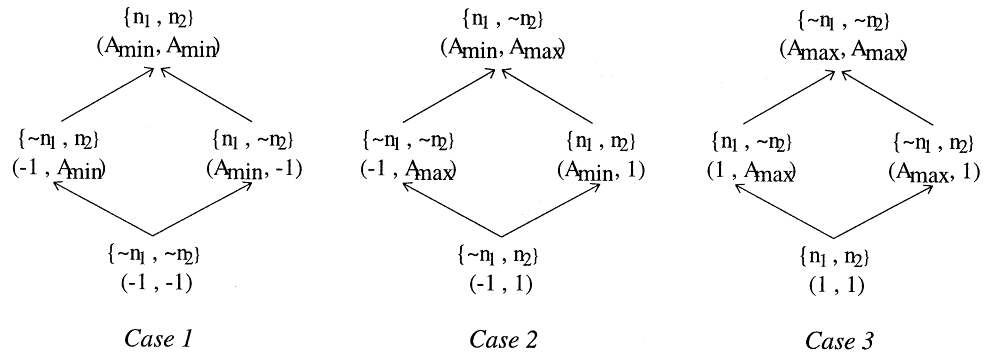


Figure 20: Orderings on *Hidden to Output BNSs* with two input neurons n_1 and n_2, using worst case analyses on $(-1, A_{max})$ and $(A_{min}, 1)$.

Let us now see if we can define a mapping for *Hidden to Output BNSs,* analogous to the mapping σ for Regular Networks and *Input to Hidden BNSs.* In fact, if we assume, without loss of generality, that $A_{max} = -A_{min}$ then the same function σ mapping input vectors of the positive form into input vectors of the *BNS* can be used here. Let $i_i \in \{-1, A_{min}\}$, $i'_i \in \{-1, -A_{min}, A_{min}, 1\}$, $x_i \in \mathcal{I}_+, 1 \leq i \leq p$. Recall that $\sigma_{[x_1,...,x_p]}(i_1, ..., i_p) = (i'_1, ..., i'_p)$, where $i'_i = i_i$ if x_i is a positive literal and $i'_i = -i_i$ otherwise. For example, $\sigma_{[a,\sim b,c,\sim d]}(A_{min}, A_{min}, -1, -1) = (A_{min}, -A_{min}, -1, 1)$. The following example illustrates the use of σ for *Hidden to Output BNSs.*

Example 47 *Given $\sigma_{[n_1,\sim n_2,n_3]}(A_{min}, A_{min}, A_{min}) = (A_{min}, A_{max}, A_{min})$, we obtain the ordering of Figure 21(b). From $n_1 = A_{min}$, $\sim n_2 = A_{max}$ and $n_3 = A_{min}$, we obtain $\sim n_1 = -1$, $n_2 = 1$ and $\sim n_3 = -1$ at Figure 21(a). As before, the extraction process can be carried out by querying the* BNS's *positive form with values $\{-1, A_{min}\}$, following 21(b). In this way, the only difference from* Input to Hidden BNSs *is that input values* 1 *should be replaced by A_{min} (see Figures 17 and 18).*

We are finally in position to present the extraction algorithm extended for non regular networks.

- *Knowledge Extraction Algorithm - General Case*

1. Split the neural network N into *BNSs*;
2. For each *BNS* $\mathcal{B}_i$ $(1 \leq i \leq r + q)$ do:
   (a) Apply the *Transformation Algorithm* and find its positive form $\mathcal{B}_i^+$;

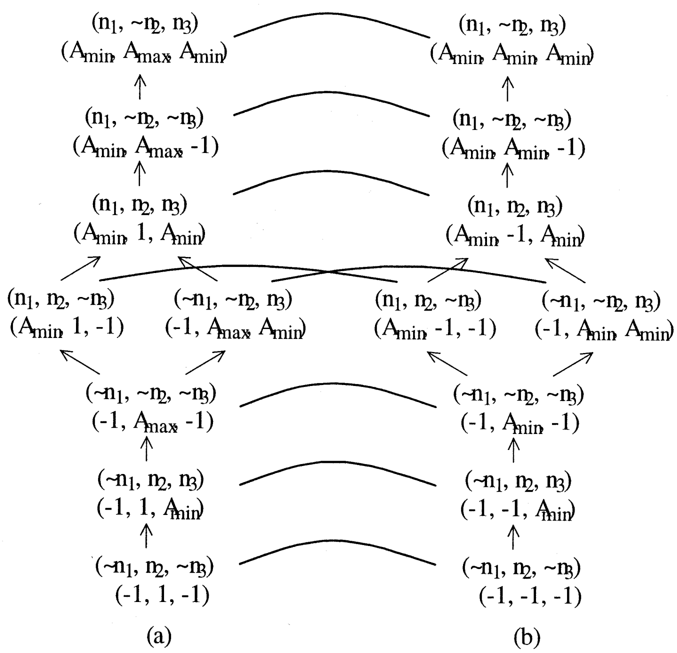


Figure 21: (a) $\preceq$ on a *Hidden to Output BNS* with three input neurons (n_1, n_2, n_3) and the associated activations in the worst case; (b) $\preceq$ on the *BNS's* positive form and the mapping σ from (b) and (a).

(b) Order $\mathcal{I}_+$ according to the weights associated with each input of $\mathcal{B}_i^+$;

(c) If $\mathcal{B}_i^+$ is an *Input to Hidden BNS*, take $i_i \in \{-1, 1\}$;

(d) If $\mathcal{B}_i^+$ is a *Hidden to Output BNS*, take $i_i \in \{-1, A_{min}\}$;

(e) Find $Inf(\mathbf{I})$ and $Sup(\mathbf{I})$ w.r.t $\mathcal{B}_i^+$, using σ;

(f) Call the *Knowledge Extraction Algorithm for Regular Networks*, step 3, where $N_+ := \mathcal{B}_i^+$;

/* *Recall that, now, we have to replace Search Space Pruning Rules 1 and 2, respectively, by BNS Pruning Rules 1 and 2.*

/* *We also need to add the following lines to the extraction algorithm for regular networks (step 3d):*

- If BNS *Pruning Rule 4* is applicable, stop generating the successors of $\mathbf{i}_\perp$;
- If BNS *Pruning Rule 3* is applicable, stop generating the predecessors of $\mathbf{i}_\top$;

3. Assemble the final Rule Set of N.

In what follows, we describe in detail step 3 of the above algorithm, and discuss the problems resulting from the worst case analysis of *Hidden to Output BNSs.*

5.3 Assembling the Final Rule Set

Steps 1 and 2 of the general case extraction algorithm generate local information about each hidden and output neuron. In step 3, such information needs to be carefully combined, in order to derive the final set of rules of N. We use n_i and $\sim n_i$ to indicate, respectively, that the activation of hidden neuron n_i is greater than A_{min} or smaller than A_{max}. Bear in mind, however, that hidden neurons n_i do not have concepts directly associated to them. Thus, the task of assembling the final rule set is that of relating the concepts in the network's input layer directly to the ones in its output layer, removing n_i from the rule set. The following Lemma 48 will serve as basis for this task.

