Rule Extraction from Trained ANN: A Survey

Rough Draft

Ashish Darbari*

July 24, 2001

Abstract

A survey of several well known rule extraction techniques is presented in my report, in the light of a broader paradigm of connectionist-symbolic learning. In the first part of the report I have covered some introductory aspects about machine learning, and investigated the reasons for a possible connectionist-symbolic integration, thereby presenting a hybrid learning framework. Within this hybrid learning framework, my report focuses on the survey of Rule extraction techniques from Trained Artificial Neural Networks (ANNs). I have presented the techniques, and thereby compare them on several grounds, and investigate the relative merits and demerits of each one of them. Towards the end, there seems to be plenty of crucial issues still unattended by all of these rule extraction techniques.

1 Introduction

Since the computers were first invented we have wondered if ever it would be possible for them to learn. We know that by programming them to learn - to improve automatically with experience - the impact would be dramatic just imagine a similar case of an infant who learns and learns over a span of few years to perform tasks of arbitrary complexity with tremendous ease. Imagine having a computer learn from medical records which treatments are most effective for new diseases, homes learning from experience to optimize

*e-mail address : `ashish@pikas.inf.tu-dresden.de,ashdarbari@hotmail.com`

energy costs based on the particular usage pattern of their occupants, or personal software assistants learning the evolving interests of their users in order to highlight especially relevant stories from the on-line morning newspaper. We do not yet know how to make computers learn nearly as well as people learn, however with the emergence of algorithms which are effective for certain types of learning tasks a theoretical understanding of learning is beginning to emerge. How do we then define the task of learning ?

Definition 1 A computer program is said to *learn from experience* E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. For example, a computer program that learns to play checkers might improve its performance as measured by its ability to win at the class of tasks involving playing checkers games, through experience obtained by playing games against itself. In general, to have a well defined learning problem we should identify these three features: *the class of tasks*, *the measure of performance to be improved*, and *the source of experience*. Designing a machine learning approach involves a number of design choices, including choosing the type of training experience, the *target function* to be learned, a representation for this target function, and an algorithm for learning the target function from training examples. Throughout the machine learning paradigm it is useful to reduce the problem of improving performance P at a task T to the problem of learning a function which is called a target function. Intuitively, a target function maps the intended problem solution as a concrete mathematical function which the system is expected to learn. As an example, in the checkers learning problem a possible target function V could map any legal board state from a set of legal states B to some real value in a set of real numbers denoted by R. Learning involves search: searching through a space of possible hypothesis, to find the hypothesis that best fits the available training examples and other prior constraints or knowledge. The different machine learning approaches can be basically classified into *Inductive Learning* and *Analytical Learning*. The subsequent paragraphs investigate the details of these methods and then proceeds to justify the combination of these two methods in order to have a sufficiently general learning scheme which is called the *hybrid learning* framework.

1.1 Inductive Learning

The phrase Inductive Learning is used for the class of methods that generalize from observed training examples by identifying features that empirically distinguish positive from negative training examples. Decision tree learning [36], neural network learning such as *backpropagation* [4, 51, 34, 7, 8], inductive logic programming [33], and genetic algorithms are all examples of inductive methods that operate in this fashion. Pure inductive learning methods formulate general hypothesis by finding empirical regularities over the training examples. The key practical limit on these inductive learners is that they perform poorly when insufficient data is available. In fact there are theoretical analysis tools to show that there are fundamental bounds on the accuracy that can be achieved when learning inductively from a given number of training examples. In inductive learning, the learner is given a hypothesis space H from which it must select an output hypothesis, and a set of training examples D={ $\langle x_1, f(x_1)\rangle, \ldots \langle x_n, f(x_n)\rangle$ } where $f(x_i)$ is the target value for the instance x_i. The desired output of the learner is a hypothesis h from H that is consistent with these training examples.

1.2 Analytical Learning

Analytical learning uses prior knowledge and deductive reasoning to augment the information provided by the training examples, so that it is not subject to these same bounds. In analytical learning, the input to the learner includes the same hypothesis space H and training examples D as for inductive learning. In addition, the learner is provided an additional input - a *domain theory*. A domain theory B consists of background knowledge that can be used to explain observed training examples. The desired output of the learner is a hypothesis h from H that is consistent with both the training examples D and the domain theory B. As an example consider a chess playing scenario where each instance x_i would describe a particular chess position, and $f(x_i)$ would be *True* when x_i is a position for which black will lose its queen within two moves, and *False* otherwise. A hypothesis space H might be defined to consist of sets of Horn clauses (if-then rules). The domain theory B would consist of a formalization of the rules of chess, describing the legal moves, the fact that players must take turns, and the fact that the game is won when one player captures her opponent's king. It is important in analytical learning that the learner must output a hypothesis that is consistent with *both* the training data and the domain theory. The notion of consistency is a usual one where a hypothesis h is consistent with domain theory B provided $B \nvdash h$. In other words the hypothesis should not be derivable from the domain theory B. This additional constraint that the output hypothesis must be consistent with B reduces the ambiguity faced by the learner when the data alone cannot resolve among all hypothesis in H. The net effect, provided the domain theory is correct, is to increase the accuracy of the output hypothesis. A very special form of analytical learning is Explanation-Based Learning (EBL). In explanation based learning, prior knowledge is used to analyze, or explain how each observed training example satisfies the target concept. This explanation is then used to distinguish the relevant features of the training example from the irrelevant, so that examples can be generalized based on logical rather than statistical reasoning.

1.3 The Need for Synergy

Two distinctly different machine learning paradigms have been examined above and the way they work gives us enough intuition to delve into the possibility of combining them. These two learning paradigms are based on fundamentally different justifications for learned hypothesis and offer complementary advantages and disadvantages. Can a combination of these two give us a powerful learning tool ? The subsequent section discusses this in detail.

1.3.1 Theoretical Motivation

Purely analytical learning methods offer the advantage of generalizing more accurately from less data by using prior knowledge to guide learning. However, they can be misled when given incorrect or insufficient prior knowledge. It is rarely the case that the real time domain theories are perfect, on the contrary most of the time the domain theory is imperfect. Purely inductive methods offer the advantage that they require no explicit prior knowledge and learn regularities based solely on the training data. However, they can fail when given insufficient training data, and can be misled by the implicit inductive *bias* they must adopt in order to generalize beyond the observed data. Bias can be understood as an apriori assumption regarding the identity of the target concept in order to have a rational basis for classifying any unseen instances of data. The inductive bias can be formally defined under the following setting: Consider a concept learning algorithm L for the set of instances X. Let c be an arbitrary concept defined over X, and let $D_c = \{ \langle x, c(x) \rangle \}$ be an arbitrary set of training examples of c. Let $L(x_i, D_c)$ denote the classification assigned to the instance x_i by L after training on the data D_c. The inductive bias of L is any minimal set of assertions B such that for any target concept c and corresponding training examples D_c

$$(\forall x_i \in X)(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c) \tag{1}$$

The difference between inductive and analytical learning methods can be seen in the nature of the *justifications* that can be given for their learned hypothesis [32]. Hypothesis output by purely analytical learning methods carry a *logical* justification; the output hypothesis follows *deductively* from the domain theory and the training examples. Hypothesis output by purely inductive learning methods such as backpropagation carry a *statistical* justification; the output hypothesis follows from statistical arguments that the training sample is sufficiently large that it is probably representative of the underlying distribution of examples. Given that analytical methods provide logically justified hypothesis and inductive methods provide statistically justified hypotheses, it is easy to see why combining them could be useful. Logical justifications are only as compelling as the assumptions, or prior knowledge, on which they are built. They are suspect or powerless if prior knowledge is incorrect or unavailable. Statistical justifications are only as compelling as the data and statistical assumptions on which they rest. They are suspect or powerless when assumptions about the underlying distributions cannot be trusted or when the data is scarce.

1.3.2 Practical Motivation

Artificial Neural Networks (ANN) have been successfully applied in the area of speech generation [44], speech recognition [50], vision and robotics [35, 18], handwritten character recognition [28], medical diagnostics [22] and game playing [47] . However their inability to produce plausible explanations render them useless for safety-critical applications like Power System Protection where remedial actions have to be taken based upon the explanation generated for a fault. On the other hand symbolic learning systems have been used successfully to solve problems such as learning which chemical bonds fragment in a mass spectrometer [10, 23], learning which chemical substructures produce mutagenic activity (a property related to carcinogenicity) [1], and learning to design finite element meshes to analyze stresses in physical structures [3]. Systems like Prolog-EBG [39] construct useful intermediate features as a side effect of analyzing individual training examples. This analytical approach to feature generation compliments the statistically based generation of intermediate features (e.g., hidden unit features) in inductive methods such as backpropagation. One important class of problems for which a correct and complete domain theory can be found is the class of large state-space search problems. In such a scenario, we have a very strong motivation to guide the integration of these two forms of learning. An issue of foremost importance at this point is to emphasize that much of the research is focussed on combining *neural network* learning as the inductive form of learning and *rule based* learning for symbolic learning. In the subsequent paragraphs a general framework for this integration has been studied.

2 Hybrid Learning

2.1 The General Framework

There is a significant gap between the knowledge-intensive, learning-by-being-told approach of hand built classifiers and the virtually knowledge-free approach of inductive learning. Some of this gap is filled by *hybrid* learning methods, which use both hand-constructed rules and classified examples during learning. Several trends have made the development of such systems an active area in machine learning. Perhaps the most important of these trends is the realization that knowledge intensive and knowledge-free learning are just two ends of a spectrum along which an intelligent system may operate. Staying at one end or the other of the spectrum of possible learning systems simplifies the learning problem by allowing strong assumptions to be made about the nature of what needs to be learned. However, the middle ground is appealing; it offers the possibility that synergistic combinations of theory and data will result in powerful learning systems.

Another trend spurring the development of hybrid systems is the growth of a body of psychological evidence that people rarely, if ever, learn purely from theory or examples. For instance, Murphy and Medin suggest that "feature correlations are partly supplied by people's theories and that the causal mechanisms contained in theories are the means by which correlational structure is represented." That is theory and examples interact closely during human learning. While it is clear that people learn from both theory and examples, the way in which the interaction occurs is yet to be determined. Jude and Shavlik in [45] have defined the scope of hybrid learning by the following three items:

- Knowledge Insertion into ANN (Rules to ANN)
- Refinement of this prior knowledge in its neural representation (Training the ANN)
- Refined Rule Extraction from trained ANN (Rule Extraction Problem)

The Rules to ANN module and the Training the ANN module are discussed in extensive detail in [45] and hence I am, not going to discuss them in detail, for the sake of completion, I would discuss the main issues briefly in a later section. Now I would like to focus on the core issue of this whole hybrid learning paradigm - which is the Rule Extraction Module. We begin in this section, by defining the rule-extraction problem and then discuss several closely related issues, examining in depth the fundamentally different kinds of rule extraction techniques.

Craven and Shavlik [49] propose the problem of rule-extraction from neural networks task as : "Given a trained neural network and the examples used to train it, produce a concise and accurate symbolic description of the network".

