Symbolic Knowledge Extraction from Trained Neural Networks: A New Approach

A. S. d'Avila Garcez[†] K. Broda[†] D. M. Gabbay[‡]
[†]Imperial College - Department of Computing
[‡]King's College London - Department of Computer Science

1st Draft April 1998
2nd Draft August 1998

Abstract

In this report, we investigate the problem of symbolic knowledge extraction from trained neural networks, and present a new extraction method. Although neural networks have shown very good performance in terms of learnability, generalizability and speed in many application domains, one of their main drawbacks lies on the incapacity to provide an explanation to the underlying reasoning mechanisms that justify a given answer. As a result, their use in many application areas, for instance in safety-critical domains, has become limited. The so called "explanation capability" of neural networks can be achieved by the extraction of symbolic knowledge from it, using "Rules' Extraction" methods.

We start by discussing some of the main problems of knowledge extraction methods. In an attempt to ameliorate these problems, we identify, in the case of regular networks, a partial ordering on the input vectors space. A number of pruning rules and simplification rules that interact with this ordering is defined. Those rules are used in our extraction algorithm in order to help reducing the input vectors search space during a pedagogical knowledge extraction from trained networks. They are also very useful in helping to reduce the number of rules extracted, what provides clarity and readability to the rule set. 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 identify that non-regular networks contain regularities in their subnetworks. As a result, the underlying extraction method for regular networks can be applied, but now in a decompositional fashion. The problem, however, is how to combine the set of rules extracted from each subnetwork into the final rule set. We propose a solution to this problem such that we are able to keep the soundness of the extraction algorithm, although we have to drop completeness.

The material presented in this report is an integral part of our neural-symbolic integration proposal. A detailed description of the system and the results obtained with its application in computational biology can be found in [5].

1 Introduction

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 generalization capabilities. 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.

The Connectionist Inductive Learning and Logic Programming (CIL^2P) system [5] is a proposal towards tightly coupled neural-symbolic integration. CIL^2P is a massively parallel computational model based on a feedforward artificial neural network that integrates inductive learning from examples and background knowledge with deductive learning from Logic Programming. 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. The extraction explains the learning process and gives justifications for the network's answers. Moreover, the symbolic knowledge extracted can be more easily analyzed by a domains knowledge expert, that decides whether or not to feed it back to the system (5), closing the learning cycle.

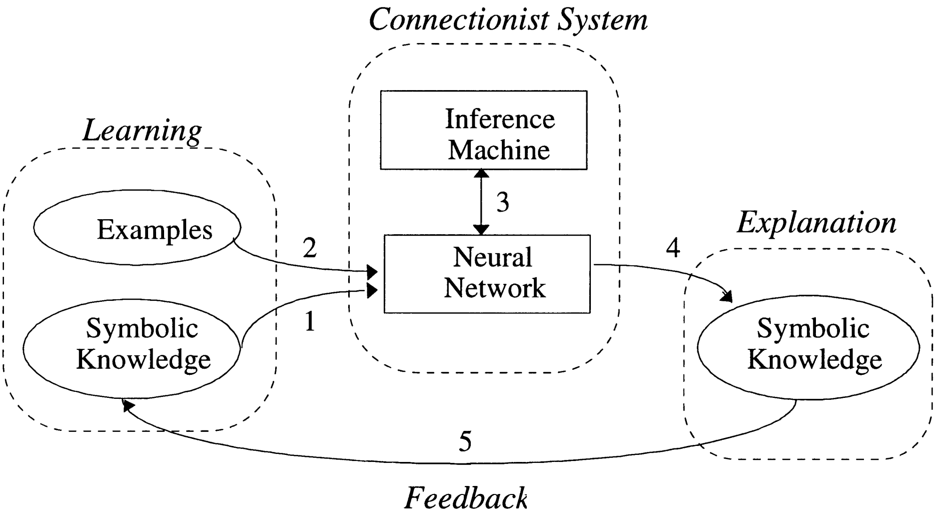


Figure 1: Neural-Symbolic Integration

In this report, 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. Briefly, the problem lies on the complexity of the network's extraction algorithm. Previously, we have shown that each acceptable or locally stratified logic program is equivalent to a single hidden layer feedforward neural network [5]. In one direction of that equivalence relation, we have obtained a translation algorithm (figure 1(1)) that, given a logic program, derives a neat neural network structure. We call that network the "canonical form" of the logic program, since numerous other network structures can be equivalent to it. The problem arises in the converse direction, that is, given a trained neural network how can we find the logic program that is equivalent to it? Our problem is that it is very unlikely that a neat, canonical network will result from a learning process. Moreover, a typical real-world application network may contain hundreds of input neurons and thousands of connections, each one with a real number weight associated.

The so called "explanation capability" of neural networks can be obtained by the extraction of symbolic knowledge from it. The extraction is responsible for making the knowledge learned accessible for an expert's analysis and for allowing the justification of the decision making process[1]. Moreover, artificial neural networks have shown to outperform symbolic machine learning methods in many application areas, and the development of techniques for extracting rules from trained neural networks may contribute to the solution of the so called "knowledge acquisition bottleneck" problem. The domain theory extracted, resulting from inductive learning with examples, can be added to a knowledge base or used in the solution of analogous domains problems. In a more general perspective, we also would like to understand the neural learning mechanisms. "We do not just want to build systems which show intelligent behavior, we also want to understand how humans do it" [18].

It is known that 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).

A classification scheme for rule extraction algorithms should be 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 [1]. The first classification item refers directly to the symbolic knowledge presented to the end user from the rule 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 first one, the rule extraction process occurs at the level of individual, hidden and output, units within the trained neural network, which is viewed as a "white box". In the second one, the neural network is viewed as a "black box", since the rule extraction process 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 rules accuracy, consistency and comprehensibility; while the last item refers to the requirement for the algorithm to be as effective as possible. In this sense, a crucial issue in developing a rule extraction algorithm is how to constrain the size of the solution space to be searched.

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

Thrun [39] defines the following desirable properties of an extraction algorithm. 1) No architectural requirements: a general extraction mechanism should be able to operate with all types of neural networks. 2) 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. 3) Correctness: the extracted rules should describe the underlying network as correctly as possible. 4) 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 combinations of inputs *imply* a particular output. It is a causality relation. Moreover, we argue that neural networks are nonmonotonic systems (see [5]) and, therefore, the set of rules extracted may contain default negation ($\sim$). Each neuron can represent a concept or its "classical" negation. 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, L_j $(1 \leq j \leq m)$ represents a neuron in the network's input layer, and L_{m+1} represents a neuron in the network's output layer[2].

The figure below gives a general idea about the pieces of knowledge represented in a neural network and about their relations. The training set (see figure 2, (2)) corroborates part of the background knowledge (1) and revises another part, while the generalization set (3) embodies the training set. We say that an extraction algorithm is sound and complete if the rule set extracted is equivalent to the network's generalization set. If, however, the rule set (see (4), below) is a subset of (3), then the extraction is sound but not complete. In this case, if the rule set is at least a superset of the training set (2), then the extraction has lost only part of the network's generalization. Rule set (5) is an example of unsound and incomplete extraction. Here, in particular unsoundness is a major problem because the complement of (5) w.r.t (3), that is (5)-(3), may be responsible for wrong generalization.

In this report we describe a new approach for symbolic knowledge extraction from trained neural networks. We start by discussing some of the main problems found in the literature. In an attempt to ameliorate them, we identify, in the case of regular networks, a partial ordering on the input vectors space and a number of pruning rules and simplification rules that interact with that ordering. Those rules are used in our extraction algorithm in order to help reducing the input vectors search space during a pedagogical symbolic

[2] Note that this is the language of extended logic programming [14].

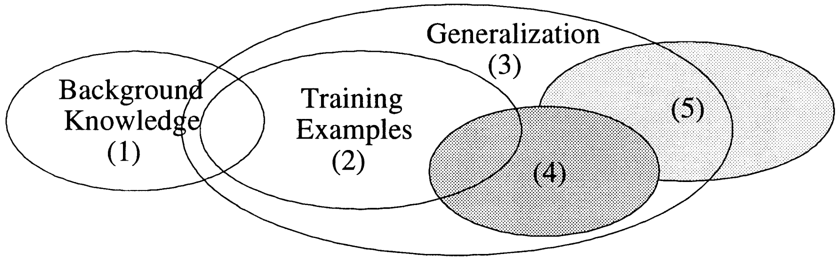


Figure 2: Knowledge encoded in a neural network × extraction.

knowledge extraction from a trained network. They are also very useful in helping to reduce the number of rules extracted, what provides clarity and readability to the rule set. We show that, in the case of regular networks, the extraction algorithm is sound and complete. We also provide the extension of the extraction algorithm to the general case, that is, to the case of non regular networks. We identify that even non regular networks contain regularities inside its subnetworks. As a result, the underlying extraction method for regular networks can be applied in the general case, but now in a decompositional fashion. The main problem we have to tackle, however, is how to combine the set of rules obtained from each subnetwork into the final rule set of the original network. We propose a solution to this problem such that we are able to keep soundness of the extraction algorithm, although we have to drop completeness.

In section 2 we discuss the main problems encountered in the task of extracting symbolic knowledge from trained feedforward neural 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 task, culminating with the outline of the extraction algorithm for the class of regular networks. In section 5 we extend the extraction algorithm for the class of non regular networks, the general case. Finally, in section 6 we conclude and discuss directions for future work.

2 Related Work

Among the rules' extraction methods, the one presented in [18], the "Ruleneg" [31], the "VIAnalysis" algorithm [39] and the "Rule-Extraction-as-

Learning" method [8] use the so called "pedagogical" approach, while the "Subset" [13], the "m of n" method [40], the "Rulex" [2] and Setiono's proposals [35] and [36] are "decompositional" methods.