Lemma 48 *The extraction of rules from* Input to Hidden BNSs *is sound and complete.*
Proof. *From Proposition 37 and Theorem 33, we obtain soundness of the rule set. From Proposition 37 and from Theorem 34 we obtain completeness of the rule set.* □

Lemma 48 allows us to use the *completion* of rules extracted from *Input to Hidden BNSs* to assemble the network's rule set, i.e., it allows an extracted rule of the form $X_1, ..., X_p \rightarrow L_j$ to be substituted by the stronger $X_1, ..., X_p \leftrightarrow L_j$. For example, assume that the extraction algorithm derives $a \rightarrow n_1$ from *BNS* $\mathcal{B}_1$ and $b \sim c \rightarrow n_2$ from *BNS* $\mathcal{B}_2$. By Lemma 48, we have $a \leftrightarrow n_1$ and $b \sim c \leftrightarrow n_2$. By contraposition, we have $\sim a \leftrightarrow \sim n_1$ from $\mathcal{B}_1$, and $\sim b \vee c \leftrightarrow \sim n_2$ from $\mathcal{B}_2$. Now that we have the necessary information regarding the activation values of n_1 and n_2, assume that we have derived the rule $n_1 \sim n_2 \rightarrow x$ from *Hidden to Output BNS* $\mathcal{B}_3$. We know that $a \rightarrow n_1$ and $\sim b \vee c \rightarrow \sim n_2$. As a result, we may assemble the final rule set w.r.t output x: $\{a \sim b \rightarrow x,\ ac \rightarrow x\}$.

The following example illustrates how to assemble the final rule set in a sound mode. It also illustrates the incompleteness of the general case extraction.

Example 49 *Consider a neural network N with two input neurons a and b, two hidden neurons n_1 and n_2 and one output neuron x. Assume that the set of weights is such that the activations on the table below are obtained for each input vector.*

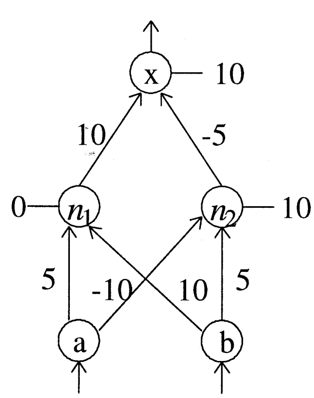
a	b	n_1	n_2	x
-1	-1	$< A_{max}$	$< A_{max}$	$< A_{max}$
-1	1	$> A_{min}$	$> A_{min}$	$< A_{max}$
1	-1	$< A_{max}$	$< A_{max}$	$< A_{max}$
1	1	$> A_{min}$	$< A_{max}$	$> A_{min}$

An exhaustive pedagogical extraction algorithm, although inefficiently, would derive the unique rule $ab \rightarrow x$ *from* N. *That is because* $(1, 1)$ *is the only input vector that activates* x. *A decompositional approach, on the other hand, would split the network into its* BNSs. *Since (-1,1) and (1,1) activate* n_1, *the rules* $\sim ab \rightarrow n_1$ *and* $ab \rightarrow n_1$ *would be derived, and hence* $b \rightarrow n_1$. *Similarly, the rule* $\sim ab \rightarrow n_2$ *would be derived, since (-1,1) also activates* n_2.

Taking $A_{min} = 0.5$, *assume that given* $(ab) = (-1, 1)$, *the activation values of* n_1 *and* n_2 *are, respectively, 0.6 and 0.95. By assuming that* n_1 *and* n_2 *are either fully active or non active, we could wrongly derive the rule* $n_1n_2 \rightarrow x$ *(unsoundness). To solve this problem, we take the worst case activations of hidden neurons* $n_1 = A_{min}$ *and* $n_2 = A_{min}$.

However, given $(ab) = (1, 1)$, *assume that the activation values of* n_1 *and* n_2 *are, respectively, 0.9 and -0.6. Now, if we consider the worst case activations,* $n_1 = A_{min}$ *and* $n_2 = -1$, *it may be the case that we do not derive the rule* $n_1 \sim n_2 \rightarrow x$ *(incompleteness) as expected.*

Finally, assume that we have managed to derive the rule $n_1 \sim n_2 \rightarrow x$ *from the* Hidden to Output BNS *of* N.[16] *The final rule set can be assembled as follows: by Lemma 48, we derive* $b \leftrightarrow n_1$ *and* $\sim ab \leftrightarrow n_2$, *and together with* $n_1 \sim n_2 \rightarrow x$ *we obtain* $b \wedge (a \vee \sim b) \rightarrow x$. *As a result, the final rule set is* $ab \rightarrow x$, *in accordance with the exhaustive pedagogical extraction process. A neural network that presents the activations used in this example is given below.*



Lemma 50 *The extraction of rules from* Hidden to Output BNSs *is sound.*

[16] Possibly by fine-tuning the value of A_{min} in the extraction algorithm.

Proof. *If we are able to derive a rule r taking $n_i \in \{-1, A_{min}\}$ then, from the mononoincally crescent characteristic of $h(x)$, r will still be valid if $n_i \in \{[-1, -A_{min}], [A_{min}, 1]\}, A_{min} > 0$.* $\square$

Theorem 51 *The extraction algorithm for non-regular networks is sound.*
Proof. *Directly from Lemmas 48 and 50.*

Theorem 52 *The extraction algorithm for non-regular networks is incomplete.*
Proof. *We give a counter-example. Let $\mathcal{B}$ be a* Hidden to Output BNS *with input n_1 and output x. Let $\beta = 1$, $W_{xn_1} = 1$, $\theta_x = 0.1$. Assume $A_{min} = 0.4$. Given $i_1 = 1$, we obtain $o_x = 0.42$, i.e., $n_1 \rightarrow x$. Taking $i_1 = A_{min}$, we have $o_x = 0.15$ and thus we have lost $n_1 \rightarrow x$.* $\square$

As far as efficiency is concerned, one can apply the extraction algorithm until a predefined number of input vectors is queried, and then test the accuracy of the set of rules derived against the accuracy of the network. If, for instance, in a particular application, the set of rules obtained classifies correctly, say, 95% of the training and testing examples correctly classified by the network, then one could stop the extraction process.

6 Experimental Results

We have used three application domains in order to test the extraction algorithm: the MONK's problems [36], DNA sequences analysis [4, 12, 35, 39], and Power Systems FAULT DIAGNOSIS [3]. In this section, we briefly describe each problem and present the results of the extraction algorithm. We also compare the results obtained in DNA sequences analysis with those obtained in [12, 35, 39].

The extraction system consists of three modules: the main one takes a trained neural network (its set of weights and activation function), searches the input vectors space and generates the network's rule set accordingly, another one simplifies the rule set, and yet another checks the rule set accuracy against that of the network, given a test set, and its fidelity to the network. The system was implemented in ANSI **C** (5K lines of code) and is available upon request. Implementation details will be discussed in another paper. We start by presenting two very simple examples which will help the reader to recall the sequence of operations contained in the extraction process.

Example 53 *(The* XOR *Problem) A network with p input neurons, q hidden neurons and r output neurons contains q* Input-to-Hidden BNSs, *each with p inputs and a single output, and r* Hidden-to-Output BNSs, *each with q inputs and a single output. To each* BNS *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* BNS. *For example, in Figure 22,* N_1 *and* N_2 *are the positive forms of the* Input-to-Hidden BNSs *of* N*, while* N_3 *is the positive form of the* Hidden-to-Output BNS *of* N*. We then define the function* σ *mapping input vectors of the positive form into input vectors of the* BNS. *For example, for* N_1 $\sigma_{[a,\sim b]}(1,1) = (1,-1)$.