Craven in his PhD thesis [48] explains the problem as: "Given a trained neural network and the data on which it is trained, produce a description of the network's hypothesis that is *comprehensible* yet *closely approximates* the network's predictive behavior".

Close approximation to the network's predictive behaviour gives us an idea of the error tolerance that is permissible in the case of rule extraction. The notion of *comprehensibility* deserves special attention here as it is one of the overlooked issues in rule extraction techniques and also because to some extent this notion is useful in characterizing the difference between inductive and analytical learning. The importance of this criterion is argued by Michalski [40] in his *comprehensibilty postulate*:

> *The results of computer induction should be symbolic descriptions of given entities, semantically and structurally similar to*

those a human expert might produce observing the same entities. Components of these descriptions should be comprehensible as single "chunks" of information, directly interpretable in natural language, and should relate quantitative and qualitative concepts in an integrated fashion.

There are a number of reasons why comprehensibility could be so important [48].

- **Validation:** In order to gain confidence in the performance of a learning system, its users often want to know how it arrives at its decisions. The ability to inspect a learned hypothesis is important in such domains. It is an especially important criterion in domains, such as medical diagnosis (e.g., [16]), in which the system occupies a position of trust.

- **Discovery:** Learning systems may also play an important role in the process of scientific discovery. A system may discover salient features and relationships in the training data whose importance was not previously recognized. If the hypothesis formed by the learner are comprehensible, then these discoveries can be made accessible to human review (e.g., [25]).

- **Explanation:** In some domains, it is not necessary to have a complete description of the learning system's induced model, but it is desirable to be able to explain the classification of individual examples. If the learned hypothesis is understandable in such a domain, then it can be used to produce explanations of classifications made for particular cases.

- **Improving predictive accuracy:** The feature representation used for a learning task can have a significant impact on how well an algorithm is able to learn and generalize. Learned models that can be understood and analyzed may provide insight into devising better feature representations.

- **Refinement:** Rule Extraction techniques should be able to refine approximately correct domain theories. An approximate domain theory is an incomplete description of how to solve the problem at hand. In order to complete the theory - refinement process it is important to be able to express, in a comprehensible manner, the changes that have been imparted to the theory during learning.

There are two general schemes of extracting rules out of the network. One approach is to extract a set of *global* rules that characterize the output classes directly in terms of the inputs. The other suggests to extract *local* rules by decomposing the multi-layer network into a collection of single-layer networks. A multi-layer network is the one which has at least one hidden layer in between the input and the output layer. The rules for the individual units are then combined into a set of rules that describe the network as a whole. Local methods are computationally less expensive than global methods. A slightly different notion on rule-extraction is suggested by [5], it is a generalization of the notions mentioned above.

Definition 2 "Given a particular set of weights W_{ij} and biases θ_i, resulting from a training process on a neural network, find for each input vector **i**, all outputs o_j in the corresponding output vector **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 **i** iff $o_j > A_{min}$). A_{min} is value guided by heuristics, and is such a value where the transition of a unit from a non-active state to an active state takes place.

It is interesting to observe that [5] has a slightly different formulation of the rule extraction problem and also a different approach towards the rule extraction task than [13]; however both the groups who have been very successful have identified some key criteria for evaluation of algorithms and also they share the common notion of having two general schemes of rule extraction. In [5], the local method of rule extraction is referred as *decompositional* and a global method as a *pedagogical* one. We examine this is depth in a later section. Now the preliminaries useful for understanding the complete scenario, are explained in the following section.

3 Preliminaries

3.1 Artificial Neural Networks

There is a wide variety of neural network architectures and learning methods for both unsupervised and supervised inductive learning tasks. However most of the research in hybrid learning has been focussed on feed-forward neural networks and hence the discussion below is restricted to this particular type of neural networks.

3.1.1 Neural Network

As shown in Figure 1, a feed-forward neural network is composed of several layers of simple processing *units*. In the figure one can see a single hidden layer neural network with input units being labelled by i_i, for $1 \leq i \leq 4$, the hidden layer units are labelled by n_j, for $1 \leq j \leq 3$ and the output layer units by o_k, for $1 \leq k \leq 4$. The state of a given unit is represented by its *activation*, which is a real-valued number, typically in the range [0,1] or in the range [-1,1]. The *input layer* of a network contains units whose activations represent values for the features of the problem domain in which the network is being applied. Typically, a real-valued feature is represented by a single input unit, and a discrete feature with n possible values is represented by n input units. The units in the *output layer* of a network represent the decisions made by the network. Interposed between the input units and the output units, there can be a number of *hidden layers* of units. The units of a network are related by weighted connections. A network for classification

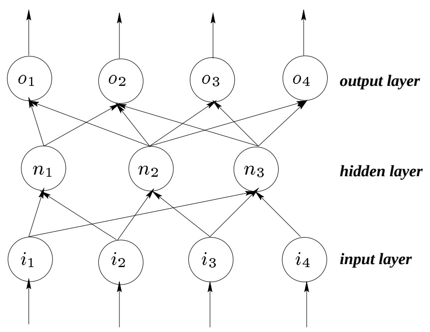


Figure 1: A single hidden layer feed forward neural network, the links connecting the units across the layers are assigned some real number values denoting the weight of the link.

that has only input units and output units is capable of representing only linear decision boundaries in its input space [30]. In order to represent more complex boundaries, it is necessary to add hidden units to the network. The role of hidden units is to transform the input space into another space in which it is more profitable for the output units to make linear discriminations. Computation in a feed-forward network proceeds by setting the activation values of the input units to represent a particular instance in the problem domain. The activation of the input units feed forward through the weighted connections to the units at the hidden layers and then to units at the output layer. The answer provided by the network is determined by the resultant activations on the output units.

For a particular example, the net input to a unit in a hidden or output layer is given by:

$$net(i) = \sum_j W_{ij}h(j) + \theta_i \tag{2}$$

where W_{ij} is the weight from unit j to unit i, $h(j)$ is the activation of unit j in response to the example, and θ_i is the *bias* for unit i. The bias of a unit, which is an adjustable parameter, can be thought of as the unit's predisposition to have a high (or low) activation before it receives any activation signals from other units. The activation of a hidden or an output unit is determined by passing its net input through a *transfer function* (sometimes called an *activation function*). One commonly used transfer function is the *logistic function*:

$$h(i) = \frac{1}{1 + e^{-net(i)}} \tag{3}$$

3.1.2 Learning with Neural Networks

One of the most widely used learning algorithms is the *backpropagation algorithm* which has been the cornerstone of neural learning for quite sometime [4, 6, 34, 51]. Learning in a neural network involves modifying the weights and biases of the network in order to minimize a *cost* function. The cost function always includes an error term - measure of how close the network 's predictions are to the class labels for the examples in the training set. Additionally, it may include a complexity term that reflects a prior distribution over the values that the parameters can take [9]. An appropriate cost function for the classification problems is the *cross-entropy function* [12]:

$$C = -\sum_i \sum_j [t_j ln(h(j)) + (1 - t_j) ln(1 - h(j))] \tag{4}$$

Here i ranges the examples in the training set, j ranges over the output units of the network, t_j is the *target value* for the j th output unit for a given example, and $h(j)$ is the activation of the j th output unit in response to the example. The target value for an output unit is the activation value that it should have for a given example.

In almost all neural-network methods, the function implemented by the network is continuous and being continuous implies being differentiable. Therefore, the cost function can be minimized by calculating its partial derivatives with respect to each of the network's parameters, and making changes to the parameters as follows:

$$\triangle \vec{W} \propto -\nabla_{\vec{W}} C \tag{5}$$

where $\vec{W}$ represents the vector of weights and biases in the network.

For a particular weight from unit j to unit i, the partial derivative is given by:

$$\frac{\partial C}{\partial W_{ij}} = \frac{\partial C}{\partial net(i)} \frac{\partial net(i)}{\partial W_{ij}} = \frac{\partial C}{\partial h(i)} \frac{\partial h(i)}{\partial net(i)} \frac{\partial net(i)}{\partial W_{ij}} \tag{6}$$

Since hidden and output units use different transfer functions, $\partial h(i)/\partial net(i)$ is easy to calculate. From equation 1, we can see that the derivative $\partial net(i)/\partial W_{ij}$ is simply $h(j)$. For the output units, the term $\partial C/\partial h(i)$ can be calculated directly from the cost function. For hidden units, however, it must be calculated by "backpropagating" error through the network. Specifically, to determine $\partial C/\partial h(i)$ for the hidden unit i, we compute the sum:

$$\frac{\partial C}{\partial h(i)} = \sum_k [\frac{\partial C}{\partial h(k)} \frac{\partial h(k)}{\partial net(k)} W_{ki}] \tag{7}$$

where k ranges over the units to which unit i is connected. The process of backpropagating errors is one of "blame" assignment. The activations of the output units are determined by the activations of the hidden units, which are in turn determined by the activations of the input units (or a lower level of hidden units). Thus the error at the output units may be due, not only to the weights directly connected to the outputs, but also to weights farther down in the network (the weights impinging on the hidden units).

Various optimization algorithms can be used to minimize the cost function, the most popular one being *on-line backpropagation.* This method can be thought of as a stochastic form of *gradient descent* in that weight changes do not follow the gradient of the cost function for the entire training set, but instead they follow the gradient for a single training example. The method also has a deterministic interpretation [26, 27] in which the cost function does not decrease monotonically. In order to avoid oscillations, these weight changes usually also incorporate a *momentum* term which is a time-decaying average of previous weight changes. Other optimization methods can be used to minimize the cost function. Standard gradient descent augmented with a momentum term is sometimes used, as is the conjugate-gradient method [15], and even second-order methods [37].

Often network training is stopped before a local minima in the cost function is reached. The motivation underlying this technique of *early stopping* is that over-fitting may occur if the network is trained to fit the training data too closely. One method for estimating a good stopping point is to use a validation set to monitor the predictive accuracy of the network as it is being trained. Instead of saving the network weights corresponding to the cost-function minimum, this procedure saves the weights from the iteration of the optimization method that results in the highest validation-set accuracy.

3.1.3 Neural Network and Comprehensibility

A hypothesis learned by a neural network is defined by (a) the topology of the network (b) the transfer functions used for the hidden and output units, and (c) the real-valued parameters associated with the network connections (i.e., the weights) and units (i.e., the biases of sigmoid units).

Hypothesis learned by neural networks are difficult to comprehend for several reasons. First typical networks have hundreds or thousands of real-valued parameters. These parameters encode the relationships between the input features $\vec{x}$, and the target value, y. Although single parameter encodings of this type are not usually hard to understand, the sheer number of parameters in the network make the task of understanding them daunting. Moreover, in multi-layer networks, these parameters may represent nonlinear, non-monotonic relationships between the input features and the target value, because this effect may be mediated by the values of other features. These nonlinear, non-monotonic relationships are represented by the hidden units in a network which combine the inputs of multiple features thus allowing the model to take advantage of dependencies among the features. Understanding the hidden units themselves is often difficult because these units often learn distributed representations [11]. Hidden units can be thought of as representing higher-level, "derived features". In a distributed representation, however these derived features may not correspond to well understood features in the problem domain. Instead features which are meaningful in the context of the problem domain are often encoded by *patterns* of activation across many hidden units. Similarly each hidden unit may play a part in representing numerous derived features.