In the CIL^2P system, after learning, a network N encodes a knowledge P' that contains the background knowledge P complemented or even revised by the knowledge learned with training examples. Hence, an accurate extraction procedure must derive P' from N. At the moment, only pedagogical approaches can guarantee that the knowledge extracted is equivalent to the network. In other words, only pedagogical approaches are sound and complete. Those methods, for instance [18], take into account all possible combinations of the input vector $\mathbf{i}$ of N 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, like [8], reduce the input vectors space by extracting rules for the learning set only, excluding the network's generalization.

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 apply, in general, heuristically guided searches to the process of rules' extraction. The "Subset" method [13], 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 may be written as a rule. One of the most interesting decompositional methods is the "m of n" technique [40]. 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 for each group of rules obtained: *If m of (A_1,..., A_n) are "true" then A is "true"*, where $m \leq n$. The recent work by Rudy Setiono [35][36] is another proposal of decompositional rules' extraction. Setiono proposes a penalty function for pruning single hidden layer feedforward neural network, and then generates rules from the pruned network by considering only a small number of activation values at the hidden units.

We argue that neural networks are nonmonotonic systems. That is the main difference between our extraction approach and the ones above. The network's nonmonotonicity should be reflected in the rule set derived. We do that by adding negation by default ($\sim$) to the language. Because of its nonmonotonic behavior, we can not expect to map a neural network properly into a set of rules composed of Horn clauses only. The following example illustrates that.

Example 1 *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 the input vector* $\mathbf{i}_1 = (1,0)$ *activates* x. *We would derive the rule* $a \rightarrow x$. *As a result, we 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 defined, it may be the case that the input vector* $\mathbf{i}_2 = (1,1)$ *does not activate* x. *In this case, we 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. As a result, in order to conclude that* $a \rightarrow x$ *we need to assure that both* $ab \rightarrow x$ *and* $a \sim b \rightarrow x$ *first.*

In our approach [5], differently from [13] and [40], the hidden units of N do not represent any specific concept of P. They actually represent rules. Hidden neurons should be allowed to change their meaning during learning, since they are responsible for the network's generalization capability. After all, we do not have any external control over then during the learning process. The following observation corroborates this idea. "The requirement for hidden units to be approximated as threshold units and the requirement that the extracted rules use an intermediate concept to represent each hidden unit may not enable a sufficiently accurate description of the network to be extracted. In the case where the meaning of a hidden unit does change during training, the comprehensibility of the extracted rules may be significantly degraded" [1].

Decompositional methods, such as [40] and [36], in general use weights pruning mechanisms prior to extraction (notice the difference between pruning the input vectors search space and pruning the networks weights). There is, however, no guarantee that a pruned network will be equivalent to the original one. Consequently, weights pruning may as well prune information. That is the reason why those methods usually require retraining the pruned network, despite of training being already an expensive task. 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, however, there is no guarantee that the training will be successful under these restrictions. Moreover, methods that use a penalty function are bounded to restrict the network's learning capability[3]. Even if we avoid penalty functions and weights' clustering and pruning, the simple task of decomposing the network into smaller subnetworks, from which rules are extracted and then put together, 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. The following example illustrates this fact.

Example 2 *Consider the network of Figure 3. Let us assume that the weights are such that* $\mathbf{i} = (1,1)$ *activates neither* n_1 *nor* n_2*, but that the composition of* n_1 *and* n_2 *activates* x*. For instance, 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* $\mathbf{i} = (1,1)$ *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, that is, considered either totally activated or non activated, then* $n_1 = 1$ *and* $n_2 = 1$ *activate* x*. For instance, the weights 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 *(lets say* $x < 0.5$*), but* $n_1 = 1$ *and* $n_2 = 1$ *do imply* x *(say,*

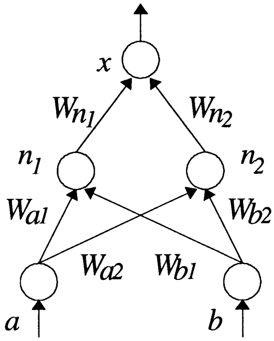


Figure 3: A simple example of unsoundness and incompleteness of some decompositional extraction algorithms.

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

$x > 0.5$). *As a result, a decompositional method, like [40], would conclude that* $ab \rightarrow x$ *when actually* $ab \not\rightarrow x$.

The first case is an example of incompleteness. The second one shows how decompositional extraction methods may turn out to be unsound. Even Fu's extraction proposal [13], that 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 rules' extraction methods and the *quality* of the knowledge extracted from the network. An alternative to reduce this trade-off is to use an "eclectic" approach. In our view, it is necessary to adopt an input vectors space pruning method, followed whenever possible by a pedagogical rules' extraction procedure. Our goal is to reduce complexity by applying the extraction algorithm in a smaller solution space, and to enhance the rule set readability by applying simplifications in order to generate a reduced, yet accurate, final set of rules.

3 Preliminaries

3.1 General

We need to assert some basic assumptions that will be used throughout this report. $\aleph$ 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 said to be* 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* $y \in X$ *s.t.* $y < x$.

x *is* minimum *if for all element* $y \in X, x \leq y$. *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 $y \in X$ s.t. $x < y$.*

x is maximum *if for all element $y \in X, y \leq x$. 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 $x, y \in X$ we define the* least upper bound *of x and y as 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 *A* metric space *is a tuple $\langle U, f \rangle$, where U is a set and $f : U \times U \rightarrow \Re$ is a function satisfying the following conditions:*

(I) $f(x, y) \geq 0$,
(II) $f(x, y) = 0 \; iff \; x = y$,
(III) $f(x, y) = f(y, x)$,
(IV) $f(x, y) \leq f(x, z) + f(z, y)$.

We say that f is a *metric* on U. A metric f on U is *bounded* iff for some constraint k, $f(x, y) \leq k$, for all $x, y \in U$. Functions $g : U \times U \rightarrow \Re$ for which conditions $(I) - (IV)$ are yet to be checked are called *distance functions.*

3.2 Neural Networks

Hornik, Stinchcombe and White [19] have proved that standard feedforward neural networks with as few as 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. In their words "Single hidden layer $\sum \prod$ feedforward networks can approximate any measurable function arbitrarily well, regardless of the continuous nonconstant function h used, regardless of the dimension of the input space **I**, and regardless of the input space environment μ. In this precise and satisfying sense, $\sum\prod$ networks are universal approximators". As a result, we concentrate on single hidden layer networks, without loss of generality.

Given a single hidden layer trained 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 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. Note that, more generically, $\mathbf{o} = \delta(\mathbf{i})$ where $\delta : \Re^p \rightarrow \Re^q$. For each output $o_j(1 \leq j \leq q)$ in $\mathbf{o}$ we have $o_j = h(\sum_{i=1}^{r}(W^2_{ji}.h(\sum_{k=1}^{p}(W^1_{ik}.i_k) - \theta_{n_i})) - \theta_{o_j})$.

Whenever it is not necessary to differentiate between hidden and output layer connections, 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.

Note that the above system of equations can be described in a more concise way by using matrices operations. In order to do so, each bias $-\theta_i$

can be seen as a weight coming from an extra neuron with input always fixed at 1. We define $i_{p+1} = 1$ and obtain $\mathbf{i}'$ by appending i_{p+1} to $\mathbf{i}$ ($\mathbf{i}' = \mathbf{i} : i_{p+1}$). Similarly, we define $n_{r+1} = 1$ and $\mathbf{n}' = \mathbf{n} : n_{r+1}$. We set the weights matrices $\mathbf{W}^1$ and $\mathbf{W}^2$ with the weights and biases of connections from input to hidden and from hidden to output layers, respectively, as follows.

$$\mathbf{W}^1 = \begin{pmatrix} W^1_{11} & W^1_{12} & \cdots & W^1_{1p} & -\theta_{n_1} \\ W^1_{21} & W^1_{22} & \cdots & W^1_{2p} & -\theta_{n_2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ W^1_{r1} & W^1_{r2} & \cdots & W^1_{rp} & -\theta_{n_r} \end{pmatrix}$$

$$\mathbf{W}^2 = \begin{pmatrix} W^2_{11} & W^2_{12} & \cdots & W^2_{1r} & -\theta_{o_1} \\ W^2_{21} & W^2_{22} & \cdots & W^2_{2r} & -\theta_{o_2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ W^2_{q1} & W^2_{q2} & \cdots & W^2_{qr} & -\theta_{o_q} \end{pmatrix}$$

Given the above matrices, we can rewrite the systems of equations 1 and 2, respectively, as equations 3 and 4 below.

$$\mathbf{n}^{\mathrm{T}} = h(\mathbf{W}^1.\mathbf{i}'^{\mathrm{T}}) \tag{3}$$

$$\mathbf{o}^{\mathrm{T}} = h(\mathbf{W}^2.\mathbf{n}'^{\mathrm{T}}) \tag{4}$$

And putting equations 3 and 4 together we obtain:

$$\mathbf{o}^{\mathrm{T}} = h(\mathbf{W}^2.((h(\mathbf{W}^1.\mathbf{i}'^{\mathrm{T}}))^{\mathrm{T}} : 1)^{\mathrm{T}}) \tag{5}$$

We define the extraction problem as follows. *Given a particular set of weights and biases for the weights matrices* $\mathbf{W}^1$ *and* $\mathbf{W}^2$*, resulting from a training process on the 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 $i_j = -1$, i.e., $i_j \in \{-1, 1\}$. 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

So far we have seen many problems related to symbolic knowledge extraction from trained neural networks. Basically, they result from what we call the *quality* × *complexity* trade-off. 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 $o_j = h(\sum_{i=1}^{r}(W_{ji}^2 n_i) - \theta_{o_j})$ *s.t.* $o_j > A_{min}$. We can equivalently define, therefore, the extraction problem as follows. Let $\mathbf{I}$ be the set of input vectors and $\mathbf{O}$ be the set of output vectors. We have seen that $\mathbf{o} = \delta(\mathbf{i})$. Since δ is not a bijective function, δ^{-1} does not exist. We define, therefore, 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 $o_j > A_{min}$. Formally, find $\mathbf{I}' = \{\mathbf{i} \mid \mathbf{o}\xi\mathbf{i}$ and each o_j in $\mathbf{o}$ is 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, note that in order to satisfy the above constraint $o_j > A_{min}$, it is required that $x = \sum_{i=1}^{r}(W_{ji}^2 n_i) - \theta_{o_j} > 0$. The above constraint over o_j is therefore written as equation 6 below, in terms of the hidden neurons' activation values. Hence, each network's output o_j is determined by the system of equations 1 plus the constraint 6.