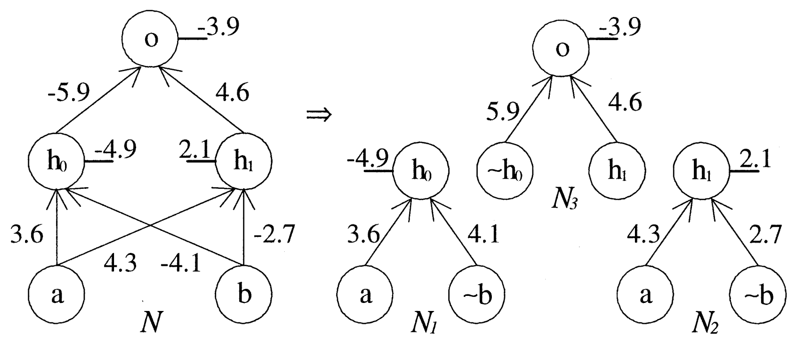


Figure 22: The network N, having tanh as activation function, computes $\overline{XOR}$. We will extract rules for h_0, h_1 and o by querying N_1, N_2 and N_3, respectively, and then assemble the rule set for N.

Given a $2-$*ary input vector,* $\preceq$ *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* $(1,1) = [a, \sim b]$ *in both. Querying* N_1, h_0 *is active for* $(1,1)$ *only. Thus, by applying* σ *we derive* $a \sim b \rightarrow h_0$. *Querying* 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* $\sim a$. *From* $ab \rightarrow h_1$ *and* $a \sim b \rightarrow h_1$ *we obtain* $a \rightarrow h_1$.

Considering now Hidden to Output BNSs, *it is usually assumed 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 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*[17]. *For* N_3 *we have* $(-1,-1) \preceq (-1, A_{min}) \preceq (A_{min}, -1) \preceq (A_{min}, A_{min})$*, where* $(A_{min}, A_{min}) = [\sim h_0, h_1]$ *and* $A_{min} = 0.5$. *Only* (A_{min}, A_{min}) *activates* o*, and we derive the rule* $\sim h_0 h_1 \rightarrow o$.

[17] 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.

Finally, to assemble the rule set of N, we take the completion *of each rule extracted from* Input to Hidden BNSs. *We have $a \sim b \rightarrow h_0$, $a \rightarrow h_1$, $\sim b \rightarrow h_1$ and $\sim h_0 h_1 \rightarrow o$. And from $a \sim b \leftrightarrow h_0$ and $a \vee \sim b \leftrightarrow h_1$ we obtain $(\sim a \vee b) \wedge (a \vee \sim b) \rightarrow o$; the $\overline{\text{XOR}}$ function.*

Example 54 *(*EXACTLY 1 OUT 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 neuron fires iff exactly one of its inputs fires. Although this is a very simple network, it is not straightforward to verify, by inspecting its weights, that it computes* exactly *1 out of $\{a, b, c, d, e\}$.*

Assume the following order on the weights linking the input layer to each hidden neuron h_o and h_1: $|W_{h_0d}| \leq |W_{h_0e}| \leq |W_{h_0c}| \leq |W_{h_0a}| \leq |W_{h_0b}|$ and $|W_{h_1d}| \leq |W_{h_1e}| \leq |W_{h_1a}| \leq |W_{h_1c}| \leq |W_{h_1b}|$. We split the network into its BNSs *and apply the extraction algorithm. Taking $\mathcal{I} = [a, b, c, d, e]$ for the* BNS *with output h_0, we find out that input $(-1, -1, -1, 1, -1)$ activates h_0, by querying the* BNS. *Since $|W_{h_0d}|$ is the smallest weight, from the ordering $\preceq$ on $\mathbf{I}$ and by applying Definitions 26 and 28, we derive the rule* $1(abcde) \rightarrow h_0$. *Note that, by Definition 25, this rule subsumes $m(abcde) \rightarrow h_0$, for* $m > 1$. *Taking again $\mathcal{I} = [a, b, c, d, e]$ but now for the* BNS *with output h_1, we find out that input $(-1, -1, -1, 1, 1)$ activates h_1. Similarly, from the ordering $\preceq$ on $\mathbf{I}$ and by applying Definitions 26 and 28, we derive the rule* $2(abcde) \rightarrow h_1$. *Finally, for the* Hidden to Output BNS, $\mathcal{I} = [h_0, \sim h_1]$. *Taking $A_{min} = 0.5$, o is only activated by (A_{min}, A_{min}) and we derive the rule $h_0 \sim h_1 \rightarrow o$.*

In order to obtain the rule mapping inputs $\{a, b, c, d, e\}$ directly into the output $\{o\}$, we take the completion *of the rules extracted from* Input to Hidden BNSs: $1(abcde) \leftrightarrow h_1$ *and* $2(abcde) \leftrightarrow h_2$. *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. As a result, a network with a single hidden neuron would not be able to learn such a rule.*

For each application below we investigate three parameters: the *accuracy* of the rule set against that of the network w.r.t a test set, the *fidelity* of the rule set to the network, i.e., its ability to mimic the network's behavior, and the *readability* of the rule set in terms of its size.

6.1 The MONK's Problems

As a point of departure for testing, we applied the extraction algorithm to the Monk's problems [36]: three examples which have been used as benchmark for performance comparison between a range of symbolic and connectionist machine learning systems. Briefly, in the Monk's 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 [36], 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 back-propagation learning algorithm [6, 32]. All networks have been trained for 5,000 epochs, with an epoch being defined as one pass through the whole training set. Differently from Thrun, we use bipolar activation function, inputs in the set $\{-1, 1\}$, and $A_{min} = 0$ (See [4] for the motivation behind this).

For Problems 1, 2 and 3, respectively, the networks' performance w.r.t their test sets was 100%, 100% and 93.2%, while the accuracy of the rule sets for the same test sets was 100%, 99.2% and 93.5%. The fidelity of the rule sets to the networks was 100%, 99.2% and 91%. Figure 23 displays the accuracy of the network, the accuracy of the rule set and the fidelity of the rule set to the network grouped for each problem.

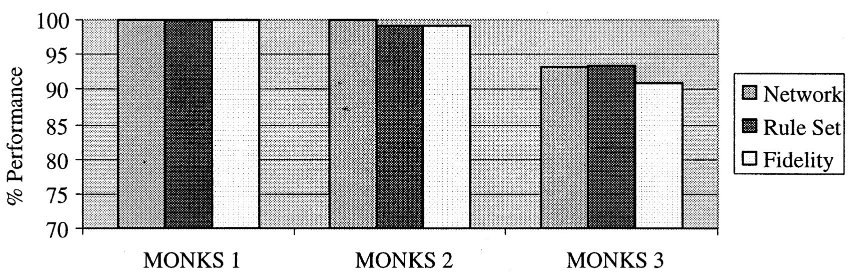


Figure 23: The accuracy of the network, the accuracy of the extracted rule set and the fidelity of the rule set to the network w.r.t the test sets of the Monk's Problems 1, 2 and 3, respectively.

The accuracy of the rule sets is very similar to that of the networks. In Problem 1, the rule set matches exactly the behavior of the network. In Problem 2, the rule set fails to classify correctly two examples, and in Problem 3 the rule set classifies correctly one example wrongly classified by the network. Such diferences are due to the incompleteness of the extraction algorithm.

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 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. In general, less than 30% of the set of input vectors is queried and, among these, less than 50% generate rules.