3.2 Partial Orders

The Partial Order Approach propounded by D. Gabbay, K. Broda and A. d' A. Garcez [5], considers the rule extraction process as a search problem, they define some useful orderings on the input vector space, and in this section preliminaries useful for understanding this approach are underlined.

- partial order: A partial order is a *reflexive, transitive and antisymmetric* relation on a set

- total order: A binary relation $\leq$ on a set X is total *if for every* x *,* $y \in X$ *, either* $x \leq y$ *or* $y \leq x$

  Let $[X, \leq]$ be a partial ordered set, and $x, y, z, w, z', w' \in X$. Then the following are defined as follows:

- immediate predecessor: x is the immediate predecessor of y if $x \leq y$ and there is no other element z in X for which $x \leq z \leq y$

- immediate successor: x is the immediate successor of y if $x \geq y$ and there is no other element z in X for which $x \geq z \geq y$

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

- *minimum:* x is a minimum if for all elements $y \in X$, $x \leq y$

- *inf(X):* If $\leq$ is also an antisymmetric relation, and if x is a minimum element, then x is unique and will be denoted as *inf(X)*

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

- *maximum:* x is a maximum if for all elements $y \in X$, $y \leq x$

- *sup(X):* If $\leq$ is also antisymmetric and if x is a maximum element, then x is unique and will be denoted as *sup(X)*

- *least upper Bound:* For any $x, y \in X$, the least upper bound of x and y is the element z such that $x \leq z$ and $y \leq z$, and if there is any element z' with $x \leq z'$ and $y \leq z'$ then $z \leq z'$

- *greatest lower Bound:* For any $x, y \in X$, 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 $w' \leq w$

- *lattice:* A partial 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 $x \cdot y$ is called a lattice. A lattice L is distributive if $x + (y \cdot z) = (x + y) \cdot (x + z)$ and $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.

- metric: Let U be a set, then $f: U \times U \rightarrow \Re$ is a metric if it satisfies the following conditions:

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

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

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

- $f(x, y) \leq f(x, z) + f(z, y)$

- metric space: A *metric space* is a tuple $\langle U, f \rangle$.

- bounded metric: A metric f on U is *bounded* iff for some constant $k \in \Re, f(x, y) \leq k$, for all $x, y \in U$

4 Local and Global Methods

Generally speaking, there are two types of approaches to extracting rules from multi-layer networks. One approach is to extract a set of *global* rules that characterize the output classes directly in terms of the inputs. The alternative is to extract *local* rules by decomposing the multi-layer network into a collection of single-layer networks. In literature, the global procedure of rule extraction is often referred to as a *pedagogical* method and the local procedure as a *decompositional* method. A set of rules is extracted to describe each individual hidden and output unit in terms of the units that have weighted connections to it. The rules for the individual units are then combined into a set of rules that describes the network as a whole.

Many approaches to rule extraction have set up the task as a search problem. This search problem involves exploring a space of candidate rules, and testing individual candidates against the network to see if they are valid rules. Several research groups have investigated rule-extraction methods which operate by conducting a search through a space of possible conjunctive rules. Unlike most of the search processes which continue until a goal is found, a rule-extraction search continues until all or most of the maximally-general rules have been found. Obviously when exploring a rule space, it is inefficient for the search procedure to visit a node multiple times. In order to avoid this inefficiency, one can impose an ordering on the literals thereby transforming the search space into a tree. One of the problematic issues that arises in search-based approaches to rule extraction is that the size of the rule space can be very large. For a problem with n binary features, there are 3^n possible conjunctive rules (since each feature can be absent from a rule antecedent, or it can occur as a positive or a negative literal in the antecedent). To address this issue, a number of heuristics have been employed to limit the combinatorics of the rule exploration process. One of the first rule-extraction methods developed by Saito and Nakano in [24] employs a breadth-first search process to extract conjunctive rules in binary problem domains. To deal with the combinatorics of the problem, Saito and Nakano use two heuristics. The first heuristic limits the number of literals in the antecedents of extracted rules. Specifically, their algorithm uses two parameters, k_{pos} and k_{neg}, that specify the maximum number of positive and negative literals respectively that can be in an antecedent. By restricting the search to a depth of k, the rule space considered is limited to a size given by the following expression:

$$\sum_{i=0}^{k} \begin{pmatrix} n \\ k \end{pmatrix} 2^k$$

For fixed k, this expression is polynomial in n, but obviously, it is exponential in the depth k. This means that exploring a space of rules might still be intractable since, for some networks, it may be necessary to search deep in the tree in order to find valid rules. The second heuristic employed by Saito and Nakano use the following method to test the rule against the network. The input units corresponding to the positive literals are set to an activation of 1, and all other input units are set to an activation of 0. Activations are forward-propagated through the network and the decision procedure is used to classify the instance. Next, the input units corresponding to the negated literals are given an activation of 1, the other inputs are not changed, and once again the instance is classified by the network. If in the first case the network classified the instance as a member of the class of interest, and in the second case it did not, then the rule is accepted. Because the depth of the search for negative literals is limited, this procedure may accept rules that do not agree with the network. For example if the search space is limited to antecedents with two negated literals, and one of the valid rules describing the network is $x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \rightarrow y$, then this procedure might accept the overly general rule $x_1 \wedge \neg x_2 \wedge \neg x_3 \rightarrow y$.

Gallant in [19] developed a similar rule-extraction technique, which like the method of Saito and Nakano, manages the combinatorics of searching for rules by limiting the search depth. The principal difference between the two approaches is the procedure used to test rules against the network. Unlike Saito and Nakano's method, Gallant's rule-testing procedure is guaranteed to accept only rules that are valid. This method operates by propagating *activation intervals* through the network. The first step in testing a rule using this method is to set the activations of the input units that correspond to the literals in the candidate rule. The next step is to propagate activations through the network. The key idea of this second step, however, is that it is assumed that input units whose activations are not specified by the rule could possibly take on any allowable value, and thus an interval of activations is propagated to the units in the next layer. Effectively, the network is computing, for the examples covered by the rule, the range of possible activations in the next layer. Activation intervals are then further propagated from the hidden units to the output units. At this point, given the conditions specified by the rule, the range of possible activations for the output units can be determined, and thus the procedure can decide whether to accept the rule or not. This algorithm is guaranteed to accept only rules that are valid. However it may fail to accept maximally general rules, and instead may return overly specific rules. The reason for this deficiency is that in propagating activation intervals from hidden units onward, the procedure assumes that the activations of the hidden units are independent of one another. In most networks this assumption is unlikely to hold.

Thrun in [43] developed a method called *validity interval analysis* (VIA) that is a generalized and more powerful version of Gallant's technique. Like

Gallant's method, VIA tests rules by propagating activation intervals through a network after constraining some of the input and output units. The key difference is that Thrun frames the problem of determining *validity intervals* (i.e., valid activation ranges for each unit) as a linear programming problem. This is an important insight because it allows activation intervals to be propagated *backward*, as well as forward through the network, and it allows arbitrary linear constraints to be incorporated into the computation of validity intervals. Backward propagation of activation intervals enables the calculation of tighter validity intervals than forward propagation alone. Thus, Thrun's method will detect valid rules that Gallant's algorithm is not able to confirm. The ability to incorporate arbitrary linear constraints into the extraction process means that the method can be used to test rules that specify very general conditions on the output units. For example, it can extract rules that describe when one output unit has a greater activation than all of the other output units. Linear constraints can also be applied to input units. Although the VIA approach is better at detecting general rules than Gallant's algorithm, it may fail to confirm maximally general rules, because it also assumes that the hidden units in a layer act independently. Thrun uses two methods to explore a space of rules. He uses a breadth-first method to search for rules with discrete-valued features. For tasks with real-valued features, he uses a method that starts with training examples as seeds of rules. Each of these initial rules describes a point in the instance space. For example, one initial rule might be the following:

$x_1 = 0.5 \wedge \neg x_2 = 0.2 \wedge \neg x_3 \rightarrow y\ = 0.8$

The rules are iteratively generalized by converting one of the rules' literals into an interval, or by increasing the bounds of an interval in an existing rule. For example, one possible generalization of the rule above is:

$x_1 \in [0.3, 0.5] \wedge \neg x_2 = 0.2 \wedge \neg x_3 \rightarrow y\ = 0.8$

However, Thrun has not described a method for exploring a rule space in domains that involve both discrete and real-valued features.

In [17] for instance, all possible combinations of the input vector $\mathbf{i}$ of N are taken into account in the process of rule generation. In this way, the method must consider 2^n different input vectors, where n is the number of neurons in the input layer of N. Some pedagogical approaches reduce the input vector space by extracting rules for the learning set only, excluding the network's generalization. At the moment, only pedagogical or global approach to rule extraction can guarantee that the knowledge extracted is equivalent to the network, i.e., that the extraction is sound and complete.

The rule-extraction methods discussed in the previous section extract rules that describe the behaviour of the output units in terms of the input units. Another approach to the rule-extraction problem is to decompose the network into a collection of networks, and then to extract a set of rules describing each of the constituent networks.

The nature of these methods is quite different depending upon what type of transfer function the units in the network employ. Craven [48] discusses methods for networks that use sigmoidal transfer function and those that use *local basis functions* [21, 46] instead of sigmoids. Local basis functions are so named because they are designed to respond to localized patterns in their input space. There are a number of local rule-extraction methods for networks that use sigmoidal transfer function for their hidden and output units. In these methods, the assumption is made that the hidden and output units can be *approximated* by threshold functions, and thus each unit can be described by a binary variable indicating whether it is "on"(activation $\approx$ 1) or "off" (activation $\approx$ 0). Given this assumption, we can extract a set of rules to describe each individual hidden and output unit in terms of the units that have weighted connections to it. The rules for each unit can then be combined into a single rule set that describes the network as a whole. If the activations of the input and hidden units in a network are limited to the interval [0,1], then the local approach can significantly simplify the rule search space. The key fact that simplifies the search combinatorics in this case is that the relationship between any input to a unit and its output is a monotonic one. That is, we can look at the sign of the weight connecting the i-th input to the unit of interest to determine how this variable influences the activation of the unit. If the sign is positive, then we know that this input can only push the unit's activation towards 1, it cannot push it away from 1. Likewise, if the sign of the weight is negative, then the input can only push the unit's activation away from 1. Thus, if we are extracting rules to explain when the unit has an activation of 1, we need to consider $\neg x_i$ literals only for those inputs x_i that can push the activation of the output unit away from 1. Similarly, we need to consider the non-negated literals x_i only for those inputs that can push the activation of output unit towards 1. When a search space is limited to including either x_i or $\neg x_i$, but not both, the number of rules in the space is 2^n for a task with n binary features. When this condition of monotonicity does not hold, the size of the rule space is 3^n.

5 Evaluating the quality of Rules

Some of the measures to use, for evaluating the quality of rule-extraction scheme are enumerated below:

1. the expressive power of the extracted rules;

2. the "translucency" of the view taken within the rule extraction technique of the underlying ANN;

3. the quality of the extracted rules;

4. the algorithmic complexity of the rule extraction/refinement technique;

5. the extent to which the underlying ANN incorporates specialized training regimes.