$$j \textit{ is true iff } W_{j1}^2 n_1 + W_{j2}^2 n_2 + \cdots + W_{jr}^2 n_r > h^{-1}(A_{min}) + \theta_{o_j} \qquad (6)$$

Remark 1 *Given* $h(x) = \frac{2}{1+e^{-\beta x}} - 1$, *we obtain* $h^{-1}(x) = -\frac{1}{\beta}\ln\left(\frac{1-x}{1+x}\right)$. *Diagram 4 below plots both* $h(x)$ *and* $h^{-1}(x)$ *with* $\beta = 1$. *Note that the parameter* β *is responsible for defining the slope of the activation function. The bigger* β *is, the more the activation function approximates the step function, as diagram 5 shows.*

4.1 Positive Networks

We start by considering a very simple network where all weights are positive real numbers, i.e., $\forall ij, W_{ij} \in \Re^+$. In other words, each W_{ij} in the system of equations 1 and in equation 6 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 $\forall j_{(1 \leq j \leq q)} o_j(n_i(\mathbf{i}_m)) > o_j(n_i(\mathbf{i}_n))$.

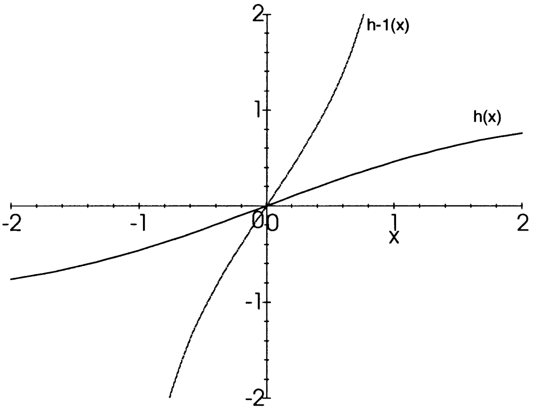


Figure 4: Ploting $h(x)$ and $h^{-1}(x)$ for $\beta = 1$.

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 clarifying the above ideas.

Example 10 *Consider the network and its constraint representation of figure 6. We know that* $n_1 = h(W_a.a + W_b.b - \theta_{n_1})$. *Since* $W_a, W_b > 0$, *it is easy to verify that the ordering of figure 7 on the set of input vectors* $\mathbf{I}$ *holds w.r.t the output* x. *The ordering says, for instance, that the activation of* 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 above 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. Note that if it is the case that both*

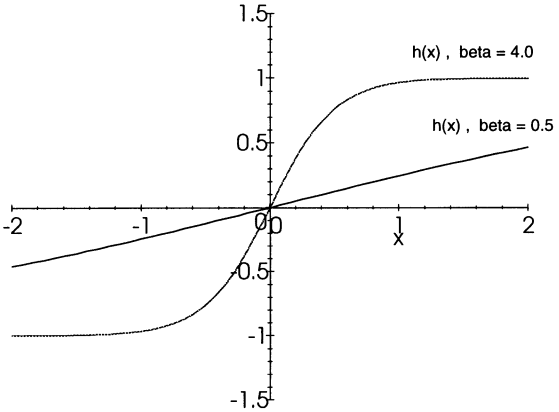


Figure 5: Ploting $h(x)$ for $\beta = 4$ and $\beta = 0.5$.

$(1, 1)$ *(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 input vectors set **I** w.r.t the output vectors set **O**. As a result, that information can be very useful to guide a pedagogical extraction procedure of symbolic knowledge from the network. The ordering can help pruning the input vectors search space, so that it may be not necessary anymore to check all the 2^n possible input vectors during a pedagogical extraction. Given an ordering on **I**, we can avoid checking some irrelevant input vectors safely, in the sense that those vectors that are not checked would not generate new rules. Moreover, each rule obtained is correct since the extraction is pedagogical, done by querying the actual network.

Note that in the worst case we still have to check all the 2^n possible input vectors, and that 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 input vectors set, but that it may be impossible to find it without having to check each input vector, considering the particular set of weights in the network. Thus, we will focus

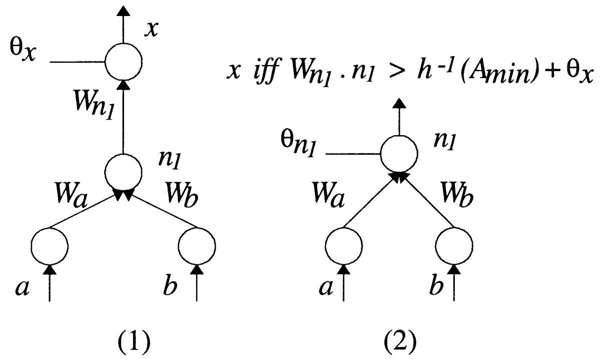


Figure 6: A single hidden neuron network (1) and its constraint representation (2) w.r.t. output x. $W_a, W_b, W_{n_1} \in \Re^+$.

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 8. If we know, for instance, that* $W_{a1} > W_{b1}$ *then we can derive the ordering of figure 9(1) on these weights 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 9(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 weights (and on the input vectors set) w.r.t the output* x *as well.*

However, if $W_{a1} > W_{b1}$ *(see figure 10(1)) and* $W_{a2} < W_{b2}$ *(10(2)) then we can only derive a partial order 10(3) w.r.t the output* x. *Note that the ordering 10(3) on the network's weights corresponds to the ordering given in example 10 on the input vectors set* **I**.

If a particular set of weights is given, for example if $W_{a1} = 10, W_{b1} = 5, W_{a2} = 2$ *and* $W_{b2} = 8$, *we can actually check that* $\{W_{b1}, W_{b2}\} > \{W_{a1}, W_{a2}\}$, *corresponding to* $(-1, 1) > (1, -1)$ *in the ordering of example 10. For the time being, we use the partial ordering of figure 10(3) because it is "easily found", regardless of the network's weights values.*

Examples 10 and 11 above indicate that the partial ordering on the input vectors set 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

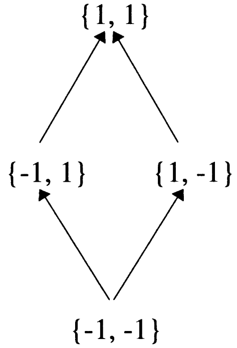


Figure 7: Ordering on the input vectors set $\mathbf{I}$ of the network (1) of figure 6.

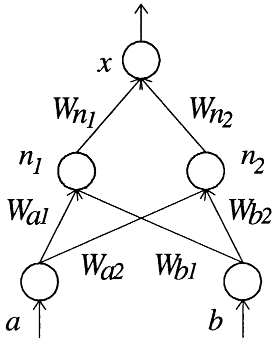


Figure 8: A two hidden neurons network. $W_{ij} \in \Re^{+}$.

$W_{ij} \in \Re^{+}$ then the partial ordering on the input vectors set is not affected by the number of hidden neurons. Note that, although the weights are different, if a given input, say $(a, b) = (1, 1)$, 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.

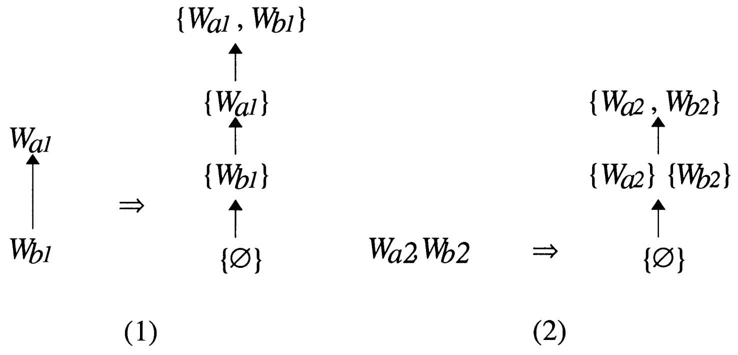


Figure 9: Linear ordering on the network's weights.

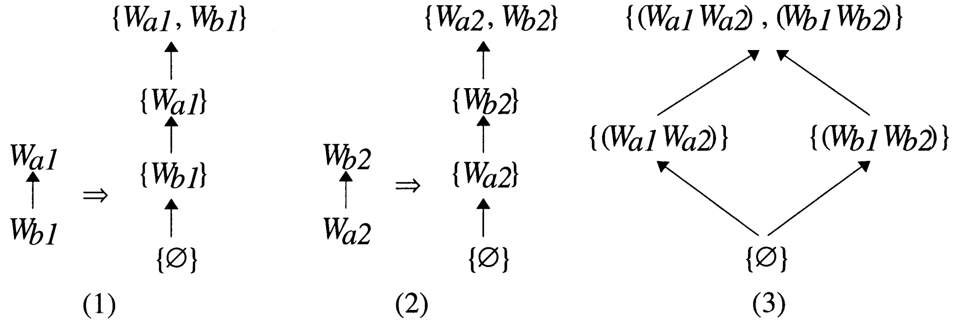


Figure 10: Partial ordering on the network's weights.

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 11 since we do not have any extra information regarding the network's weights. The ordering is built starting from the element $(-1, -1, -1)$ and then flipping each input at a time from -1 to 1 until $(1, 1, 1)$ is obtained.