MONKS 1	Input Vectors	Queried	Extracted	Simplified
h_0	131072	18724	9455	2633
h_1	131072	18598	9385	536
h_2	131072	42776	21526	1793
o	8	8	2	1

MONKS 1: The number of input vectors queried, rules extracted, and rules remaining after simplification.

MONKS 2	Input Vectors	Queried	Extracted	Simplified
h_0	131072	131070	58317	18521
h_1	131072	43246	21769	5171
o	4	4	1	1

MONKS 2: The number of input vectors queried, rules extracted, and rules remaining after simplification.

MONKS 3	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

MONKS 3: The number of input vectors queried, rules extracted, and rules remaining after simplification.

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 below. For short, we name each attribute value with a letter from a to q in the sequence presented above, such that $a = (head\_shape = round), b = (head\_shape = square)$, and so on. We also use the *Integrity Constraints* of the Monk's

Problems in order to present a clearer rule set. For example, we do not present derived rules where *has_tie* = *yes* and *has_tie* = *no* simultaneously.

$\sim \mathrm{h}_1 \sim \mathrm{h}_2 \rightarrow o$	
$\sim abcd \sim e \rightarrow \mathrm{h}_1$	$a \sim b \sim dek \sim l \rightarrow \mathrm{h}_2$
$bd \sim e \sim l \rightarrow \mathrm{h}_1$	$ac \sim dem \sim q \rightarrow \mathrm{h}_2$
$b \sim i \sim lmn \rightarrow \mathrm{h}_1$	$a \sim b \sim def \sim l \rightarrow \mathrm{h}_2$
$bcd(\sim l \vee \sim ef) \rightarrow \mathrm{h}_1$	$ae \sim gjm(n \vee o) \rightarrow \mathrm{h}_2$
$b \sim ef(mn \vee mo) \rightarrow \mathrm{h}_1$	$\sim be \sim g \sim ln(a \vee \sim d) \rightarrow \mathrm{h}_2$
$\sim abdf(\sim l \vee m \vee n) \rightarrow \mathrm{h}_1$	$a \sim b \sim de \sim l(c \vee \sim h) \rightarrow \mathrm{h}_2$
$mno(\sim l \vee b \sim e \vee d \sim e \vee$	$\sim b \sim de \sim g \sim l(m \vee o) \rightarrow \mathrm{h}_2$
$bc \vee cd \vee \sim ab \vee bf) \rightarrow \mathrm{h}_1$	$a \sim b \sim de \sim l(j \vee p \vee i) \rightarrow \mathrm{h}_2$
$1(mno) \wedge (bd \sim e \vee bd \sim l \vee bcdf \vee$	$a \sim be \sim l \sim q(\sim d \vee m) \rightarrow \mathrm{h}_2$
$b \sim ef \sim l \vee \sim abcd \vee \sim ab \sim e \sim l \vee$	$a \sim be \sim g \sim l(\sim d \vee m \vee o) \rightarrow \mathrm{h}_2$
$bc \sim e \sim l \vee cd \sim e \sim l) \rightarrow \mathrm{h}_1$	$aem(\sim gn \sim p \vee \sim go \sim p \vee \sim hkn \vee$
	$\sim hko) \rightarrow \mathrm{h}_2$
	$1(mno) \wedge (a \sim de \sim h \vee a \sim de \sim g \vee$
	$a \sim de \sim l \vee a \sim b \sim de \vee ac \sim def \vee$
	$a \sim b \sim df \sim l \vee \sim b \sim def \sim l \vee$
	$a \sim bef \sim l \vee a \sim b \sim d \sim g \sim l \vee$
	$a \sim be \sim h \sim l \vee a \sim b \sim d \sim h \sim l \vee$
	$\sim b \sim de \sim h \sim l \vee a \sim bce \sim l \vee$
	$a \sim bc \sim d \sim l \vee \sim bc \sim de \sim l) \rightarrow \mathrm{h}_2$

Rules extracted for the Monk's Problem 1.

By looking at the rule set extracted and the much simpler description of Monk's Problem 1, it is clear that neural networks do not learn rules in a simple and structured way. Instead, they use a complex and redundant way of implementing rules. Not surprisingly, such a redundant representation is responsible for the network's robustness.

It is interesting that because the rule obtained for the *Hidden-to-Output BNS* of Monk's Problem 1 was $\sim \mathrm{h}_1 \sim \mathrm{h}_2 \rightarrow o$, 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 generalised, given the rule set and the classification task learned.

6.2 DNA Sequence Analysis

Molecular Biology is an area of increasing interest for computational learning systems analysis and application. Specifically, DNA sequence analysis problems have recently become a benchmark for learning systems' performance comparison. We apply the extraction algorithm on eukaryotes promoter recognition and prokaryotes splice junction determination, which are very large real world problems. Differently from the Monk's Problems, now an exhaustive pedagogical extraction (sound and complete) turns out to be impossible due to the large number of input neurons: the networks trained in both problems contain more than 200 input neurons.

In what follows we briefly introduce the problems in question from a computational application perspective (see [40] for a proper treatment on the subject). A DNA molecule contains two strands that are linear sequences of nucleotides. The DNA is composed from four different nucleotides - *adenine, guanine, thymine, and cytosine* - which are abbreviated by a, g, t, c, respectively. Some sequences of the DNA strand, called genes, serve as a blueprint for the synthesis of proteins. Interspersed among the genes are segments, called non-coding regions, that do not encode proteins.

Following [39], we use a special notation to identify the location of nucleotides in a DNA sequence. Each nucleotide is numbered with respect to a fixed, biologically meaningful, reference point. For example, "*@3 atcg*" states the location relative to the reference point in the DNA, followed by the sequence of symbols that must occur, i.e., an a must appear three nucleotides to the right of the reference point, followed by a t four nucleotides to the right of the reference point and so on. By convention, location zero is not used, and '$*$' indicates that any nucleotide will suffice in a particular location. Each location is encoded in the network by four input neurons, representing nucleotides a, g, t and c, in this order. Figure 24 shows part of the network for promoter recognition. Suppose that input vectors with $@-1g = 1$, $@1\ c = 1$ and $@5\ t = 1$ activate the output Promoter. We want to extract a rule of the form $@-1\ gc***\ t \rightarrow Promoter$.

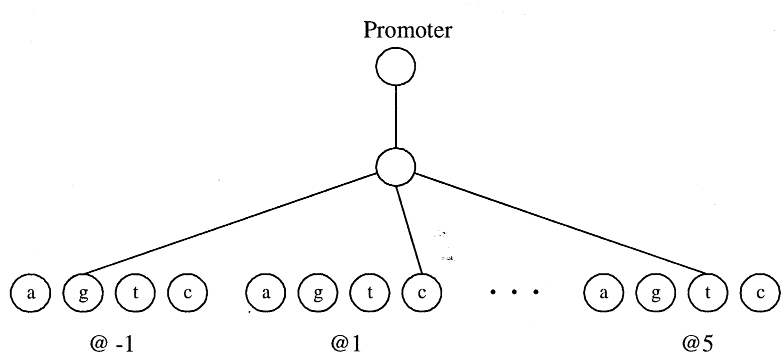


Figure 24: Part of the network for Promoter Recognition.