The first classification item refers to the symbolic knowledge presented to the user from the extraction process. In general the best form of expression of the rule is an if-then-else construct; however Craven [48] has used a *decision tree* to represent the rules generated from the network.

The second classification item contains two basic categories: decompositional and pedagogical as explained above and a third labelled eclectic - which essentially combines elements of the two basic categories. In the decompositional the extraction occurs at the level of individual, hidden and output, units within the trained network. In the pedagogical, the neural network is viewed as a "black box",and the extraction is done by mapping inputs directly into outputs.

The third classification item intends to measure how well the task of extracting the rules has been performed, considering the accuracy, consistency and comprehensibility of the set of rules.

The fourth item refers to the requirement for the algorithm to be as effective as possible, in this sense a crucial issue in developing an extraction algorithm is how to constrain the size of search space.

The last point encapsulates the notion of generality in the sense that the rule extraction method adopted should be able to generate the rules out of any neural network. [42] suggests that there should be no architectural constraints irrespective of the training algorithm used, or the way network has been built and how the weights and biases have been learned.

6 Trends in Hybrid Learning

6.1 KBANN

Towell and Shavlik developed a framework for hybrid learning - Knowledge Based Artificial Neural Network (KBANN). It is one of the most widely known framework in the field of hybrid learning. This framework encapsulates the central theme of hybrid machine learning. The basic KBANN system [45] proposed by Towell and Shavlik comprises of two distinctly independent algorithms.

- Rules to Network Module
- Refiner Module

The flow-chart of theory refinement is shown in the following figure. However they later developed an extension of KBANN by building the Rule

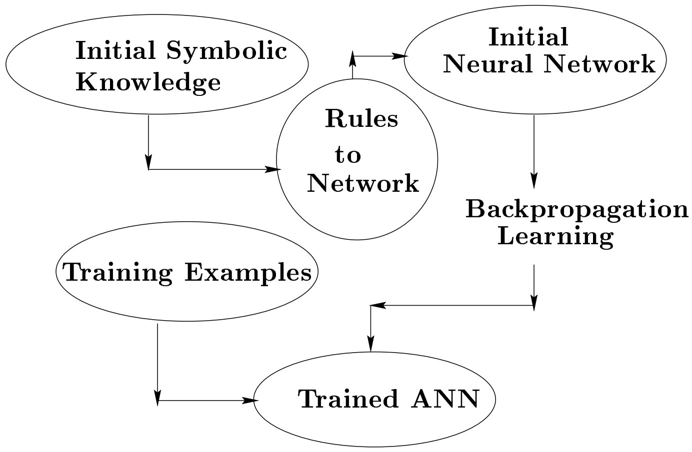


Figure 2: Flow Chart of Theory Refinement by KBANN

Extraction Module on top of the basic two modules which I would like to refer as XKBANN (extended KBANN), thereby specifying their '*Subset*' algorithm and the famous 'MofN' algorithm. The complete algorithm flow of XKBANN can be seen in Figure 3.

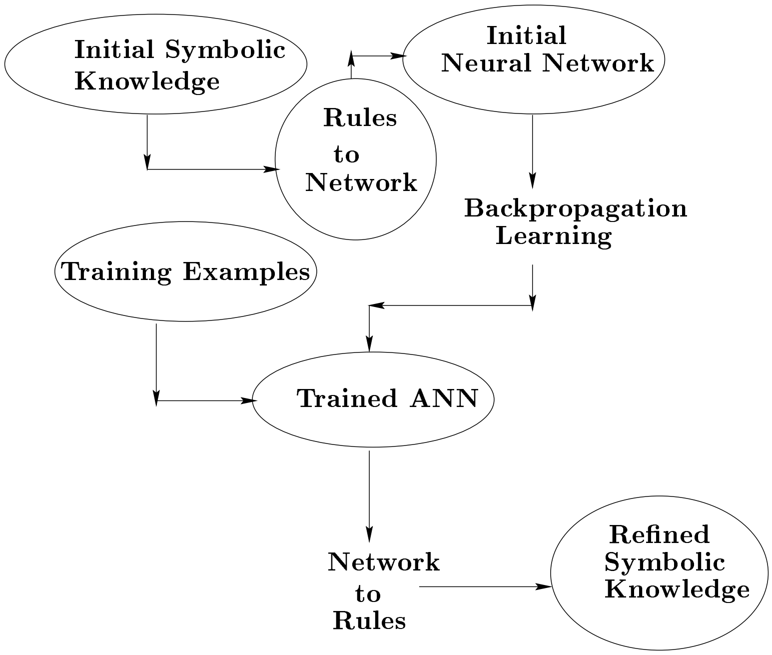


Figure 3: The Extended Flow Chart of KBANN, now with the refined rule extraction

6.1.1 Rules to Network Module

The rules-to-network translation is accomplished by establishing a mapping between a rule set and a neural network. This mapping specified in Table 1, defines the topology of networks created by KBANN as well as the initial link weights of the network. By defining some networks this way, some of the problems inherent to neural networks and empirical (inductive) learning are ameliorated. The translation specifies the features that are probably relevant to making a correct decision. This specification of features addresses problems such as spurious correlations, irrelevant features, and the unboundedness of the set of possible features. Rule translation can specify important "derived features", thereby simplifying the learning problem. Moreover, these derived features can capture contextual dependencies in an example's description. In addition, the rules can refer to arbitrarily small regions of feature space. Hence, the rules can reduce the need for the empirical portion of a hybrid system to learn about uncommon cases. This procedure also indirectly addresses many problems of hand-built classifiers. For instance, the problem of intractable domain theories is reduced because approximately correct theories are often quite brief.

Table 1. Rules-to-Network algorithm

1. Rewrite rules so that disjuncts are expressed as a set of rules that each have only one antecedent.
2. Directly map the rule structure into a neural network.
3. Label units in the KBANN-net according to their "level".
4. Add hidden units to the network at user-specified levels (optional).
5. Add units for known input features that are not referenced in the rules.
6. Add links not specified by translation between all units in topologically-contiguous levels.
7. Perturb the network by adding near-zero random numbers to all link weights and biases.

Rules to be translated into KBANN-nets are expressed as Horn clauses. There are two constraints on the rule set. First, the rules must be *propositional.* This constraint results from the use of neural learning algorithms which are, at present, unable to handle predicate calculus variables. Second, the rules must be acyclic. This "no cycles" constraint simplifies the training of the resulting networks. However, it does not represent a fundamental limitation on KBANN, as there exist algorithms based upon backpropagation that can be used to train networks with cycles. In addition to these constraints, the rule sets provided to KBANN are usually hierarchically structured. That is, rules do not commonly map directly from inputs to outputs. Rather, at least some of the rules provide intermediate conclusions that describe useful conjunctions of the input features. These intermediate conclusions may be used by other rules to either determine the final conclusion or other intermediate conclusions. It is the hierarchical structure of a set of rules that creates derived features for use by the example-based learning system. Hence if the domain knowledge is not hierarchically structured, then the networks created by KBANN will have no derived features that indicate contextual dependencies or other useful conjunctions within example descriptions. Also, the KBANN-net that results from translating a rule set with no intermediate conclusions would have no hidden units. As a result it would be capable of only perceptron-like learning.

6.1.2 Refiner Module

KBANN refines its networks using backpropagation, the standard neural learning algorithm. The KBANN-nets use the cross-entropy function described in section 3.1.2 rather than the standard error function. Using cross entropy function has yielded improvements in both training time and generalization for KBANN-nets. In addition to changing the definition of error, the error function was augmented with a term that penalizes the network for making changes to the original domain theory by adding to the cross entropy function in Equation 4, the regularization term in the following equation:

$$regularizer = \lambda \Sigma_{i \in \omega} \frac{(\omega_i - \omega_{init_i})^2}{1 + (\omega_i - \omega_{init_i})^2} \tag{8}$$

The λ term in the above equation controls the tradeoff between the ability of the network to learn the training set and distance from the initial rules. The principal effect of the addition of this regularization term is to increase the interpretability of trained KBANN-nets by encouraging networks to leave their original weights unchanged.

6.1.3 Rule Extraction Module

As can be seen in Figure 3 with the knowledge of the domain theory already included in the initial neural network (Rules to Network Module) the actual task as it turns out is simpler to achieve. The final refined ANN which Towell and Shavlik prefer to call KNN (Knowledge Based Neural Networks) goes as an input to the rule-extraction module and the output is a set of if-then-else rules which is precisely the refined symbolic knowledge. Before the actual algorithm of rule extraction is explained, some key precursors need to be examined.

There are two important assumptions on which the main algorithm is based:

1. Units are either maximally active (i.e., have activation near one) or inactive (i.e., have activation near zero). However this assumption is not very restrictive as the logistic activation function can be slightly modified to ensure that units approximate step functions. By making this assumption however, each non-input unit in trained KNN can be interpreted as a step function or a Boolean rule. Therefore the rule extraction problem reduces to determining the situations in which the rule is true.

2. The second assumption is that training does not significantly alter the meaning of units. By making this assumptions, the methods are able to attach labels to extracted rules that correspond to consequents in the symbolic knowledge upon which a KNN is based, thereby enhancing the comprehensibility of rules. However [41] suggests that this assumption may not be generally true.

The implication of the first assumption leads to a simplified search by ensuring that the links carry a signal equal to their weight or no signal at all. As a result Equation 2 is simplified to the following equation.

$$net(i) = \sum_j W_{ij} - \theta_i \tag{9}$$

where a minus sign has been placed now instead of a plus to indicate the semantic notion of the threshold. This reduces rule extraction to a search of each unit for sets of incoming links whose summed weights guarantees that the unit's bias is exceeded, regardless of the activation carried by other incoming links.

The consequence of second assumption is that it guarantees that non-input units always have non-negative activations. Since the input units always have activations in [0,1], therefore negatively weighted links can only give rise to non-negated antecedents, while positively weighted links can only give rise to positive antecedents. This has a considerable restriction on the size of the search space [29].

There is an important facet of this rule extraction scheme which is that the extracted rules may make errors of omission and commission with respect to the network. These errors generally result from the cumulative effects of low-weight links that do not appear in extracted rules. Thus one of the chief problems - to minimize such errors are identified with this rule extraction scheme.

The approach of the XKBANN is common to some of the approaches suggested by [24, 29], in that the intention is to find combinations of the input values to a unit that result in it having an activation near one.

6.1.4 Subset Algorithm

Building on the approach proposed by Saito and Nakano in 1988, Shavlik and Towell have proposed an algorithm which searches explicitly for subsets of incoming weights that exceed the bias on a unit. For this reason it has been coined as a *Subset* algorithm. A simple breadth-first search subset algorithm starts by determining whether any sets containing a single link are sufficient to guarantee that the bias is exceeded. If yes, then these sets are rewritten as rules. The search proceeds by increasing the size of the subsets until all possible subsets have been explored. Finally the subsumed rules are removed. The major problem with subset algorithms is that the cost of finding all subsets grows as the size of the power set of the links to each unit. Hence this approach can only be expected to exactly reproduce the behaviour of simple networks working on small problems. To avoid this prohibitive combinatorics, all implementations of subset algorithms use heuristics. Establishing an a priori ceiling on the number of antecedents has several shortcomings. For instance, a ceiling that for one domain, sets an acceptable tradeoff between the number of extracted rules and the difference between the extracted rules and the network may not be acceptable for another domain. A more serious problem occurs when properties of the problem being studied suggest that some rules require large number of antecedents. Therefore, rather than setting a bound on the number of antecedents, this algorithm implementation uses a branch-and-bound method that it is limited in terms of the number of rules it may find.