It seems that for an arbitrary number of input and hidden neurons, if $W_{ij} \in \Re^+$ then there will always exist a unique minimal element $(-1, -1, ..., -1)$ and a unique maximum element $(1, 1, ..., 1)$ in the ordering on the input vectors set 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

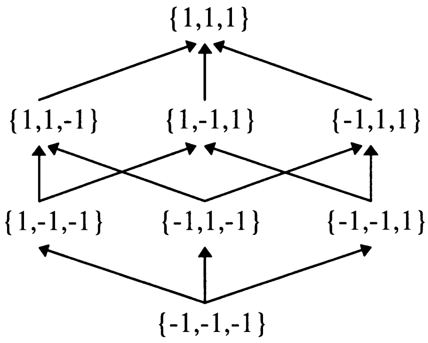


Figure 11: Partial ordering w.r.t set inclusion on the network's input vectors set $(p = 3)$.

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 $\mathbf{P}$. That is, each literal occurs at most in one input neuron and in one output neuron. More precisely, let $\mathcal{I}$ be the set of input neurons, the function $f : \mathcal{I} \to \mathbf{P}$ is injective. Similarly, $g : \mathcal{O} \to \mathbf{P}$ is injective, where $\mathcal{O}$ is the set of output neurons. As a result, 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 $\mathbf{i}(q) = ff$.

A usual way to represent subsets of a set as abstract data types is to impose an ordering on the set and then use a bit vector, a vector with 1's and 0's, to denote which members of the set are, respectively, present or absent in the subset. We identify, therefore, that an input vector $\mathbf{i}$ can also be seen as the above abstract representation of a subset of the set of input neurons, with the only difference that, instead of 1's and 0's, we use 1's to denote presence and -1's to denote absence in the subset. 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 *[32] 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 more than that.

Proposition 16 *[30] 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. *If* $\mathbf{i}_m = \mathbf{i}_n$ *the proof is trivial. If* $\mathbf{i}_m \neq \mathbf{i}_n$ *and* $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_n$ *then at least one input in* $\mathbf{i}_m$, *say* $\mathbf{i}_m(i_i)$, *is flipped from -1 to 1 in* $\mathbf{i}_n$, *that is* $\mathbf{i}_m(i_i) = -1$ *and* $\mathbf{i}_n(i_i) = 1$. *Let r be the number of hidden neurons in the network. Firstly, we have to show that* $h(\sum_{i=1}^{p}(W_{1i}^{1}\mathbf{i}_m(i_i) - \theta_{n_1})) + h(\sum_{i=1}^{p}(W_{2i}^{1}\mathbf{i}_m(i_i) - \theta_{n_2})) + \cdots + h(\sum_{i=1}^{p}(W_{ri}^{1}\mathbf{i}_m(i_i) - \theta_{n_r})) \leq h(\sum_{i=1}^{p}(W_{1i}^{1}\mathbf{i}_n(i_i) - \theta_{n_1})) + h(\sum_{i=1}^{p}(W_{2i}^{1}\mathbf{i}_n(i_i) - \theta_{n_2})) + \cdots + h(\sum_{i=1}^{p}(W_{ri}^{1}\mathbf{i}_n(i_i) - \theta_{n_r}))$. *By the definition of* $\leq_{\mathbf{I}}$ *and since* $W_{ji} \in \Re^+$ *we derive immediately that* $\forall j (1 \leq j \leq r)$ $\sum_{i=1}^{p}(W_{ji}^{1}\mathbf{i}_m(i_i) - \theta_{n_j})$ $\leq \sum_{i=1}^{p}(W_{ji}^{1}\mathbf{i}_n(i_i) - \theta_{n_j})$, *and by the monotonically crescent characteristic of* $h(x)$ *we obtain* $\forall j (1 \leq j \leq r)$ $h(\sum_{i=1}^{p}(W_{ji}^{1}\mathbf{i}_m(i_i) - \theta_{n_j})) \leq h(\sum_{i=1}^{p}(W_{ji}^{1}\mathbf{i}_n(i_i) - \theta_{n_j}))$. *This proves that* $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_n$ *implies* $(\mathbf{n}_m(n_j) = \delta(\mathbf{i}_m)) \leq (\mathbf{n}_n(n_j) = \delta(\mathbf{i}_n))$ *for all* $1 \leq j \leq r$. *In the same way, we obtain that* $h(\sum_{i=1}^{r}(W_{ji}^{2}\mathbf{n}_m(n_i) - \theta_{o_j}))$ $\leq h(\sum_{i=1}^{r}(W_{ji}^{2}\mathbf{n}_n(n_i) - \theta_{o_j}))$ *for all* $1 \leq j \leq q$ *and therefore that* $(\mathbf{o}_m(o_j) = \delta(\mathbf{i}_m)) \leq (\mathbf{o}_n(o_j) = \delta(\mathbf{i}_n))$. □

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 8), 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. We can, therefore, transform the network at figure 8 into the network of figure 12, where all weights are positive and the input neuron b is negated.*

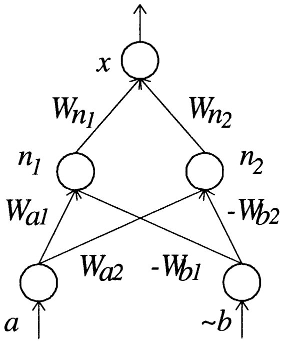


Figure 12: The positive form of a (regular) network.

Given the network of figure 12, we can find an ordering on the input vectors set 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 8 and 12 are equivalent.*

Let us analyze what we have done in the above example. If all connections from hidden to output layer of a network are either positive or 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 *Transformation Algorithm.*

Example 19 *Consider again the network given at example 11 (figure 8), but now assume that only* $W_{a2} < 0$. *Applying the transformation algorithm we obtain the network of figure 13.*

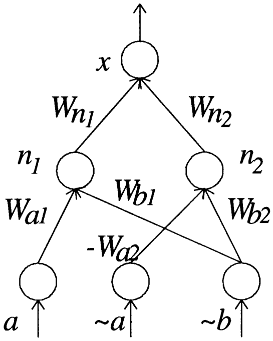


Figure 13: The positive form of a (non regular) network.

Although the network of figure 13 has positive weights only, it is clearly not equivalent to the original network (figure 8). 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 13) 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 all either positive or 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 input vectors set 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 12. Given $\mathcal{I}_+ = [a, \sim b]$ we obtain the ordering (1) of figure 14 w.r.t set inclusion. The ordering 14(2) on the input vectors set of the original network N is obtained by mapping each element of (1) into (2) using $\sim b = 1 \Rightarrow b = -1$ and $\sim b = -1 \Rightarrow b = 1$.*

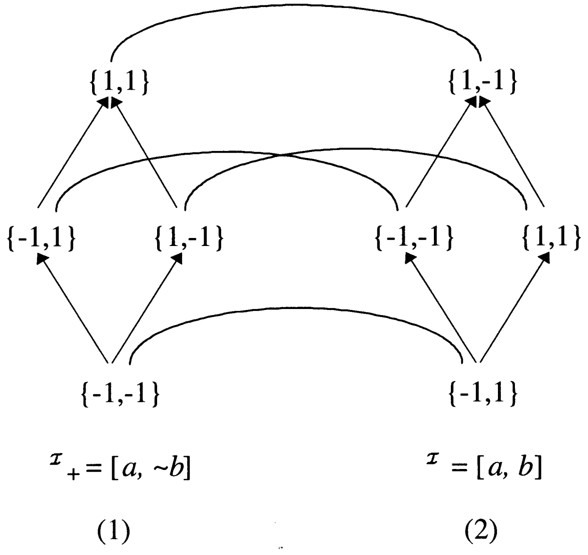


Figure 14: The ordering w.r.t set inclusion on the positive form of a network (1) and the ordering on the original network (2).

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 input vectors set. 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 $\{-1,1\}$ s. t. $\sigma([x_1,...,x_s](i_1,...,i_s)) = (i'_1,...,i'_s)$, 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 then* $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_n$ *implies* $(\mathbf{o}_m(o_j) = \delta(\sigma(\mathbf{i}_m)) \leq (\mathbf{o}_n(o_j) = \delta(\sigma(\mathbf{i}_n))$*, for all* $1 \leq j \leq q$.
Proof. *Straightforward by proposition 17 and by the above definition of the mapping function σ.*□

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 the positive form 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.* □

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 $\mathbf{Co}_j$ on the j-th output neuron of N, then $\mathbf{i}_n$ also satisfies $\mathbf{Co}_j$.*
Proof. *This is analogous to the proof of proposition 23.*□

Note that proposition 24 is only the contrapositive of proposition 23. They 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.

One of the most important properties of an extraction algorithm is the clarity of the set of rules extracted. What is the reason for extracting an enormous, although sound, set of rules if one can not understand and use it? To cope with this problem we do the following. 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 they 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 we derive a rule of the form $\rightarrow L_{j+1}$ (L_{j+1} is a fact) then L_{j+1} holds in the presence of any combination of the truth values of literals $L_1, ..., L_j$ in $\mathcal{I}$.*

Definition 27 is a important special case of definition 26. Considering the ordering on **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 **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 **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 that 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 [4]. There is a rather intricate relation between each rule of the form *m of n* and the ordering on the input vectors set **I**, in the sense that each valid *m of n* rule represents a subset of **I**. Here is a flavor 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, that is, constraint $\mathbf{C}_{o_x}$ has to be satisfied by at least one input vector in **I** in order for us to derive one or more rules. 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 $m < p$.*

[4] 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* $\sim ab \sim c \rightarrow x$.

Back to the illustration about the relation between $m\ of\ n$ rules and subsets of $\mathbf{I}$, let us see what happens if $sup(\mathbf{I})$, the elements at distance 1 from $sup(\mathbf{I})$, and the elements at distance 2 from $sup(\mathbf{I})$ all satisfy $\mathbf{C}_{o_x}$. We would expect that the set of seven rules obtained from $\mathbf{I}$ could be represented by $1(abc) \rightarrow x$, and in fact it is. 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 $sup(\mathbf{I})$ s. t. $0 \leq d' < d$ also satisfy $\mathbf{C}_{o_x}$.