The first application is the prokaryotic[18] promoter recognition. Promoters are short DNA sequences that precede the beginning of genes. The aim

[18]Prokaryotes are single-celled organisms that do not have a nucleus, e.g. E. Coli.

of "*promoter recognition*" is to identify the starting location of genes in long sequences of DNA. The network's input layer for this task contains 228 neurons (57 consecutive DNA nucleotides), its single hidden layer contains 16 neurons, and its output neuron is responsible for classifying the DNA sequence as promoter or nonpromoter. The training examples consist of 48 promoter and 48 nonpromoter DNA sequences, while the test set contains only 10 examples.

The second application is eukaryotic[19] splice-junction determination. Splice-junctions are points on a DNA sequence at which the non-coding regions are removed during the process of protein synthesis. The aim of "*splice-junction determination*" is to recognize the boundaries between the part of the DNA retained after splice - called exons - and the part that is spliced out - the introns. The task consists therefore of recognizing exon/intron (E/I) boundaries and intron/exon (I/E) boundaries. Each example is a DNA sequence with 60 nucleotides (240 input neurons), where the center is the reference point. The network contains 26 neurons in its single hidden layer, while two output neurons are responsible for classifying the DNA sequences into E/I or I/E. The third category (neither E/I nor I/E) is considered true when neither output neurons are active. The training set for this task contains 1000 examples, in which approximately 25% are of I/E boundaries, 25% are of E/I boundaries and the remaining 50% are neither. We use a test set with 100 examples.

Figure 25 displays the accuracy of the network, the accuracy of the rule set and the fidelity of the rule set to the network for the promoter recognition and splice junction determination problems. Note that for the splice junction problem we should not evaluate each output neuron individually. Instead, the combined activations {1,-1} indicate E/I, {-1,1} indicate I/E, {-1,-1} indicate neither, and {1,1} are inconsistent.

In both applications, due to the intractability of the set of input vectors (2^{228} and 2^{240} elements each), we limit the maximum number of rules generated to 50,000 per hidden neuron. We also speed up the search process by doing the following: we jump, in a kind of binary search, from the ordering's minimum element to a new minimal element in the frontier at which input vectors start to generate rules[20].

The results obtained for the Promoter problem do not have statistical significance due to the reduced number of examples available for testing. However, the accuracy of the set of rules w.r.t the network's training set was

[19] Unlike prokaryotic cells, eukaryotic cells contain a nucleus, and so are higher up the evolutionary scale.

[20] Instead of searching from the ordering's maximum and minimum elements, we pick an input vector at distance $n/2$ from them, where n is the number of input neurons, and query it. If it activates the output then it becomes a new maximal element; otherwise, it becomes a new minimal element. We carry on with this process until maximal and minimal elements are at distance 1 from each other.

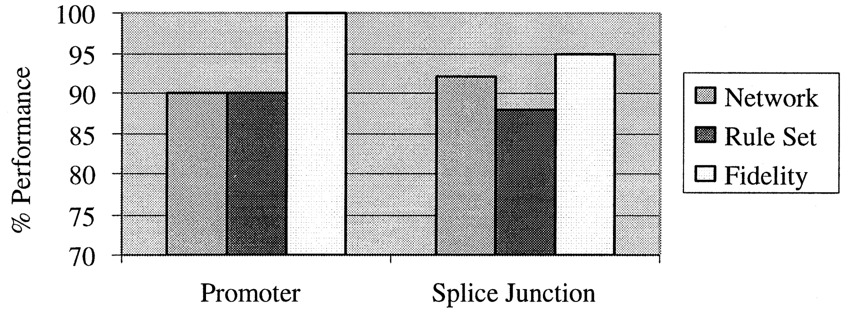


Figure 25: The accuracy of the network, the accuracy of the rule set and the fidelity of the rule set to the network for the promoter recognition and splice junction determination problems.

90.6%, therefore similar to that obtained for the test set. Unfortunately, it is not easy to compare the results here obtained with the ones in [12], [35], and [38]; differences in training and testing methodology are sufficient to preclude comparisons. For example, in [35] Setiono trains a network with three output neurons for the splice junction determination problem, while in [38] Towell uses cross-validation to test the network and the accuracy of the set of rules. Nevertheless, the results reported are similar (see Figure 26). The fidelity achieved by these extraction algorithms in the Splice Junction problem is shown in Figure 27. In [35], 100% of fidelity (which we report here) seems to be assumed from the observation that the accuracies of network and rule set are identical. However, that may not be the case when less than 100% of accuracy is achieved. The figures reported for the MofN and Subset methods refer to the network's training set. In [38], it is reported though that the figures w.r.t the network's test set are similar. Comparison with these extraction methods indicates that a drawback of our algorithm lies in the much larger size of the rule set, at least before simplification, while an advantage is the fact that the extraction is provably sound. We will come back to this in the discussion at the end of this section.

6.3 Power Systems Fault Diagnosis

Finally, we apply the extraction algorithm to power systems fault diagnosis. Figure 28 shows a simplified version of a real power plant alarms' set. This is an example of a safety-critical domain, so that the explanation provided by the set of rules is very important. In this application, we can also illustrate the extraction of rules with classical negation ($\neg$), together with default negation ($\sim$), because some neurons are labelled $\neg x$ in the net-

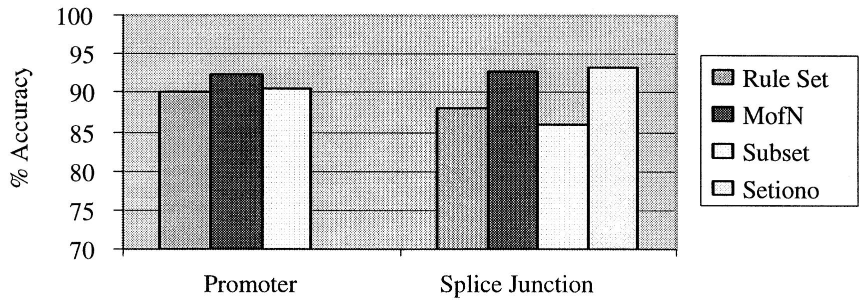


Figure 26: Comparison with the accuracies obtained by other extraction methods in the Promoter recognition and Splice Junction determination problems.

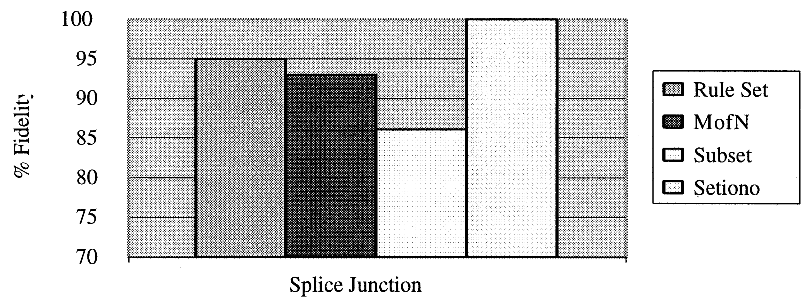


Figure 27: Comparison with the fidelity achieved by other extraction methods in the Splice Junction determination problem.

work's input and output layers (see [13] for the motivation behind adding classical negation to logic programs and the answer sets semantics[21]; see [3] about encoding background knowledge with classical negation into neural networks).