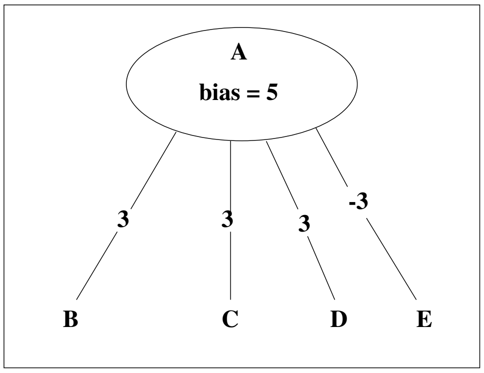
The SUBSET algorithm is enumerated in Table 2. The algorithm iterates over every non-input unit in a trained KNN. At each such unit, it uses a branch-and-bound algorithm over only the positive weighted links to find up to β_p 'positive' subsets whose summed weights exceed the unit's bias. Then, for each positive subset, it makes a new branch-and-bound search over the negatively weighted links to find minimal 'negative' subsets whose summed weight is greater in magnitude than the *difference* between the summed weights of positive subset and the unit's bias.

*T*able 2. SUBSET algorithm

With each hidden and output unit:

1. Extract upto β_p subsets of the positively weighted incoming links whose summed weight is greater than the bias of the unit.
2. With each subset $\mathcal{P}$ of the β_p subsets found in step 1:
   (a) Extract upto β_n minimal subsets of negatively weighted links whose summed weights is greater than the sum of $\mathcal{P}$ less the bias on the unit.
   (b) Let $\mathcal{Z}$ be a new predicate used nowhere else.
   (c) With each subset $\mathcal{N}$ of the β_n subsets found in step 2(a), form the rule:
   "if $\mathcal{N}$ then $\mathcal{Z}$".
   (d) Form the rule:
   "if $\mathcal{P}$ and **not** $\mathcal{Z}$ then <name of unit>".

Finally, the algorithm forms rules that result in the consequent being satisfied when all the antecedents of the positive subset are true and none of the negative subsets associated with that positive subset contain antecedents that are all true. That is, each of the negative subsets are sufficient to prevent the node from being active. This algorithm may discover up to β_p * $(1 + \beta_n)$ rules for each non-input unit in the network to be translated. There is a tradeoff between the number of extracted rules and accurate reproduction of the network's behaviour. Hence the values for β_p and β_n are set to find reasonably-sized set of rules that approximate the network from which they came. The settings which Jude and Shavlik used in their experimental domains resulted in the finding of about 300 rules.



(a)

if B, C, and not(E) then A.

if B, D, and not(E) then A.

if C, D, and not(E) then A.

if B, C, D then A.

(b)

Figure 4: Rule Extraction using the SUBSET algorithm.

Interestingly though the SUBSET algorithm extracts a large set of rules, it is much lesser than many hand-crafted systems (e.g., [20]). Hence SUBSET delivers sets of rules that are potentially tractable. However these rules tend to hide significant structures in trained networks. For instance in Figure 4, part(a) the links to B, C and D all have the same weight, while the link to E has the negative of that weight. Looking at the problem in this way suggests that the rules in Figure 4 could be rewritten with the following rule, which provides a much clearer statement of the conditions on A:

`If 3 of {B, C, D, not(E)} then A.`

Jude and Shavlik indicate that this kind of recognition in the structure among several conjunctive rules led to the development of the MOFN algorithm.

6.1.5 MofN Algorithm

Table 3. The MofN approach to Rule Extraction

1. **clustering** : With each hidden and output unit, form groups of similarly weighted links.
2. **averaging** : Set link weights of all group members to the average of the group.
3. **eliminating**: Eliminating any groups that do not significantly affect whether the unit will be active or inactive.
4. **optimizing** : Holding all link weights constant, optimize biases of all hidden and output units using the backpropagation algorithm.
5. **extracting** : Form a single rule for each hidden and output unit. The rule consists of a threshold given by the bias and weighted antecedents specified by the remaining links.
6. **simplifying**: Where possible, simplify rules to eliminate superfluous weights and thresholds.

The MofN algorithm addresses, both the combinatorial and presentation problem inherent to subset algorithms. It differs from subset algorithms in that it explicitly searches for rules of the form:

`If (` M `of the following` N `antecedents are true) then ...`

As stated above, the motivation for MofN algorithm was discovered when it was noticed that the rules generated by Subset often contain M-of-N style concepts. Further support for this method came from experiments that indicated neural networks are good at learning M-of-N concepts [14], as well as experiments with a variant of ID3 showed a bias towards M-of-N style concepts [31]. Purely conjunctive rules result if $N = M$, while a set of disjunctive rules result when $M = 1$; hence using M-of-N rules does not restrict generality.

The central idea is that individual antecedents (links) do not have a unique importance. Rather, groups of antecedents form equivalence classes in which each antecedent has the same importance as, and is interchangeable with, other members of the class. This equivalence class idea is the key to the MofN algorithm; it allows the algorithm to consider groups of links without worrying about the particular links within the group.

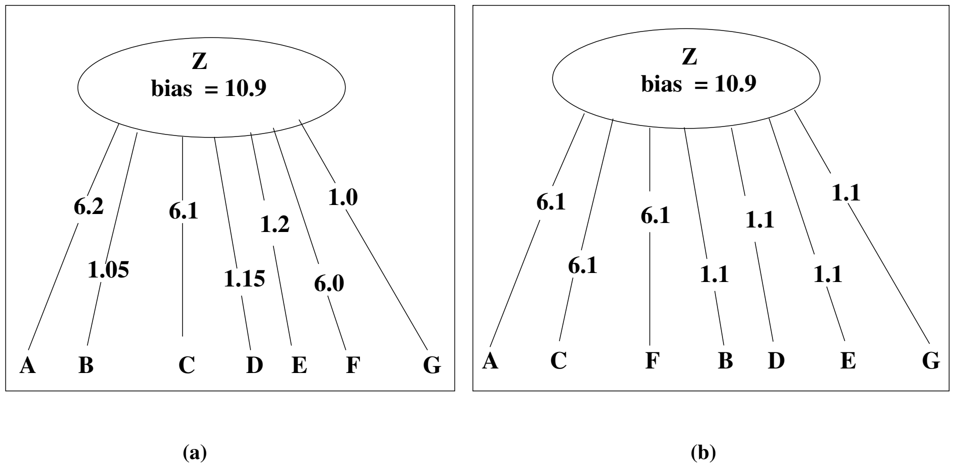


Figure 5: (a) Initial Unit (b) After Step 1 and Step 2

Figure 5 and 6 illustrate the process through which a single unit with seven incoming links is transformed by the MofN procedure into a rule that requires two of three antecedents to be true. The first two steps group the links into two clusters, one of the four links with weight 1.1 and one of the three links with weight 6.1. The third step algorithmically eliminates the cluster with weight 1.1 because there is no situation in which these links can affect the unit. The fourth step, bias optimization, is unnecessary for this example; the averaging and elimination procedures did not significantly change the properties of the unit. The fifth and sixth steps are simple because there is only a single cluster remaining.

6.2 Partial Order Approach

The Rule Extraction problem has been defined in earlier sections, however the notion which is precisely used in developing a solution by the Partial Order Approach is the following:

Let $\mathbf{I}$ be the set of input vectors and $\mathbf{O}$ be the set of output vectors. We define a binary relation ξ on $\mathbf{I}$ x $\mathbf{O}$ such that $\mathbf{o}\xi\mathbf{i} \leftrightarrow \mathbf{o} = \delta(\mathbf{i})$, and

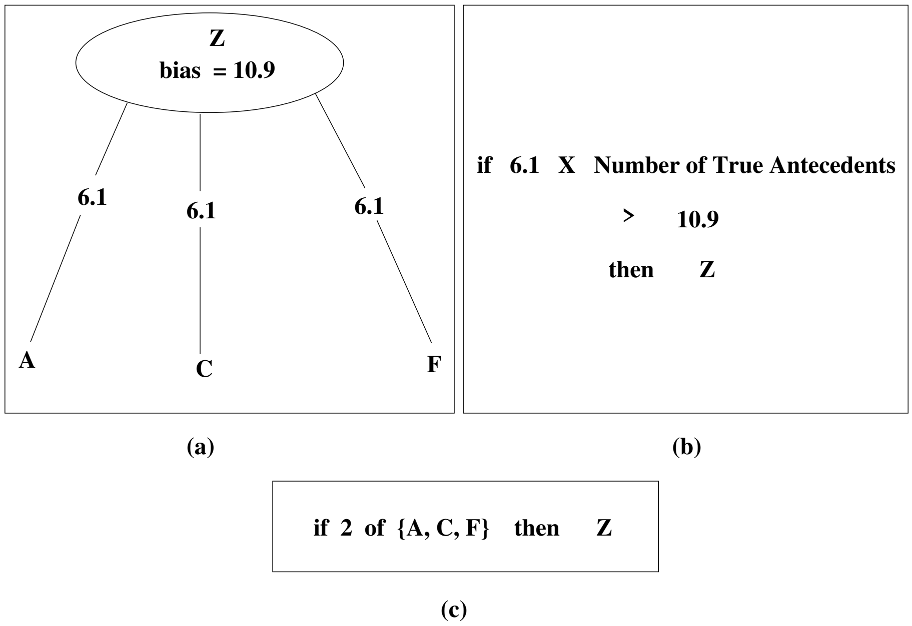


Figure 6: After Step 3

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 $o_j > A_{min}$

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 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.*

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

As in earlier approaches formulated by Jude and Shavlik in the Subset and the MofN algorithms, the version of the rule extraction problem used by [5] concerns itself with the search process, however the search is on *input vector* space, guided by useful simplification and pruning rules defined on the input vector space. In subsequent paragraphs I would explain this approach. But prior to this it is important to see why such an approach was needed at all.

6.2.1 Problems with the previous approaches

Pedagogical approaches are the only ones which ensure that the knowledge extracted is equivalent to the network. However they are not effective when the size of the network increases. So dependence on decompositional methods increases e.g., Subset and MofN methods where the search space is reduced by clustering and pruning weights. There is however no guarantee that the pruned network is equivalent to the original one. So remedy is to re-train the network but with the constraint that only the thresholds change not the weights. With this constraint there is no guarantee that re-training would be successful. More importantly decompositional approaches tend to be *unsound* because the collective effect of the network could be different than the superposition of its parts. To address the above shortcomings, this approach was developed.