Remark 2 *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 rule set generated by the extraction algorithm.*

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

- *Knowledge Extraction Algorithm for Regular Networks (Outline)*

1. Apply the *Transformation Algorithm* over N, obtaining its positive form N_+;
2. Find $\mathit{inf}(\mathbf{I})$ and $\mathit{sup}(\mathbf{I})$ 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 $\mathit{inf}(\mathbf{I})$. If $o_j > A_{min}$, apply the Simplification Rule *Fact* and stop.

    (b) Query N_+ with input vector $\mathit{sup}(\mathbf{I})$. If $o_j < A_{min}$, stop.

    /* *Search the input vectors space* $\mathbf{I}$.

    (c) $\mathbf{i}_\perp := \mathit{inf}(\mathbf{I})$; $\mathbf{i}_\top := \mathit{sup}(\mathbf{I})$;

    (d) While $\mathit{dist}(\mathbf{i}_\perp, \mathit{inf}(\mathbf{I})) \leq n\text{DIV}2$ or $\mathit{dist}(\mathbf{i}_\top, \mathit{sup}(\mathbf{I})) \leq n\text{DIV}2 + n\text{MOD}2$, where n is the number of input neurons of N_+, do:

    /* *Generate new* $\mathbf{i}_\perp$ *and* $\mathbf{i}_\top$ *from old* $\mathbf{i}_\perp$ *and* $\mathbf{i}_\top$, *respectively, and query the network.*

        i. set new $\mathbf{i}_\perp$:= old $\mathbf{i}_\perp$ flipped according to the ordering on $\mathbf{I}$;
        ii. Query N_+ with input vector $\mathbf{i}_\perp$;
        iii. If Search Space *Pruning Rule 2* is applicable, stop generating the successors of $\mathbf{i}_\perp$;
        iv. Apply the Simplification Rule *Complementary Literals*, and Add the rules derived accordingly to the rule set.

        v. set new $\mathbf{i}_\top$:= old $\mathbf{i}_\top$ flipped according to the ordering on $\mathbf{I}$;
        vi. Query N_+ with input vector $\mathbf{i}_\top$;
        vii. If Search Space *Pruning Rule 1* is applicable, stop generating the predecessors of $\mathbf{i}_\top$;
        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 rule set. 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 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 Simplification 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, $\mathbf{i}_2$ activates x, we can derive $ab \rightarrow x$, using Proposition 24 and Simplification 26. If not only $\mathbf{i}_2$ but also the other inputs in $\mathbf{I}^{**}$ activate x then we obtain $2(abc) \rightarrow x$, which subsumes $abc \rightarrow x$ 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 rule set or irrelevant and will not give rise to any new rule.

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

Proof. *We have to show that each rule extracted is either 1) obtained directly from querying the network or 2) a simplification of rules obtained from querying the network. Case 1 is trivial. In Case 2 we have to show that each simplification creates rules that subsume rules obtained from querying the network.*

Complementary Literals: Consider a set $\mathbf{I}$ *of p-ary input vectors* $\mathbf{i} = (i_1, i_2, ..., i_p)$ *under the set inclusion order. Let* $\mathbf{i}_m, \mathbf{i}_n \in \mathbf{I}$*, where* $dist(\mathbf{i}_m, \mathbf{i}_n) = 1$ *and* $\langle \mathbf{i}_m \rangle < \langle \mathbf{i}_n \rangle$*. Assume that* $\mathbf{i}_m$ *satisfies* $\mathbf{C}_{o_j}$*. The rule* $L_1, ..., L_i, ..., L_p \rightarrow L_j$ *can be derived by querying the network, where* L_i $(1 \leq i \leq p)$ *is a positive literal if* $i_i = 1$ *or a negative literal if* $i_i = -1$*, while* L_j *is the literal associated with the output neuron* o_j*. By Proposition 24,* $\mathbf{i}_n$ *also satisfies* $\mathbf{C}_{o_j}$ *and the rule* $L_1, ..., \sim L_i, ..., L_p \rightarrow L_j$ *can be derived. By Definition 26 (Complementary Literals), the rule* $L_1, ..., L_{i-1}, L_{i+1}, ..., L_p \rightarrow L_j$ *can be obtained, simplifying* L_i *and* $\sim L_i$*. And by Definition 25 (Subsumption),* $L_1, ..., L_{i-1}, L_{i+1}, ..., L_p \rightarrow L_j$ *subsumes both* $L_1, ..., L_i, ..., L_p \rightarrow L_j$ *and* $L_1, ..., \sim L_i, ..., L_p \rightarrow L_j$*.*

Fact: Now assume that $\mathbf{i}_m = inf(\mathbf{I})$ *and* $\mathbf{i}_m$ *satisfies* $\mathbf{C}_{o_j}$*. By Proposition 24,* $\mathbf{i}_o \in \mathbf{I}$ *also satisfies* $\mathbf{C}_{o_j}$*. By Definition 27 (Fact), the rule* $\rightarrow L_j$ *can be obtained. Since* $\mathbf{I} = \wp(\mathcal{I})$*,* $L_i \rightarrow L_j$ *and* $\sim L_i \rightarrow L_j$ *for all* $1 \leq i \leq p$*. Clearly, by Definition 25 (Subsumption),* $\rightarrow L_j$ *subsumes any rule derived from* $\mathbf{I}$ *to* L_j*.*

M of N and M of N Subsumption: Assume that for all $\mathbf{i}_m, \mathbf{i}_n \in \mathbf{I}$ *such that* $\langle \mathbf{i}_m \rangle = \langle \mathbf{i}_n \rangle$*,* $\mathbf{i}_m$ *and* $\mathbf{i}_n$ *satisfy* $\mathbf{C}_{o_j}$*. The rules* $\sim L_1, L_2, ..., L_p \rightarrow L_j$*;* $L_1, \sim L_2, ..., L_p \rightarrow L_j; ...;$ $L_1, L_2, ..., \sim L_p \rightarrow L_j$ *can be derived. Let q be the number of positive literals in the body of each rule derived. By Definition 28 (M of N), the rule* $q(L_1, ..., L_p) \rightarrow L_j$ $(q < p)$ *is obtained. By Definition 25 (Subsumption),* $L_1, ..., L_q \rightarrow L_j$ *subsumes* $L_1, ..., L_q, L_{q+r} \rightarrow L_j$ *and* $L_1, ..., L_q, \sim L_{q+r} \rightarrow L_j$*, and therefore* $q(L_1, ..., L_p) \rightarrow L_j$ *subsumes* $\sim L_1, L_2, ..., L_p \rightarrow L_j$*;* $L_1, \sim L_2, ..., L_p \rightarrow L_j; ...;$ $L_1, L_2, ..., \sim L_p \rightarrow L_j$*. The same result holds for any subset* $\mathcal{I}\prime$ *of* $\{L_1, ..., L_p\}$*, provided that* $q < |\mathcal{I}\prime|$*. Now, let* $s(L_1, ..., L_p) \rightarrow L_j$*. By Definition 25,* $L_1, ..., L_s \rightarrow L_j$ *subsumes* $L_1, ..., L_q \rightarrow L_j$ *for any set of s elements chosen from* $\{L_1, ..., L_p\}$ *s.t.* $s < q$*. As a result, any rule obtained as a simplification subsumes rules derived from querying the network.* □

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

Proof. *We have to show that the extraction algorithm terminates either when 1) all possible combinations of the input vector is queried in the network or 2) the elements not queried do not generate any new rule that is not subsumed by the rules already in the rule set. Case 1 is trivial. In Case 2, we have to show that the elements not queried either do not generate any rule at all (Case* $2(i)$*) or generate rules that are subsumed by the rules extracted (Case* $2(ii)$*).*

Case 2(i): Consider a set $\mathbf{I}$ *of* p*-ary input vectors* $\mathbf{i} = (i_1, i_2, ..., i_p)$ *under the set inclusion order. Let* $\mathbf{i}_m, \mathbf{i}_n \in \mathbf{I}$*, where* $dist(\mathbf{i}_m, \mathbf{i}_n) = 1$ *and* $\langle \mathbf{i}_m \rangle < \langle \mathbf{i}_n \rangle$*. Assume that* $\mathbf{i}_n$ *does not satisfy* $\mathbf{C}_{o_j}$*. By Proposition 23,* $\mathbf{i}_m$ *does not satisfy* $\mathbf{C}_{o_j}$*.*

Case 2(ii): Now assume that $\mathbf{i}_m$ *satisfies* $\mathbf{C}_{o_j}$*. The rule* $L_1, ..., L_p \rightarrow L_j$ *can be derived. Let* q $(q \leq p)$ *be the number of positive literals in the body of the above rule. Let* $\{L_1, ..., L_q\}$ *be the maximal subset of* $\{L_1, ..., L_p\}$ *only containing positive literals. Using Proposition 24 and Definition 26 (Complementary Literals), the rule* $L_1, ..., L_q \rightarrow L_j$ *can be obtained. By Definition 25 (Subsumption),* $L_1, ..., L_q \rightarrow L_j$ *subsumes any rule derived by querying the network with* $\mathbf{i}_n$*, since* $\langle \mathbf{i}_m \rangle < \langle \mathbf{i}_n \rangle$*.*

Therefore, for any $\mathbf{i}$ *not queried in the network, either* $\mathbf{i}_o$ *(*$\langle \mathbf{i}_o \rangle > \langle \mathbf{i} \rangle$ *in the chain) does not satisfy* $\mathbf{C}_{o_j}$*, or* $\mathbf{i}_o$ *(*$\langle \mathbf{i}_o \rangle < \langle \mathbf{i} \rangle$ *in the chain) satisfies* $\mathbf{C}_{o_j}$ *and the rule derived from* $\mathbf{i}_o$ *subsumes the rule derived from* $\mathbf{i}$*. As a result, any input vector not queried in the network either does not generate rules at all, or generates rules that are subsumed by rules already in the rule set.* □

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 find easily an ordering on the input vectors set, and apply a sound and complete pedagogical extraction algorithm that searches for relevant input vectors in that ordering. As a result, by means of the above defined Pruning Rules and Simplification Rules, we are able to reduce the complexity of the extraction algorithm by pruning the search space, and to enhance the readability of the rule set extracted by simplifying the rules derived. Furthermore, the neural network and its rule set 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 rule set.