The system has two generators, two transformers with their respective circuit breakers, two buses (main and auxiliary) and two transmission lines also with their respective circuit breakers. Each transmission line has six associated alarms: breaker status (indicates whether it is open or not), phase over-current (shows that there was an over-current in the phase line), ground over-current (shows that there was an over-current in the ground

[21] In this case, the network's answer set contains three possible values: *true*, *false* and *unknown*. In our application, either there is definitely a fault (x), or definitely there is not a fault ($\neg x$), or yet there is no evidence of a fault ($\sim x$).

line), timer (shows that there was a distant fault from the power plant generator), instantaneous (shows that there was a close-up fault from the power plant generator), and auxiliary (indicates that the transmission line is connected to the auxiliary bus). In addition, each transformer has three associated alarms: breaker status (indicates whether it is open or not), overloading (shows that there was a transformer overload) and auxiliary (indicates that the transformer is connected to the auxiliary bus). Finally, there are five alarms associated with the by-pass circuit breaker: breaker status, phase over-current, ground over-current, timer and instantaneous.

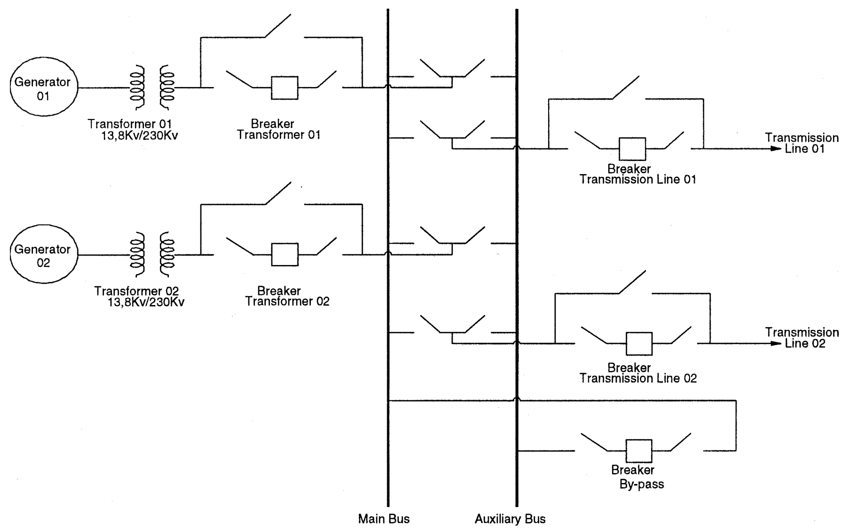


Figure 28: Configuration of a simplified power system generation plant.

Certain combinations of the set of alarms indicate either close-up or distant faults at Transmission Line 01 (11 faults), Transmission Line 02 (11 faults), or both (1 fault). In addition, each transformer may present three different faults. Finally, some alarms indicate the inexistence of a fault in the main bus or in each of the transformers.

We train a network with 23 input neurons (alarms) and 32 output neurons (faults), with 35 neurons in its single hidden layer, using standard backpropagation. Each training example associates a set of alarms with possible faults. For example, if the instantaneous alarm of Transmission Line 01 is activated then there is a Transmission Line 01 close-up fault. The set of 278 training examples contains noisy (absence of one of the characteristic alarms[22]) single and multiple faults. We use two test sets: one with 92 examples of single faults, and another with 70 examples of multiple faults.

[22] In the event of a system's fault, an alarm may fail to activate due to some equipment failure.

Figures 29, 30 and 31 display the accuracy of the network, the accuracy of the rule set and the fidelity of the rule set to the network w.r.t the test set with single faults, for each output neuron. For example, taking output neuron Fault 1 at Figure 29, the network's accuracy was 95.7% (4 misclassifications in 92 examples), the accuracy of the set of rules extracted was also 95.7%, and the fidelity of the set of rules to the network was 100%, i.e., the network and the set of rules misclassify the same 4 examples. Figures 32, 33 and 34 show the same parameters for the test set with multiple faults. A typical rule extracted from the network for this problem is of the form: *Alarm (Auxiliary_ Bus, Transmission_ Line_ 01), ∼Alarm (Main_ Bus, Transmission_ Line_ 01)→ ¬Fault (Main_ Bus, Transmission_ Line_ 01).*

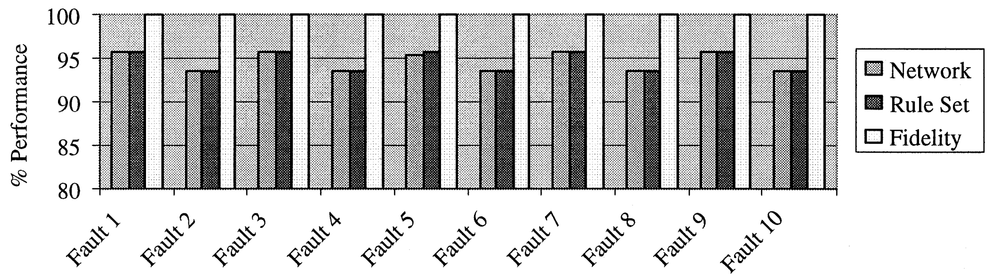


Figure 29: Network, Rule Set and Fidelity percent w.r.t the single faults test set (outputs 1-10).

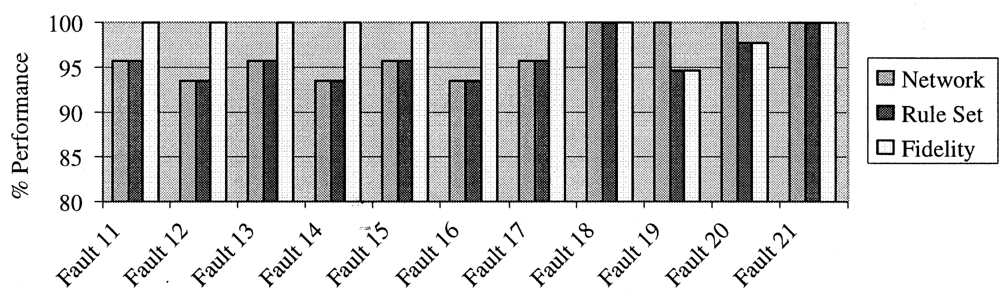


Figure 30: Network, Rule Set and Fidelity percent w.r.t the single faults test set (outputs 11-21).

The results above show the percentage of successful diagnosis achieved for each failure independently. Apart from faults 24 and 30 in the multiple

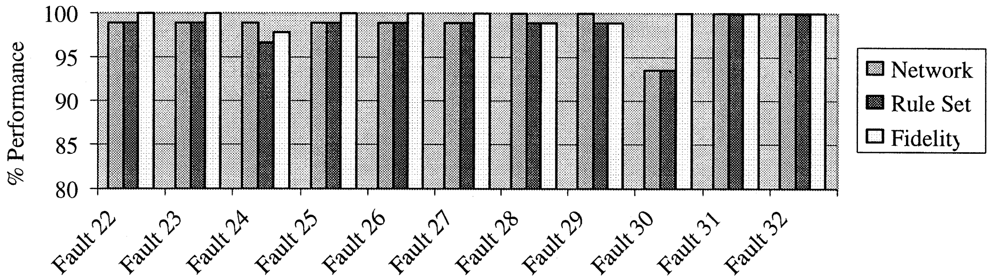


Figure 31: Network, Rule Set and Fidelity percent w.r.t the single faults test set (outputs 22-32).