6.2.2 Features of this approach

Some of the new features of this approach is enumerated below:

- Partial ordering the input *vector* space and not input *weight* space
- An eclectic approach: *pedagogical* and *decompositional* both
- Define pruning rules to reduce the search space during *pedagogical* approach
- Define simplification rules to reduce the size of extracted rules
- Extraction algorithm is *sound* and *complete* for so called *regular* networks
- Extraction is *sound* for even non-regular networks
- Non-regular networks contain regularities in their subnetworks
- Method for regular network can be applied to a non-regular network but now in a decompositional fashion preserving soundness
- Complexity of extraction algorithm reduced in some cases

- Non-Monotonicity is captured by using the default negation

In the following sequel some basic notions specifically useful for understanding this algorithm are presented and then the solution to the rule extraction problem for regular networks is explained. Thereafter the case for non-regular networks is explained. An example is presented to illustrate how this algorithm works. For a given neural network, the following notations are used to describe it.

- *Input vector:* It is denoted by $\mathbf{i} = (i_1, i_2, ..., i_p)$ where ($i_{j(1 \leq j \leq p)} \in$ [-1,1])
- *Output vector:* It is denoted by $\mathbf{o} = (o_1, o_2, ..., o_q)$ where ($o_{j(1 \leq j \leq q)} \in$ [-1,1])
- *Hidden layer vector:* It is denoted by $\mathbf{n} = (n_1, n_2, ..., n_r)$ where ($n_{j(1 \leq j \leq r)} \in$ [-1,1])
- $\theta_{n_j\ (1 \leq j \leq r)}$: It is the j^{th} hidden neuron threshold ($\theta_{n_j} \in \Re$)
- $\theta_{o_j\ (1 \leq j \leq r)}$: It is the j^{th} output neuron threshold ($\theta_{o_j} \in \Re$)
- - θ_{n_j} (resp. - θ_{o_j}): It is called the *bias* of the j^{th} hidden neuron (resp. output neuron)
- $W^1_{ij(1 \leq i \leq r, 1 \leq j \leq p)}$: It is the weight between the j^{th} *input* neuron and i^{th} *hidden* neuron ($W^1_{ij} \in \Re$)
- $W^2_{ij(1 \leq i \leq q, 1 \leq j \leq r)}$: It is the weight between the j^{th} *hidden* neuron and i^{th} *output* neuron ($W^2_{ij} \in \Re$)
- The activation function used in the example later is given by the equation below

  $$h(x) = \frac{2}{1 + e^{-\beta x}} - 1$$

  However any crescent shaped function could be used as well, like a hyperbolic tangent function.
- For each output $o_j(1 \leq j \leq q)$ in $\mathbf{o}$ we have
  $o_j = h(\Sigma_{i=1}^{r}(W^2_{ji} \cdot h(\Sigma_{k=1}^{p}(W^1_{ik} \cdot i_k) - \theta_{n_i})) - \theta_{o_j})$

- The desirable constraint to be satisfied on the output unit j is:
  j is true iff $W^2{}_{j1}n_1 + W^2{}_{j2}n_2 + ... + W^2{}_{jr}n_r > h^{-1}(A_{min}) + \theta_{o_j}$

  Positive Neural Network: A neural network where all weights $W_{ij} \in \Re^+$

  Regular Neural Network: A single layer neural network is said to be regular if its connections from the *hidden* layer to each *output* neuron have either all positive or all negative weights, and if the transformation algorithm (follows later) generates on it a network without complementary literals in the input layer.

Following conventions are assumed. 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 a set of boolean values namely true, false. Intuitively, an interpretation maps each literal to either *true* or *false*. Given a neural network, each input and output neuron is associated with a unique literal in $\mathbf{P}$. If $\mathcal{I}$ denotes a set of input neurons and $\mathcal{O}$ be the set of output neurons, then each input vector $\mathbf{i}$ can be seen as an interpretation. The input vector $\mathbf{i}$ is represented 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 a boolean true, and the value -1 means an assignment of boolean false.
Example: Suppose $\mathcal{I} = \{p, q, r\}$[1], if $\mathbf{i}$ = (1,-1,1), then it represents the set $\{p, r\}$, if $\mathbf{i}$ = (-1,-1,-1) it represents the empty set ϕ.

Hence the set of input vectors $\mathbf{I}$ is an abstract representation of the power set of the input neurons $\mathcal{I}$. It is written as $\mathbf{I} = \wp(\mathcal{I})$.

6.2.3 Some Useful Notions

- 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:* $\mathbf{I} \times \mathbf{I} \rightarrow \aleph$)

- The function *dist* is a metric on $\mathbf{I}$

[1] If a linear ordering is fixed on the symbols of $\mathcal{I}$, it can be represented as a list, say [p,q,r].

- 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 $\langle \mathbf{i}_m \rangle = \Sigma_{j=1}^{p} \mathbf{i}_m(i_j)$ $(\langle\rangle : \mathbf{I} \rightarrow \mathbf{Z})$

Definition 3 Let $\mathbf{i}_m$ and $\mathbf{i}_n$ be input vectors in $\mathbf{I}$. $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_n$ iff $\mathbf{i}_m \subseteq \mathbf{i}_n$ where $\subseteq$ is the usual subset ordering on $\mathbf{i}$.

Definition 4 δ is defined as a mapping from the set of input vectors to the set of output vectors.

Example: $\delta(\mathbf{i_m}) = \mathbf{o_m}(o_j)$, where $\mathbf{o_m}(o_j)$ is a notation which gives the value of the j^{th} output unit in the output vector $\mathbf{o_m}$.

Proposition 1 The partially ordered set [I, $\leq_{\mathbf{I}}$] is a distributive lattice.

$\leq_{\mathbf{I}}$ *is actually the the ordering of interest w.r.t the network's output and is a useful concept for classifying the input vectors for a special class of networks as is evident from the following proposition*

Proposition 2 If $W_{ji} \in \Re_{+}$ then $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_n$ implies $\mathbf{o}_m(o_j) \leq \mathbf{o}_n(o_j)$, for all $1 \leq j \leq q$, where $\mathbf{o}_m(o_j) = \delta(\mathbf{i}_m)$ and $\mathbf{o}_n(o_j) = \delta(\mathbf{i}_n)$

It is evident from the experience of the neural network community that the neural networks will not always be positive, and hence the Partial Order Approach transcends to the case of regular networks where there could be negative weights from the input layer to the hidden layer or hidden to the output layer. It is found that a particular kind of *transformation* process from any *regular* to a *positive* network would help in the search process. It is possible by a transformation algorithm enumerated below that any regular network be changed to an equivalent positive form, and then the usual subset ordering on the input vectors of the positive form can be used to curtail the search process, and once a set of rules for the positive form are obtained, the mapping σ (to be defined subsequently) can be applied to get the rules for the original regular network. The Knowledge Extraction Algorithm for obtaining the rules from regular networks would be explained in the next section.

Definition 5 Let $\mathbf{I}$ be the set of input vectors of s tuples. Given $\mathcal{I}_{+}$ and an abstract representation $\mathbf{I}_{+}$ of $\wp(\mathcal{I}_{+})$, each element $x_i \in \wp(\mathcal{I}_{+})$, $1 \leq i \leq s$, is mapped to the set {-1,1} such that:

$\sigma_{[x_1,...,x_s]}(i_1, ..., i_s) = (i_1{}', ..., i_s{}')$, where

$i_i{}' = i_i$ if x_i is a positive literal and

$i_i{}' = -i_i$ if x_i is a negative literal.

Example: $\sigma_{[a,\sim b,c,\sim d]}(1, 1, -1, -1) = (1, -1, -1, 1)$

6.2.4 The Transformation Algorithm

1. If y is linked to the n_j^{th} neuron in 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 neurons in the hidden layer through positive and negative connections, and if the negative connection is to the n_j^{th} neuron:

   (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.

Figure 7 illustrates the application of Transformation Algorithm. Interestingly, it follows from the transformation, that querying the network N_+ with $\mathbf{i}$ = (1,1,1) in Figure 8, is equivalent to querying the network N with $\mathbf{i}$ = (1,1,-1), and so on. Proposition 3 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.* If the network is already positive, then σ is an identity function. In the partial order approach taken in [5] the positive form is used for querying.

Proposition 3 If a network is regular with input $\mathcal{I}_+$, then $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_n$ implies $(\mathbf{o}_m(\mathbf{o_j}) = \delta(\sigma_{[\mathcal{I}_+]}(\mathbf{i}_m))) \leq (\mathbf{o}_n(\mathbf{o_j}) = \delta(\sigma_{[\mathcal{I}_+]}(\mathbf{i}_n)))$ for all $1 \leq j \leq q$, for q output neurons in the given neural network.

6.2.5 Pruning and Simplification Rules

Pruning Rule 1: Let $\mathbf{i}_m$ and $\mathbf{i}_n$ be input vectors of a regular neural network N, such that dist $(\mathbf{i}_m, \mathbf{i}_n) = \mathbf{1}$ and $\langle \mathbf{i}_m \rangle < \langle \mathbf{i}_n \rangle$.

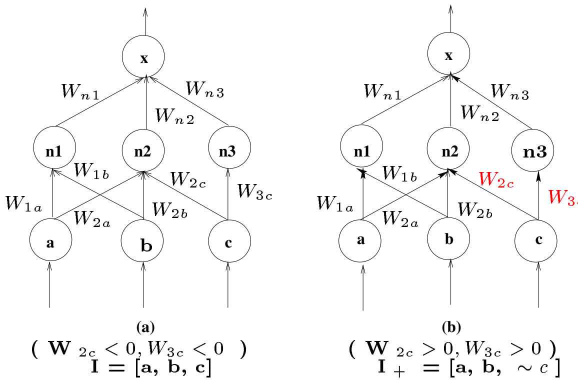


Figure 7: (a) Original Neural Network and (b) Equivalent Positive Network

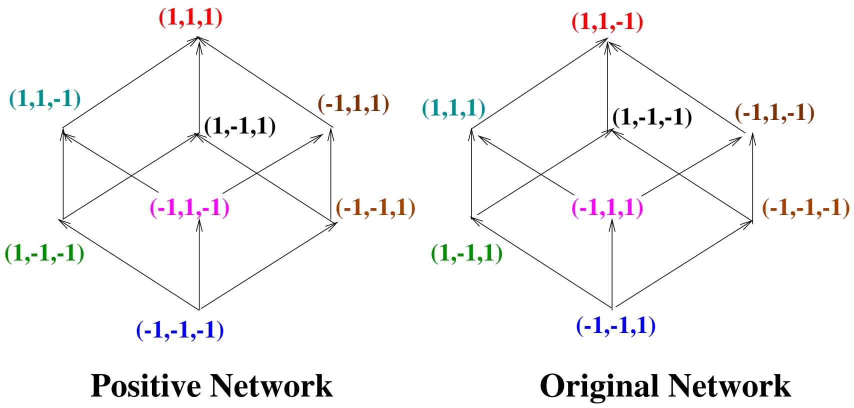


Figure 8: Lattices for (a) Equivalent Positive Network and (b) Original Neural Network

If $\mathbf{i}_n$ does not satisfy the constraint $\mathbf{C}_{o_j}$ [2] on the j^{th} output neuron of N, then $\mathbf{i}_m$ does not satisfy $\mathbf{C}_{o_j}$ either.