The problem with non-regular networks is that it is difficult to find the ordering on the input vectors set without having to actually check each input in the network. In that 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 from each subnetwork into the final rule set. That problem is due mainly to the non discrete nature of the network's hidden neurons. As we have seen in Example 2, 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 algorithm is sound, although we will have to drop 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 analyze 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 analyze 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 neurons labeled 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 $(A_{min}, 1)$, 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.

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* $\mathbf{Co}_j$ *on the* BNSs *output neuron, then* $\mathbf{i}_m$ *does not satisfy* $\mathbf{Co}_j$ *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* $\mathbf{Co}_j$ *on the* BNSs *output neuron, then* $\mathbf{i}_n$ *also satisfies* $\mathbf{Co}_j$*.*
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[5]. 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* $\langle \mathbf{i}_m \rangle , \langle \mathbf{i}_n \rangle < \langle \mathbf{i}_o \rangle$*, that is,* $\mathbf{i}_m$ *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*

[5] 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.

$A_{min} > 0$. *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* $\left|W_{o_j i_k}\right| \leq \left|W_{o_j i_i}\right|$, *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,* $\mathbf{i}_m$ *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* $\left|W_{o_j i_k}\right| \leq \left|W_{o_j i_i}\right|$, *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 analogous to the proof of proposition 41.* □

Example 43 *Consider the network of figure 15 and its positive form at figure 16. It contains three* BNSs*; two* Input to Hidden BNSs, *having inputs* $\{a, b, c\}$ *and outputs* n_1 *and* n_2, *and one* Hidden to Output BNS, *having inputs* $\{n_1, n_2\}$ *and output* x.

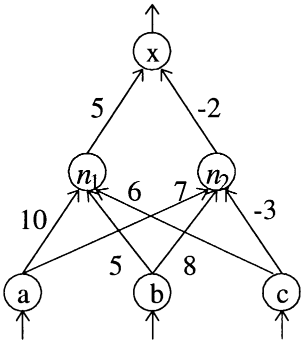


Figure 15: A non-regular network.

Applying the Transformation Algorithm on each BNS, *we verify that* (abc) *is the maximum element in the ordering of the* BNS *with output* n_1, *that*

$(ab \sim c)$ is the maximum element in the ordering of the BNS *with output n_2, and that $(n_1 \sim n_2)$ is the maximum element in the ordering of the* BNS *with output x (see figure 16).*

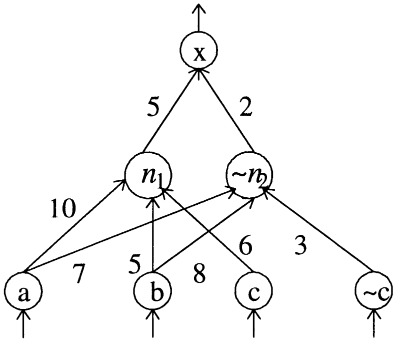


Figure 16: The positive form of a non-regular network, obtained by applying the Transformation Algorithm over its *BNSs*.

Figure 17(1) shows the set inclusion ordering on the Input to Hidden BNSs, *where* $(1,1,1) = (abc)$ *for the* BNS *with output* n_1 *and* $(1,1,1) = (ab \sim c)$ *for the* BNS *with output* n_2. *Figure 17(2) shows the set inclusion ordering on the* Hidden to Output BNS. *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.*

If now we add to the above ordering the information about BNSs *weights, we will be able to use the results of Pruning Rules 3 and 4 as well. Take, for example, the positive form of the* BNS *with output* n_1, *where* $W_{n_1b} \leq W_{n_1c} \leq W_{n_1a}$. *Using Pruning Rules 3 and 4, we can obtain a new ordering on the input vectors* $\mathbf{i}_m$ *and* $\mathbf{i}_n$ *such that* $\langle \mathbf{i}_m \rangle = \langle \mathbf{i}_n \rangle$. *We obtain* $(-1,1,1) \leq_{\langle\rangle} (1,1,-1) \leq_{\langle\rangle} (1,-1,1)$ *and* $(-1,1,-1) \leq_{\langle\rangle} (-1,-1,1) \leq_{\langle\rangle} (1,-1,-1)$. *Figure 18 contains a diagram where this new ordering is superimposed on the previous set inclusion ordering of the* BNS. *Dotted lines indicate relations that can be eliminated by transitivity.*

The above example illustrates the ordering $\preceq$ on the input vectors set $\mathbf{I}$ 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.

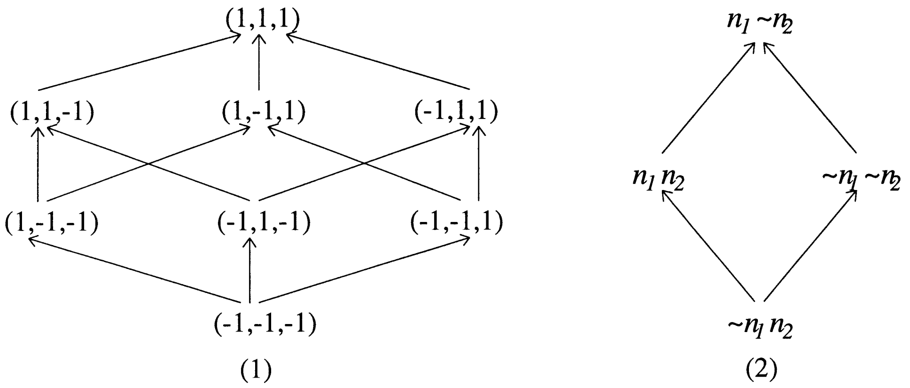


Figure 17: Ordering on *Input to Hidden BNSs* (1), where $(1,1,1) = (abc)$ w.r.t n_1 and $(1,1,1) = (ab \sim c)$ w.r.t n_2. Ordering on *Hidden to Output BNS* (2).

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 diagram 19 below (see also figure 18). Note also that $\preceq$ is a chain for the *BNS* with output x; given that $W_{x\sim n_2} \leq W_{xn_1}$, we derive $(\sim n_1, \sim n_2) \leq_{\langle\rangle} (n_1, n_2)$ and, therefore, $(\sim n_1, n_2) \preceq (\sim n_1, \sim n_2) \preceq (n_1, n_2) \preceq (n_1, \sim n_2)$.

Figure 20 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 $|W_a| + |W_b| + |W_c| + |W_d|$.

Incomparable elements in $\preceq$, as $\mathbf{i}_1 = (1, -1, -1)$ and $\mathbf{i}_2 = (-1, 1, 1)$ at figure 19, means that it is not easy to establish whether $\mathbf{i}_1 \preceq \mathbf{i}_2$ without actually querying the *BNS* with both inputs.

$\preceq$ 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 21(1), and its positive form 21(2). The ordering's maximum element is input vector* $\mathbf{i}_\top = (1, 1, 1, 1) = (a, b, \sim c, \sim d)$. *In other words, the maximum activation of* n_i *is*

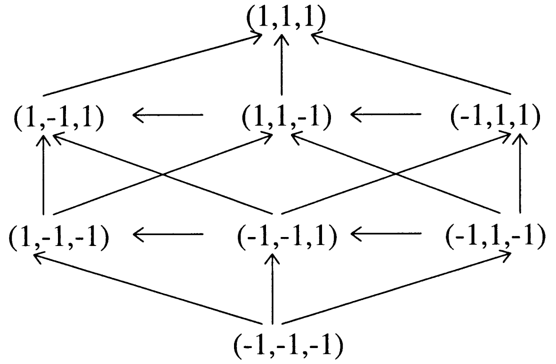


Figure 18: Adding information about the weights of the *BNS* with output n_1.

obtained when $(1,1,1,1)$ *is presented to (2). Equivalently, the maximum activation of* n_i *can be obtained when* $(1,1,-1,-1)$ *is presented to (1). Taking* BNS*(2), 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* $\mathbf{i}_m$*. Applied to this example, it says that, for instance,* $(1,1,1,-1) = (a, b, \sim c, d)$ *provides a smaller activation value to* n_i *than* $(1,1,-1,1) = (a, b, c, \sim d)$*.*

Therefore, given $W_{n_i \sim c} \leq W_{n_i a} \leq W_{n_i \sim d} \leq W_{n_i b}$*, if we start from* $\mathbf{i}_\top$ *by flipping from 1 to -1 the input* $\sim c$ *with the smallest weight* $W_{n_i \sim c}$*, we 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. If, alternatively, we start from* $\mathbf{i}_\top$ *by flipping from 1 to -1 the input* b *with the greatest weight* $W_{n_i b}$*, we obtain the input vector* $\mathbf{i}_2 = (1,-1,1,1)$*. By Pruning Rule 3, the activation of* n_i *given* $\mathbf{i}_2$ *is smaller than the activation of* n_i *given any other element* $\mathbf{i}_m$ *such that* $\langle \mathbf{i}_m \rangle = \langle \mathbf{i}_2 \rangle$*. And if* $n_i(\mathbf{i}_2) > A_{min}$ *then* $n_i(\mathbf{i}_m) > A_{min}$*. In this case, we could stop the search and derive the rule* $3(ab \sim c \sim d) \rightarrow n_i$*.*

Suppose we carry on the search within the elements derived from $\mathbf{i}_1$ *(see figure 22). If we flip from 1 to -1 the input* a *with the smallest weight* $W_{n_i a}$*, we obtain* $\mathbf{i}_3 = (-1,1,-1,1)$*. As before, if* $n_i(\mathbf{i}_3) < A_{max}$ *then for any* $\mathbf{i}_n$ *such that* $\langle \mathbf{i}_n \rangle = \langle \mathbf{i}_3 \rangle$ *and* $dist(\mathbf{i}_n, \mathbf{i}_1) = 1$*,* $n_i(\mathbf{i}_n) < A_{max}$*. Similarly, if we flip in* $\mathbf{i}_1$ *the input* b *with the greatest weight* $W_{n_i b}$*, we obtain* $\mathbf{i}_4 = (1,-1,-1,1)$*.*

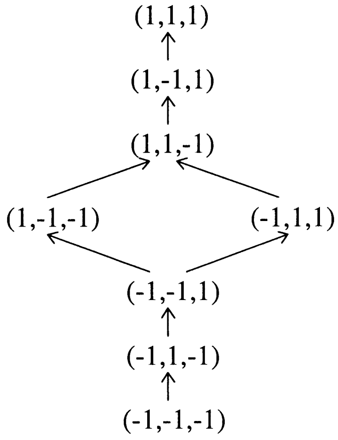


Figure 19: The ordering $\preceq$ on the input vectors set of the *BNS* with output n_1.