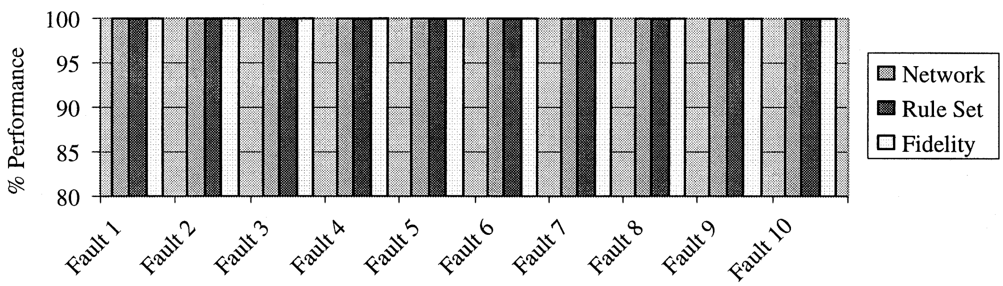


Figure 32: Network, Rule Set and Fidelity percent w.r.t the multiple faults test set (outputs 1-10).

faults case, the accuracy of the rule set is very good. Similarly, the fidelity of the rule set to the network is excellent in most cases, and in general better than the accuracy of the rule set. This suggests that the extraction algorithm prioritizes fidelity over accuracy, i.e., it tries to mimic the network's behavior, which results from the fact that the extraction is made by querying the actual network.

However, fault diagnosis systems performance is typically evaluated not only by determining the percentage of successful diagnosis but also the average size of the ambiguity set (when the system isolates failures from several possible fault modes, but fails to correctly identify the set of faults)[23]. For the network, the average size of the ambiguity set was 0.5% and 0% of the size of the set of activated faults, respectively, for the single and multiple faults test sets. For the rule set extracted, the size of the ambiguity set was

[23]For each example, *size of ambiguity set = (number of wrongly activated outputs / number of activated outputs)* × *100.*

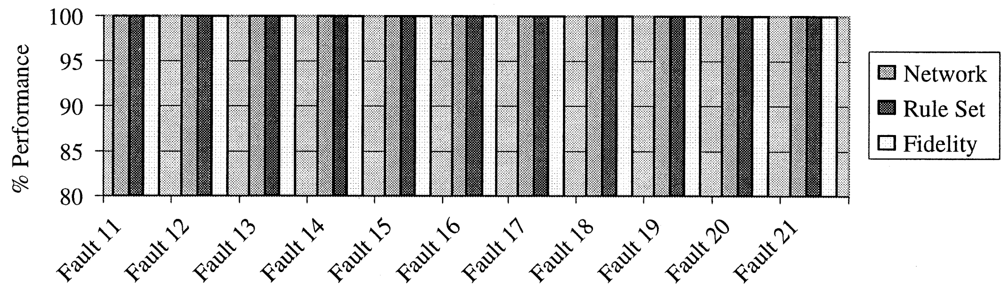


Figure 33: Network, Rule Set and Fidelity percent w.r.t the single faults test set (outputs 11-21).

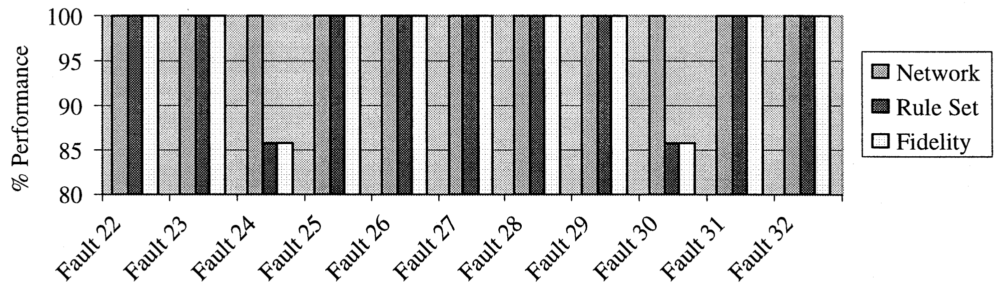


Figure 34: Network, Rule Set and Fidelity percent w.r.t the single faults test set (outputs 22-32).

2.2% and the same 0% of the size of the set of activated faults, again for the single and multiple faults test sets.

6.4 Discussion

The above experimental results corroborate two important properties of the extraction algorithm: it captures nonmonotonicity and it is sound. Soundness is also reflected in the high fidelity achieved in the applications, because it guarantees that any rule extracted is actually encoded in the network, even if such a rule does not comply with the network's test set. In other words, our extraction algorithm is bound to produce a rule set that tries to mimic the network, regardless of the network's performance in the training and test sets.

The above experiments also indicate that the drawback of the extraction algorithm lies in the size of the rule set. In comparison with [35] and [38], in the DNA sequence analysis domain, the number of rules extracted before any simplification is done is considerably bigger than, for example, the number of rules extracted by the MofN algorithm (despite the differences in sintax). Nevertheless, there are many possible improvements to be made in the simplification process. Firstly, MofN simplification metarules (not implemented) can be very powerful here, as in [38], in helping reduce the size of the rule set. Even better, simplifications could be made on the fly, at the same time that rules are generated[24]. Since every simplification rule relates to the ordering on the set of input vectors in a rather nice way, such an approach seems promising. However, we believe that a larger rule set, if not intractable, is a good price to pay for soundness.

A possible extension to the extraction algorithm concerns the extraction of *meta-level priorities* [26, 28] directly from the network's *Hidden to Output BNSs*. Negative weights from hidden to output neurons implement a preference relation. We could use this information to extract directly from the network, together with object level rules, a set of meta-level priorities between rules. Alternatively, this could be done after the extraction, when the rules are assembled to derive the final rule set. The result would be the enhancement of the rule set readability and compactness.

Improvements could also be made in the search process, exploring the ordering on the set of input vectors, and adding some new heuristics to the extraction algorithm. An example is what we did in the DNA sequence analysis case, when we jump to new minimal elements in the ordering, thus enhancing efficiency.

7 Conclusion

We have seen that most decompositional methods of extraction are unsound. On the other hand, sound and complete pedagogical extraction methods have exponential complexity. We call this problem the *complexity* $\times$ *quality* trade-off. In order to ameliorate it, we started by analyzing the cases where regularities can be found in the set of weights of a neural networks. If such regularities are present, a number of *pruning rules* can be used to safely reduce the network's input vectors search space during extraction. These pruning rules reduce the extraction algorithm's complexity in some interesting cases. Notwithstanding, we have shown that the extraction method is sound and complete w.r.t an exhaustive pedagogical extraction. A number of *simplification rules,* that fit very well into the extraction method due to a counterpart graphical representation on the network's input vectors

[24]The idea here is to implement a buffer of rules extracted and, whenever a new rule is generated, try to simplify it together with the rules in the buffer. Potentially good rules for simplification, the ones with many don't cares, would remain in the buffer for longer periods.

ordering, also help reducing the length of the extracted set of rules.

We then extended the extraction algorithm to the cases where regularities are not present in the network as a whole. That is the general case, since we do not fix any constraints on the network's learning algorithm. We identify subnetworks that always contain regularities, by showing that the network's building block, here called Basic Neural Structure (*BNS*), is regular. As a result, using the same underlying ideas, we are able to derive rules from each *BNS*. Now, however, we are applying a decompositional approach, and our problem is how to assemble the final rule set of the network. We need to provide a special treatment for *Hidden to Output BNSs*, since hidden neurons' activations are not discrete values, but real numbers in the interval (-1,1). In order to deal with that, we assume, without loss of generality, two possible intervals of activations $(-1, A_{max})$ and $(A_{min}, 1)$, and perform a worst case analysis. Finally, we use the completeness of the extraction from *Input to Hidden BNSs* to assemble the network's rule set, and show that the general case extraction method is still sound.