Pruning Rule 2: Let $\mathbf{i}_m$ and $\mathbf{i}_n$ be input vectors of a regular neural network N, such that dist $(\mathbf{i}_m, \mathbf{i}_n) = \mathbf{1}$ and $\langle \mathbf{i}_m \rangle < \langle \mathbf{i}_n \rangle$.

[2] The desirable constraint to be satisfied on the output unit j is: *j is true iff* $W^2{}_{j1}n_1 + W^2{}_{j2}n_2 + ... + W^2{}_{jr}n_r > h^{-1}(A_{min}) + \ \theta_{o_j}$

If $\mathbf{i}_m$ satisfies the constraint $\mathbf{C}_{o_j}$ on the j^{th} output neuron of N, then $\mathbf{i}_n$ also satisfies $\mathbf{C}_{o_j}$ either.

Subsumption: A rule r_1 subsumes a rule r_2 iff r_1 and r_2 have the same conclusion and the set of premises of r_1 is a subset of the set of premises of r_2
Example $a \to x$ subsumes $ab \to x$ and $a \sim b \to x$

Complementary Literals: Let $r_1 = L_1, ..., L_i, ..., L_j \to L_{j+1}$ and $r_2 = L_1, ..., \sim L_i, ..., L_j \to L_{j+1}$ be derived rules, where j $\leq |\mathcal{I}|$. Then, $r_3 = L_1, ..., L_{i-1}, L_{i+1}, ..., L_j \to L_{j+1}$ is also a derived rule. r_3 subsumes r_1 and r_2
Example If $\mathcal{I}$ = {a,b,c} and there be rules $a \sim bc \to x$ and $a \sim b \sim c \to x$, then after simplification the following is obtained $a \sim b \to x$.

Fact: If a literal L_{j+1} holds in the presence of any combination of the truth values of literals $L_1, ..., L_j$ in $\mathcal{I}$ then a rule of the form $\to L_{j+1}$ is derived (L_{j+1} is a fact).
Example Considering the ordering on **I**, an output neuron x is a fact iff inf (**I**) satisfies the constraint on x. By the Pruning Rule 2, if inf (**I**) satisfies x then any other input vector in **I** satisfies x as well.

M of N: Let m, n $\in \aleph$, $\mathcal{I}' \subseteq \mathcal{I}$, $|\mathcal{I}'|$ = n, m $\leq$ n. Then if any combination of m elements chosen from $\mathcal{I}'$ implies L_{j+1} we derive a rule of the form m $(\mathcal{I}') \to L_{j+1}$.
Example Consider the following set of literals {a,b,c} of input neurons, and say one derives the following rules: $ab \to x, ac \to x, bc \to x$. Then by the above M of N simplification rule one can write $2\,(abc) \to x$. [5] gives a more detailed insight into this.

M of N Subsumption: Let m,p $\in \aleph$, $\mathcal{I}' \subseteq \mathcal{I}$. m $(\mathcal{I}') \to L_{j+1}$ subsumes p $(\mathcal{I}') \to L_{j+1}$ iff m $<$ p. [5] explains this in more depth.

Remark The above definitions have been defined on regular networks, keeping all positive weights in mind from the hidden to the output layer, however regular networks could also have all negative weights from the hidden layer to the output layer, and it is easy to see that the constraint $\mathbf{C}_{o_j}$ on the j^{th} output neuron of a regular network with all negative weights from hidden layer to output layer is given by: $W^2{}_{j1}n_1 + W^2{}_{j2}n_2 + ... + W^2{}_{jr}n_r < h^{-1}(A_{min}) + \theta_{o_j}$. In other words the only difference is on the sign ($<$) of the constraint, and one should simply invert the signs at Pruning Rule 1 and 2, rest of the rules are valid as it is.

6.2.6 Knowledge Extraction from Regular Networks

1. Apply the *Transformation Algorithm* over N, obtaining its positive form N_+;

2. Find $inf(\mathbf{I})$ and $sup(\mathbf{I})$ w.r.t N_+ using σ;

3. For each neuron o_j in the output layer of N_+ do:

   (a) Query N_+ with input vector $inf(\mathbf{I})$. If $o_j > A_{min}$, apply the simplification rule *Fact* and stop.

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

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

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

      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 new $\mathbf{i}_\perp$;

      iv. Apply the simplification rule *Complementary Literals*, and add the rules derived accordingly to the rule set.

      v. Set new $\mathbf{i}_\top :=$ old $\mathbf{i}_\perp$ 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 new $\mathbf{i}_\top$;

      viii. Apply the simplification rule *M of N*, and add the rules derived accordingly to the rule set.

   (e) Apply the simplification rules subsumption and *M of N* subsumption on the rule set regarding o_j.

Example

- Given the original neural network shown in Figure 7(a), applying the first step of transformation algorithm gives an equivalent positive neural network shown in the figure 7(b) .

- We have the $inf(\mathbf{I})$ and $sup(\mathbf{I})$ of the positive network known, since the usual subset ordering is applied on the positive network.

- By making use of the mapping σ we get the equivalent vertices for the lattice of the original network. This can be used to get the actual rules in the last step.

- The procedure is started by checking if $inf(\mathbf{I})$ activates the output neuron x , i.e., $inf(\mathbf{I})$ satisfies the constraint C_{o_x} , if it does then by pruning rule 2 we know that any other input vector also activates x and by the $Fact$ simplification rule we extract $\to x$ and stop.

- If however, $inf(\mathbf{I})$ does not activate x , then we may need to query the network with the immediate successors of $inf(\mathbf{I})$. This is the new $\mathbf{i}_\perp$ in our algorithm. Here let us call them $\mathbf{I}^*$, where $dist(inf(\mathbf{I}), \mathbf{I}^*)$ = 1.

- We then proceed to check the element $sup(\mathbf{I})$. If $sup(\mathbf{I})$ does not satisfy C_{o_x} , then by pruning rule 1 we stop, extracting no rules with conclusion x . If $sup(\mathbf{I})$ activates x, we conclude $abc \to x$, but still have to check the input vectors $\mathbf{I}^{**}$ - the immediate predecessors of $sup(\mathbf{I})$. Later some simplification on $abc \to x$ could be applied if at least one of the input vectors in $\mathbf{I}^{**}$ activates x . Meanwhile we keep abc in stand by and proceed.

- We just choose to start by checking $\mathbf{i_1}$ = (-1,-1,1) in $\mathbf{I}^*$. If $\mathbf{i_1}$ does not satisfy C_{o_x} , then remaining inputs in $\mathbf{I}^*$ are checked.

- However if $\mathbf{i_1}$ activates x then, again by pruning rule 2, we know that (-1,1,1) and (1,-1,1) also do.

- If all the elements in $\mathbf{I}^*$ activate x then we can use M of N subsumption rule to derive the following rule
  1(abc) $\to x$

- 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 pruning rule 1.

- If, however $\mathbf{i_2}$ activates x, we can derive $ab \rightarrow x$, using pruning rule 2 and complementary literal simplification rule. *Note: Here the rule $abc \rightarrow x$ is simplified to $ab \rightarrow x$.*

- 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 Subsumption rule and M of N rule.

- 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 in $\mathbf{I}^*$ and therefore we can stop.

- *Stopping Criterion:* Either all elements in the ordering are visited or, if not, for each element not visited, pruning rules 1 and 2 guarantee that it is safe not to consider it, in the sense that it is either already represented in the set of rules or irrelevant and will not give rise to any new rule.

6.2.7 Extraction Algorithm for Non-Regular Networks

Non-regular networks contain regularities in their subnetworks. Method for regular network can be applied to a non-regular network but now in a decompositional fashion preserving soundness. Intuitively, it suggests that one can 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. With this motivation a subnetwork is defined as:

Definition 6 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' 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' 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 $\theta_{o'_k} = \theta_{o_k}$.

It is however difficult to find an ordering on the set of input vectors without having to actually check each input. So gain obtained in terms of complexity could be lost. It is also very much likely that the subnetworks produce unsound rules, may be 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. The problem is due to the non-discrete nature of the network's hidden neurons. So an important assumption which is made is that *the hidden neurons present possible activations in the range* $\{-1, A_{max}\}$ *or* $\{A_{min}, 1\}$. *where* $A_{max} \in [-1, 0]$. Apart from this assumption it was planned to break the original neural network into so called *basic neural structures* and then assemble the rules obtained from each of these structures. This motivates the development of the concept of a basic neural structure.

Definition 7 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.

A network N with r hidden neurons and q output neurons contains r + q BNSs. There are basically two kinds of BNS that a neural network could be split into:

- *Input to Hidden BNS:* A BNS containing no output neurons of N.
- *Hidden to Output BNS:* A BNS containing no input neurons of N.

Remark Every BNS is regular.

The development of the Knowledge Extraction Algorithm for non-regular networks follows in the same way as for the regular networks, however this time the information about weights is also used to curtail the search space, this is formulated in the form of two more pruning rules besides the ones already mentioned above for the positive networks. Pruning Rules 1 and 2 for Positive Networks could be directly applied to a BNS once it has been changed to its positive form by the application of the Transformation Algorithm. This follows directly from the following proposition:

Proposition 4 If a network is regular with input $\mathcal{I}_+$, then $\mathbf{i}_m \leq_{\mathbf{I}} \mathbf{i}_n$ implies $(\mathbf{o}_m(o_j)) \leq (\mathbf{o}_n(o_j))$ for all $1 \leq j \leq q$, for q output neurons in the given neural network.

> **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, both $\mathbf{i}_m$ and $\mathbf{i}_n$ are immediate predecessors of $\mathbf{i}_o$. Let $\mathbf{i}_m$ be obtained from $\mathbf{i}_o$ by flipping the i^{th} input from 1 (resp. A_{min} for *Hidden to Output BNSs*) to -1, while $\mathbf{i}_n$ is obtained from $\mathbf{i}_o$ by flipping the k^{th} input from 1 (resp. A_{min} for *Hidden to Output BNSs*) to -1.
If $|W_{jk}| \leq |W_{ji}|$ then $o_j\,(\,\mathbf{i}_m\,) \leq o_j\,(\,\mathbf{i}_n\,)$ and this formally means: $\mathbf{i}_m \leq_{\langle\rangle} \mathbf{i}_n$

Example Figure 9 part(a) shows the input to hidden BNS with the input vector $\mathbf{i}_o$, and the output o_j when the values i_i and i_k are 1 each, in the input vector $\mathbf{i}_o$, and the weights $|W_{jk}| = 3$ and $|W_{ji}| = 5$ ($|W_{jk}| \leq |W_{ji}|$). Part(b) shows the case when the i^{th} value i_i is flipped to -1, and the part(c) shows the case when k^{th} value i_k is flipped to -1. It is clear from this figure for example if the transfer function is taken as a step function for the sake of simplicity, that the output of the j^{th} unit o_j , because of vector $\mathbf{i}_n$ (part(c)) is greater than for the input vector $\mathbf{i}_m$ (part(b)).