And if $n_i(\mathbf{i}_4) > A_{min}$ *then* $n_i(\mathbf{i}_n) > A_{min}$.

Figure 22 shows the search process for the *BNS* of figure 21, up to the elements with distance 2 from the maximum element.

The same searching and pruning procedure of Example 45 applies if we start from the ordering's minimum element $\mathbf{i}_\perp = (-1,-1,-1,-1) = (\sim a, \sim b, c, d)$. However, in this case, Pruning Rule 4 should be used.

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 that way, we avoid repeating input vectors. Figure **??** 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 24 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.

Note the symmetry between figures 23 and 24, reflecting, respectively,

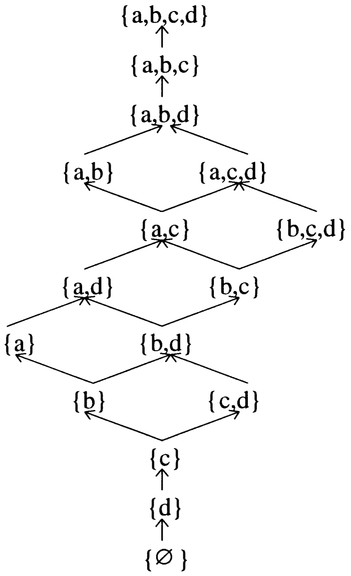


Figure 20: $\preceq$ on $\wp(\mathcal{I})$, given $\mathcal{I} = \{a, b, c, d\}$ and $(1, 1, 1, 1) = [a, b, c, d]$.

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 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*

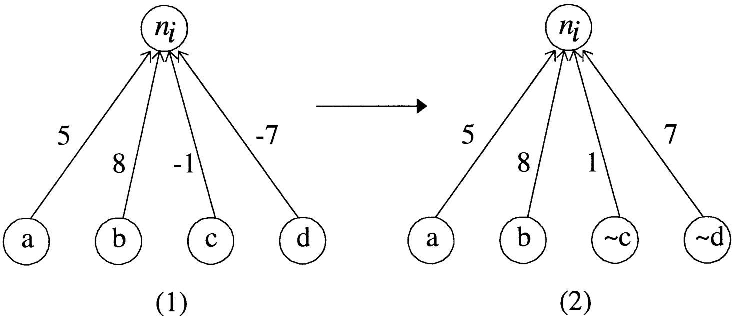


Figure 21: An *Input to Hidden BNS* (1), and its positive form (2).

$n_2 = 0.25$.

At this stage, we need to compromise in order to keep soundness. Roughly, we have to analyze 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 25. 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 then 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 analyze the case where it is equal to A_{min}. In this way, we can derive the ordering of figure 25 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 then assume $n_i = A_{min}$ and $\sim n_i = -1$. If the weight from n_i to o_j is*

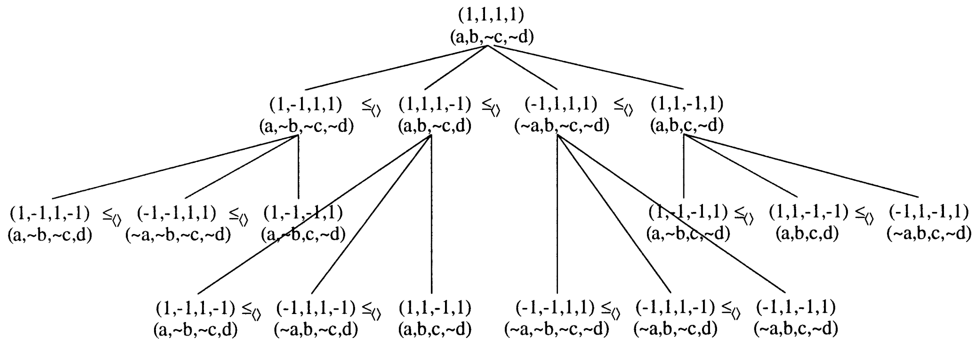


Figure 22: Searching for elements $\mathbf{i}_m$ and $\mathbf{i}_n$ s.t. $\langle \mathbf{i}_m \rangle = \langle \mathbf{i}_n \rangle$, and $dist(\mathbf{i}_m, \mathbf{i}_k) = 1$ and $dist(\mathbf{i}_n, \mathbf{i}_k) = 1$. Starting from $\mathbf{i}_\top$ and up to the elements $\mathbf{i}_j$ s.t. $dist(\mathbf{i}_j, \mathbf{i}_\top) = 2$.

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 3 *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 [5]). Note that one can always assume* $A_{max} = A_{min} = 0$.

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 26(*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 26(*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 as-

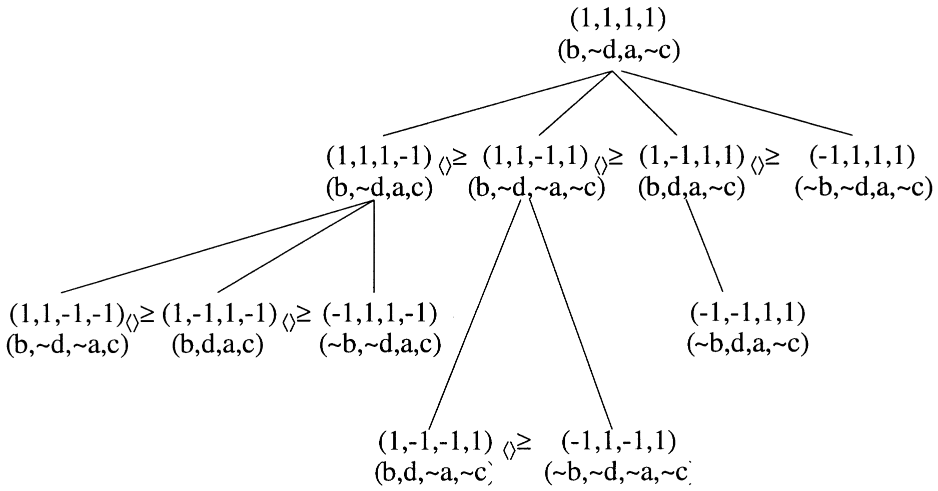


Figure 23: Systematically deriving input vectors from $\mathbf{i}_\top$ without repetitions.

sume that $\sim n_i = A_{max}$ and $n_i = 1$ for $i = \{1, 2\}$, as shown in figure 26(*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 $(A_{min}, -1) \leq_{()} (-1, A_{min})$ in figure 26(Case 1), $(A_{min}, 1) \leq_{()} (-1, A_{max})$ in 26(Case 2), and $(A_{max}, 1) \leq_{()} (1, A_{max})$ in 26(Case 3). Thus, the resulting orders $\preceq$ are chains, as expected. Note that the orders of figure 26 are valid for the original *BNSs,* and not for their positive forms.

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 $i_i \in \{-1, A_{min}\}$, $i'_i \in \{-1, -A_{min}, A_{min}, 1\}$, $x_i \in \mathcal{I}_+, 1 \leq i \leq s$. Recall 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 $i'_i = -i_i$ 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 orders of figure 27. 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 27(a). As before, the extraction process can be carried out by querying the* BNS's *positive*

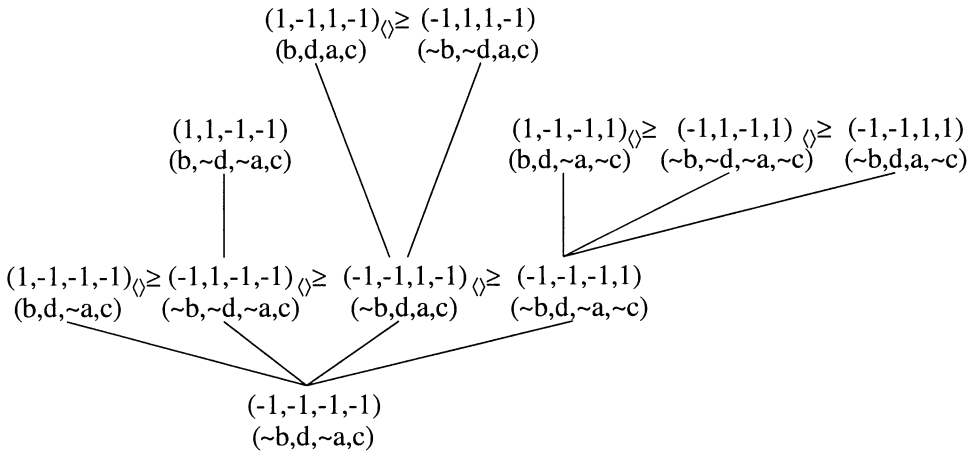


Figure 24: Systematically deriving input vectors from $\mathbf{i}_\perp$ without repetitions.

form with values $\{-1, A_{min}\}$, *following 27(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 23 and 24).*

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

- *Knowledge Extraction Algorithm - General Case (Outline)*

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 $\mathcal{B}_i^+$;
   (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;