In this paper, we have investigated the problem of extracting the symbolic knowledge encoded in trained neural networks. Although neural networks have shown very good performance in many application domains, one of their main drawbacks lies on the incapacity to explain the reasoning mechanisms that justify a given answer. As a result, their use has become limited. This motivated the first attempts towards finding the justification for neural networks' reasoning, dating back to the end of the 1980's. Nowadays, it seems to be a consensus that the way to try and solve this problem is to extract the symbolic knowledge from the trained network. The problem of knowledge extraction turned out to be one of the most interesting open problems in the field. So far, some extraction algorithms were proposed [2, 7, 12, 30, 35, 38] and had their effectiveness empirically confirmed using certain applications as benchmark. Some theoretical results have also been obtained [4, 12, 17, 37]. However, we are not aware of any extraction method that fulfills the following list of desirable properties suggested by Thrun in [37]: 1) no architectural requirements; 2) no training requirements; 3) correctness; and 4) high expressive power. The extraction algorithm presented here satisfies the above requirements 2 and 3. It does impose, however, some restriction on the network's architecture. For instance, it assumes that the network contains a single hidden layer. This, according to the results of Hornik et al.[18], is not a drawback though. Concerning the rule set expressive power, our extraction algorithm enriches the language commonly used by adding default negation. This is done because neural networks encode nonmonotonicity. In spite of that, we believe that item 4 is the subject, among the above, that needs most attention and further development.

References

[1] R. Andrews, J. Diederich and A. B. Tickle, "*A Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks*", Knowledge-based Systems, Vol. 8, n^o 6, 1995.

[2] R. Andrews and S. Geva, "*Inserting and Extracting Knowledge from Constrained Error Backpropagation Networks*", 6th Australian Conference on Neural Networks, 1995.

[3] A. S. d'Avila Garcez, G. Zaverucha, V. N. L. da Silva, "*Applying the Connectionist Inductive Learning and Logic Programming System to Power System Diagnosis*", IEEE International Joint Conference on Neural Networks, ICNN97, Houston, USA, 1997.

[4] 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, 11(1):59-77, 1999.

[5] N. K. Bose and P. Liang, "*Neural Networks Fundamentals with Graphs, Algorithms, and Applications*", McGraw-Hill, 1996.

[6] Y. Chauvin and D. Rumelhart (eds.), "*Backpropagation: Theory, Architectures and Applications*", Lawrence Erlbaum, 1995.

[7] M. W. Craven and J. W. Shavlik, "*Using Sampling and Queries to Extract Rules from Trained Neural Networks*", Eleventh International Conference on Machine Learning, 1994.

[8] B. DasGupta and G. Schinitger, "*Analog Versus Discrete Neural Networks*", Neural Computation 8, pp.805-818, 1996.

[9] B. A. Davey and H. A. Priestley, "*Introduction to Lattices and Order*", Cambridge University Press, 1990.

[10] M. Fitting, "*Metric Methods - Three Examples and a Theorem*", Journal of Logic Programming 21, pp.113-127, 1994.

[11] L. M. Fu; "*Integration of Neural Heuristics into Knowledge-based Inference*"; Connection Science, Vol. 1, pp. 325-340; 1989.

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

[13] M. Gelfond and V. Lifschitz, "*Classical Negation in Logic Programs and Disjunctive Databases*", New Generation Computing, Vol. 9, Springer-Verlag, 1991.

[14] J.L. Gersting, "*Mathematical Structures for Computer Science*", Computer Science Press, 3^{rd} edition, 1993.

[15] J. Hertz, A. Krogh and R. G. Palmer, "*Introduction to the Theory of Neural Computation*", Santa Fe Institute, Studies in the Science of Complexity, Addison-Wesley, 1991.

[16] M. Hilario, "*An Overview of Strategies for Neurosymbolic Integration*", Connectionist-Symbolic Integration: from Unified to Hybrid Approaches - IJCAI 95, 1995.

[17] S. Holldobler and Y. Kalinke, "*Toward a New Massively Parallel Computational Model for Logic Programming*", Workshop on Combining Symbolic and Connectionist Processing, ECAI 94, 1994.

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

[19] H. Kautz, M. Kearns and B. Selman, "*Horn Approximations of Empirical Data*", Artificial Intelligence, 74.129-145, 1995.

[20] N. Lavrac and S. Dzeroski, "*Inductive Logic Programming: Techniques and Applications*", Ellis Horwood Series in Artificial Intelligence, 1994.

[21] N. Lavrac, S. Dzeroski and M. Grobelnik, "*Experiments in Learning Nonrecursive Definitions of Relations with LINUS*", Technical Report, Josef Stefan Institute, Yugoslavia, 1990.

[22] J. W. Lloyd, "*Foundations of Logic Programming*", Springer - Verlag, 1987.

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

[24] M. Minsky, "*Logical versus Analogical, Symbolic versus Connectionist, Neat versus Scruffy*", AI Magazine, Vol. 12, n^o 2, 1991.

[25] S. Muggleton and L. Raedt, "*Inductive Logic Programming: Theory and Methods*", The Journal of Logic Programming, 1994.

[26] D. Nute, "*Defeasible Logic*", In D. Gabbay, C.J. Hogger and J. A. Robinson, Handbook of Logic in Artificial Intelligence and Logic Programming, Vol.3, pp.353-396, Oxford Science Publications, 1994.

[27] D. Ourston and R. J. Mooney, "*Theory Refinement Combining Analytical and Empirical Methods*", Artificial Intelligence, Vol. 66, pp. 273-310, 1994.

[28] H. Prakken and G. Sartor, "*Argument-based Extended Logic Programming with Defeasible Priorities*", Journal of Applied Non-Classical Logic, 7.1/2, pp.25-75, 1997.

[29] F. P. Preparata and R. T. Yeh, "*Introduction to Discrete Structures*", Addison-Wesley, 1973.

[30] E. Pop, R. Hayward and J. Diederich, "*RULENEG: Extracting Rules from a Trained ANN by Stepwise Negation*", QUT NRC, 1994.

[31] O. T. Rodrigues, "*A Methodology for Iterated Information Change*" PhD Thesis, Dept. of Computing, Imperial College, 1997.

[32] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "*Learning Internal Representations by Error Propagation*", Parallel Distributed Processing, Vol. 1, D. E. Rumelhart, J. L. McClelland and the PDP Research Group, MIT Press, 1986.

[33] A. Schrijver, "*Theory of Linear and Integer Programming*", John Wiley and Sons, 1986.

[34] R. Setiono, "*A Penalty-function for Pruning Feedforward Neural Networks*", Neural Computation 9, pp.185-204, 1997.

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

[36] 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.

[37] S. B. Thrun, "*Extracting Provably Correct Rules from Artificial Neural Networks*", Technical Report, Institut fur Informatik, Universitat Bonn, 1994.

[38] G. G. Towell and J. W. Shavlik, "*The Extraction of Refined Rules From Knowledge Based Neural Networks*", Machine Learning, Vol. 131, 1993.

[39] G. G. Towell and J. W. Shavlik, "*Knowledge-Based Artificial Neural Networks*", Artificial Intelligence, Vol. 70, 1994.

[40] J. D. Watson, N. H. Hopkins, J. W. Roberts, J. A. Steitz and A. M. Weiner, *Molecular Biology of the Gene, Volume 1*, Benjamin Cummings, Menlo Park, 1987.