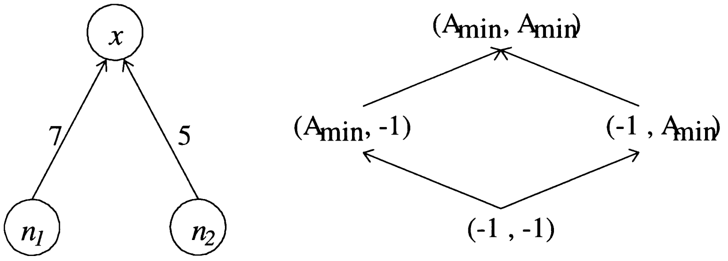
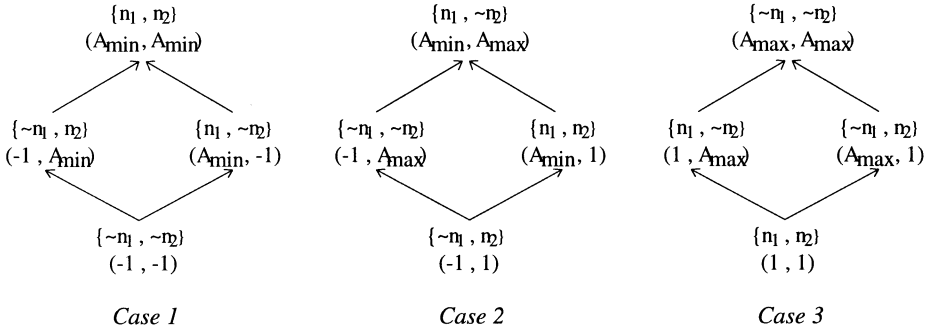


Figure 25: A *Hidden to Output BNS* and the corresponding set inclusion ordering on the hidden neurons activations in the worst case.



Case 1 *Case 2* *Case 3*

Figure 26: Orderings on *Hidden to Output BNSs* with two input neurons n_1 and n_2, using worst case analyses on $(-1, A_{max})$ and $(A_{min}, 1)$.

(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 to the extraction algorithm for regular networks, step 3d, the following lines:*

- 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 $\mathbf{i}_\top$;

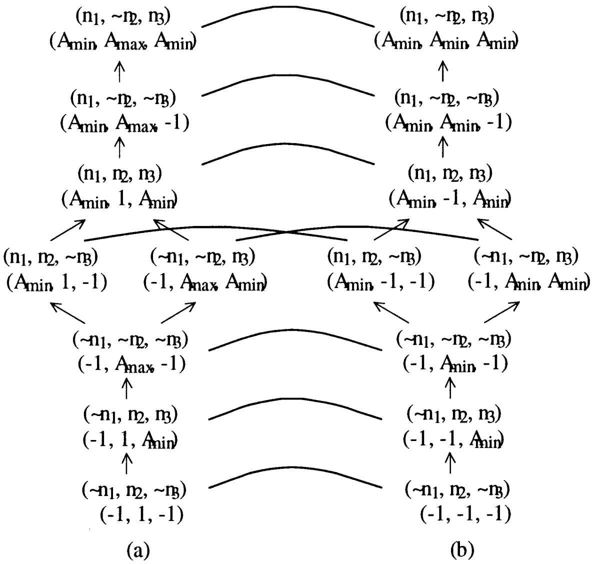


Figure 27: (a) $\preceq$ on a *Hidden to Output BNS* with three input neurons (n_1, n_2, n_3) and the associated activations in the worst case. (b) $\preceq$ on the *BNS's* positive form and the mapping σ from (b) and (a).

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 our 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 need 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 (e.g., the extracted rule $ab \rightarrow n_1$ can be substituted by the stronger $ab \leftrightarrow n_1$). For example, assume that the extraction algorithm derives $a \rightarrow n_1$ from $\mathcal{B}_1$ and $b \sim c \rightarrow n_2$ from $\mathcal{B}_2$. By Lemma 48, we have $a \leftrightarrow n_1$ and $b \sim c \leftrightarrow n_2$. In other words, any rule not subsumed by $a \rightarrow n_1$ implies $\sim n_1$, where $\sim n_i$ means that the activation of n_i is smaller than A_{max}. 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* $\mathcal{B}_3$. 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 we 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 of Table 49 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	$> A_{min}$	$< A_{max}$	$> A_{min}$

Table 50: Ranges of activation values in N for each input vector (a, b).

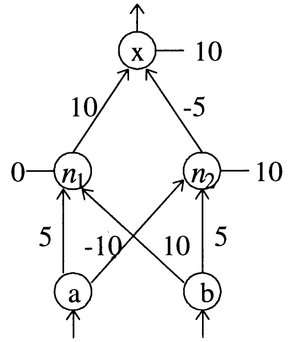
An exhaustive pedagogical extraction algorithm, that although inefficient, is provably sound and complete, would derive the unique rule $ab \rightarrow x$ from N.

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. Similarly, the rule* $\sim ab \rightarrow n_2$ *would be derived, since (-1,1) also activates* n_2.

Now, taking $A_{min} = 0.5$, *assume that given* $(ab) = (-1,1)$, *the actual activation values of* n_1 *and* n_2 *are, respectively, 0.6 and 0.95. Table 49 indicates that given these activations,* x *is non active. However, decompositional approaches such as [40] and [12], by assuming that hidden neurons are either fully active or non active, could wrongly derive the rule* $n_1 n_2 \rightarrow x$ *(unsoundness). This is because* $n_1 = 1$ *and* $n_2 = 1$ *may activate* x. *In order to solve this problem, we take the worst case activations of hidden neurons* $n_1 = A_{min}$ *and* $n_2 = A_{min}$, *so that we do not risk deriving incorrect rules. However, given* $(ab) = (1,1)$, *assume that the actual 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 (see Table 49).*

Finally, assume that we have been able to derive the rule $n_1 \sim n_2 \rightarrow x$ *from the* Hidden to Output BNS *of* N [6]. *The final rule set is assembled as follows. From* $\sim ab \rightarrow n_1$ *and* $ab \rightarrow n_1$ *we derive* $b \rightarrow n_1$; *from* $\sim ab \rightarrow n_2$ *we derive* $a \vee \sim b \rightarrow \sim n_2$; *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.*

An example of a neural network that presents the activations of Table 49 is given below.



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

Lemma 50 *The extraction of rules from* Hidden to Output BNSs *is sound.*
Proof. *If we are able to derive a rule* r *taking* $n_i \in \{-1, A_{min}\}$ *then, from the mononononically crescent characteristic of* $h(x)$, r *will still be valid if* $n_i \in \{[-1, -A_{min}], [A_{min}, 1]\}, A_{min} > 0$. □

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* $n_1 \rightarrow x$. □

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. Otherwise, one could carry on with the extraction process or even perform an exhaustive pedagogical extraction. Note that, in terms of accuracy, there is not much difference between having unsoundness or incompleteness.

6 Conclusion

We have seen that most decompositional methods for rules' extraction from trained neural networks are unsound. On the other hand, sound and complete pedagogical extraction methods have exponential complexity. We call this problem the *complexity* × *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, some *pruning rules* can be used to safely prune the network's input vectors search space during the extraction process. 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. We have also defined a set of *simplification* *rules.* These rules fit very well into the extraction method, since they have a counterpart graphical representation on the network's input vectors ordering. They help reducing considerably the length of the final set of rules extracted, enhancing the rule set comprehensibility, a major concern.

We carried on to extend 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 identified subnetworks that contain regularities. We showed 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. Here, however, we are applying a decompositional approach, and we need to investigate 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 take advantage of the completeness of *Input to Hidden BNSs* extraction to assemble the network's rule set. We show that for the case of non regular networks, although using a decompositional approach, we are able to maintain soundness. We have to drop completeness, however, as a result of the above worst case analysis.

A possible extension to the extraction algorithm concerns the extraction of *meta-level priorities* 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.

We have been investigating the relations between first order Horn clause logics and neural networks. We believe that, once such relations are established, we could take advantage of using a more expressive language for the extraction process. It may be the case that an extraction algorithm which uses a more powerful language, such as the language of First Order Logic, could help reducing the networks' input vectors search space, for example by using variables in the language for the extraction.

In this report, we have investigated the problem of extracting the symbolic knowledge encoded in trained neural networks. Although neural networks have shown very good performance in learnability, generalizability and speed 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 in some application areas, for instance in safety-critical domains, has become limited, if not unacceptable. This has motivated the first attempts towards finding the justification for neural networks' reasoning mechanisms, 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 encoded in the network. The problem of symbolic knowledge extraction from trained networks turned out to be one of the most interesting open problems in the field. So far, some extraction algorithms were proposed [2, 8, 13, 31, 36, 40] and had their effectiveness empirically confirmed using certain applications as benchmark. Some theoretical results have also been obtained [5, 13, 18, 39]. However, we are not aware of any extraction method that fulfills Thrun's list of desirable properties: 1) no architectural requirements; 2) no training requirements; 3) correctness; and 4) high expressive power [39]. The extraction algorithm presented here satisfies the above requirements 2 and 3. It does impose, however, some restrictions 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.[19], 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, we argue, 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.

The material presented in this report is an integral part of our neural-symbolic integration proposal, consisting on the final step of the system (*explanation,* see figure 1(4)). A detailed description of the system and the results obtained with its application in computational biology, specifically DNA sequence analysis, can be found in [5].

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, L. A. Carvalho; "*Logical Inference and Inductive Learning in Artificial Neural Networks*"; Workshop on Knowledge Representation and Neural Networks, ECAI96, Budapest; 1996.

[4] 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.

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

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

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

[8] 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.

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

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

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

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

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

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

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

[16] 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.

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

[18] 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.

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

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

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

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

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

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

[25] M. Minsky; "*Logical versus Analogical, Symbolic versus Connectionist, Neat versus Scruffy*"; AI Magazine, Vol. 12, no 2; 1991.

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

[27] 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.

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

[29] 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.

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

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

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

[33] 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.

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

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

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

[37] S. Sharma, "*Applied Multivariate Techniques*", John Wiley and Sons, 1996.

[38] 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.

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

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

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