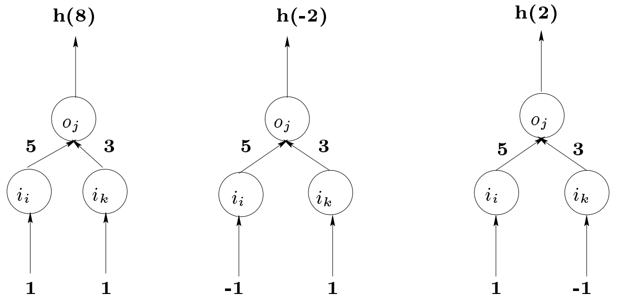


Figure 9: The *Input to Hidden BNS* (a) with the input vector $\mathbf{i}_o$ (b) with the input vector $\mathbf{i}_m$ and (c) with the input vector $\mathbf{i}_n$

Pruning Rule 4: Let $\mathbf{i}_m$, $\mathbf{i}_n$ and $\mathbf{i}_o$ be three different input vectors in $\mathbf{I}$ such that $dist\,(\,\mathbf{i}_m\,,\mathbf{i}_o\,) = 1$, $dist\,(\,\mathbf{i}_n\,,\ \mathbf{i}_o\,) = 1$ and $\langle \mathbf{i}_o \rangle < \langle \mathbf{i}_m \rangle$, $\langle \mathbf{i}_n \rangle$ that is, both $\mathbf{i}_m$ and $\mathbf{i}_n$ are immediate successors of $\mathbf{i}_o$. Let $\mathbf{i}_m$ be obtained from $\mathbf{i}_o$ by flipping the i-th input from -1 to 1 (resp. A_{min} for *Hidden to Output BNSs*), while $\mathbf{i}_n$ is obtained from $\mathbf{i}_o$ by flipping the k^{th} input from -1 to 1. (resp. A_{min} for *Hidden to*

Output BNSs).
If $|W_{jk}| \leq |W_{ji}|$ then $o_j(\mathbf{i}_n) \leq o_j(\mathbf{i}_m)$ and this formally means: $\mathbf{i}_n \leq_{\langle\rangle} \mathbf{i}_m$

Example Figure 10, part(a) shows the input to hidden BNS with the input vector $\mathbf{i}_o$, and the output o_j when the values i_i and i_k are -1 each, in the input vector $\mathbf{i}_o$, and the weights $|W_{jk}| = 3$ and $|W_{ji}| = 5$ ($|W_{jk}| \leq |W_{ji}|$). Part(b) shows the case when the i^{th} value i_i is flipped to 1, and the part(c) shows the case when k^{th} value i_k, is flipped to 1. It is clear from this figure for example if the transfer function is taken as a step function again for the sake of simplicity, that the output of the j^{th} unit o_j, because of vector $\mathbf{i}_n$ (part(c)) is lesser than for the input vector $\mathbf{i}_m$ (part(b)).

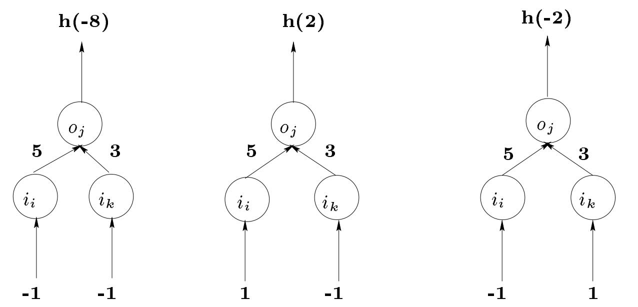


Figure 10: The *Input to Hidden BNS* (a) with the input vector $\mathbf{i}_o$ (b) with the input vector $\mathbf{i}_m$ and (c) with the input vector $\mathbf{i}_n$

Consider the following neural network in Figure 11 part(a), and its equivalent positive form in part(b), if the information coming from the weights is incorporated by application of pruning rules 3 and 4, we get the lattice shown in Figure 12 for the output n_1, corresponding to the *Input to Hidden BNS*. For output x in Figure 13, corresponding to the *Hidden to Output BNS*. Now that we have the information coming from two different orderings superimposed upon one another, a new ordering encapsulating the concept of the previous orderings is defined as:
Definition 8 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 $\mathbf{i}_m \leq_{\langle\rangle} \mathbf{i}_n$.

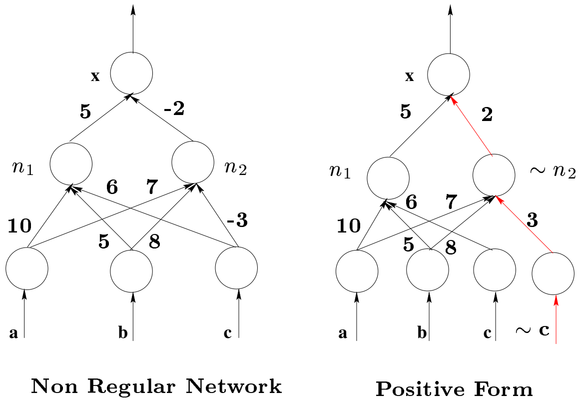


Figure 11: (a) A non - regular network, and its (b) equivalent positive form

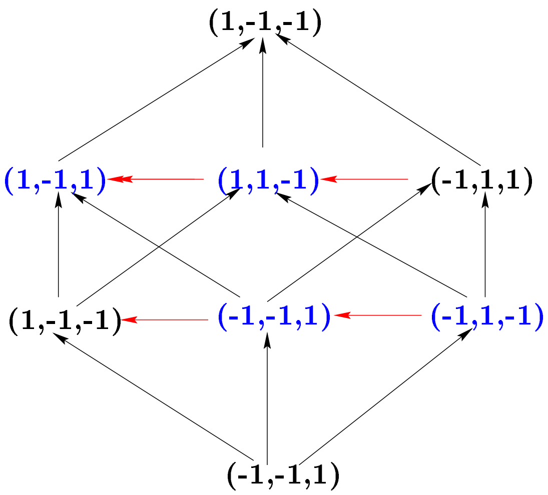


Figure 12: After adding information about weights for the *Input to Hidden BNS*, for the output n_1

$\preceq$ provides a systematic way of searching the input vectors space. 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

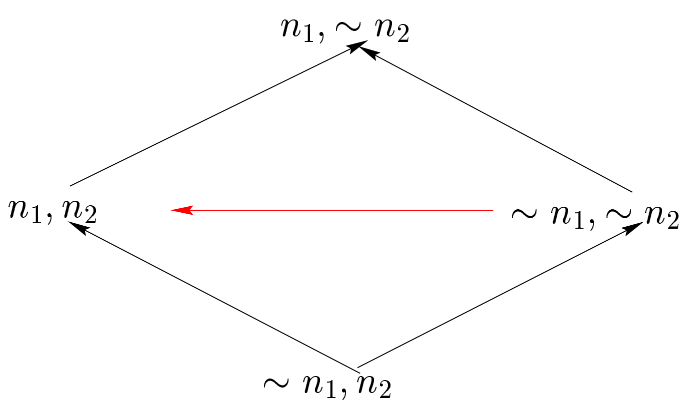


Figure 13: After adding information about weights for the *Hidden to Output BNS*, for the output x

left of inputs with smaller weights. Consider an *Input to Hidden BNS* shown in Figure 14, where $\mathbf{i}_\top = (1,1,1,1) = \{a, b, \sim c, \sim d\}$. Then it should be

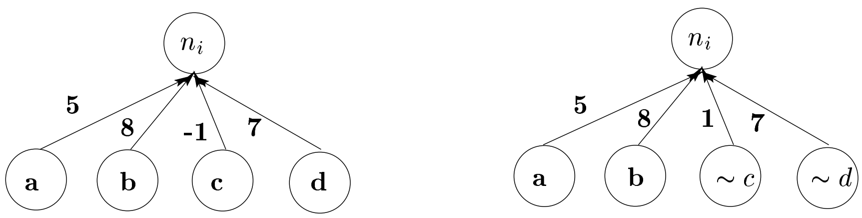


Figure 14: (a) An *Input to Hidden BNS* and (b) its positive form, $\mathbf{i}_\top = (1,1,1,1)$

rearranged to $[b, \sim d, a, \sim c]$ (written in the list form) and it is equal to (1,1,1,1). 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, only the inputs on the left most -1 input are flipped. In this way, repetition of input vectors is avoided. Similarly starting from the minimum element, we rearrange $\{\sim a, \sim b, c, d\}$ and obtain (-1,-1,-1,-1) $= [\sim b, d, \sim a, c]$. Now at a distance 2 from $inf(\mathbf{I})$ and beyond, we only flip the inputs on the left of the left most 1 input. For a more comprehensive treatment of this, one should refer to [5]. Here I would proceed to explain the case now for the *Hidden to Output BNSs*. As stated above the problem with the *Hidden to Output BNS* lies in the fact that the hidden neurons do not present discrete activations in the interval {-1,1}, rather it is understood that they are active if the activation values lie on the interval { A_{min} ,1} , and non-active if the activation is in the range of {-1, A_{max} }. Hence the next step is to find a suitable ordering on the hidden neurons ranges of activations (-1, A_{max}) and (A_{min} ,1). However it is not easy to define such an ordering easily. Its not the case that we can say that ($n_1 < A_{max}$) *and (* $n_2 < A_{max}$ *)* $\preceq$ *(* $n_1 < A_{max}$ *)* and ($n_2 > A_{min}$). At this stage [5] emphasizes that a compromise needs to be made in order to preserve soundness. Roughly one should analyze the activation values of the hidden neurons in the "worst cases". These activations are given by -1 and A_{min} in the case of a hidden neuron connected by a positive weight to an output neuron and A_{max} and 1 in the case of a hidden neuron connected through a negative weight to an output neuron.

Remark When the activation values of hidden neurons are either positive in the interval (A_{min} ,1) or negative in the interval (-1, A_{max}), it is assumed without loss of generality that the network's learning algorithm is such that no hidden neurons present activations in the range [A_{max}, A_{min}]. Even one can assume that $A_{max} = A_{min} \approx 0$

Figure 15 shows a *Hidden to Output BNS* and the corresponding set inclusion ordering when both the weights from the hidden neuron to the output neuron are positive. Figure 16 shows orderings on *Hidden to Output BNS* with two input neurons n_1 and n_2 , and when the weights are positive and negative, using the worst case analysis.

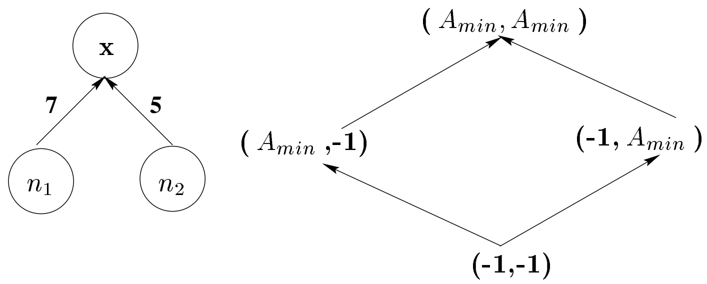


Figure 15: A *Hidden to Output BNS* and the corresponding set inclusion ordering

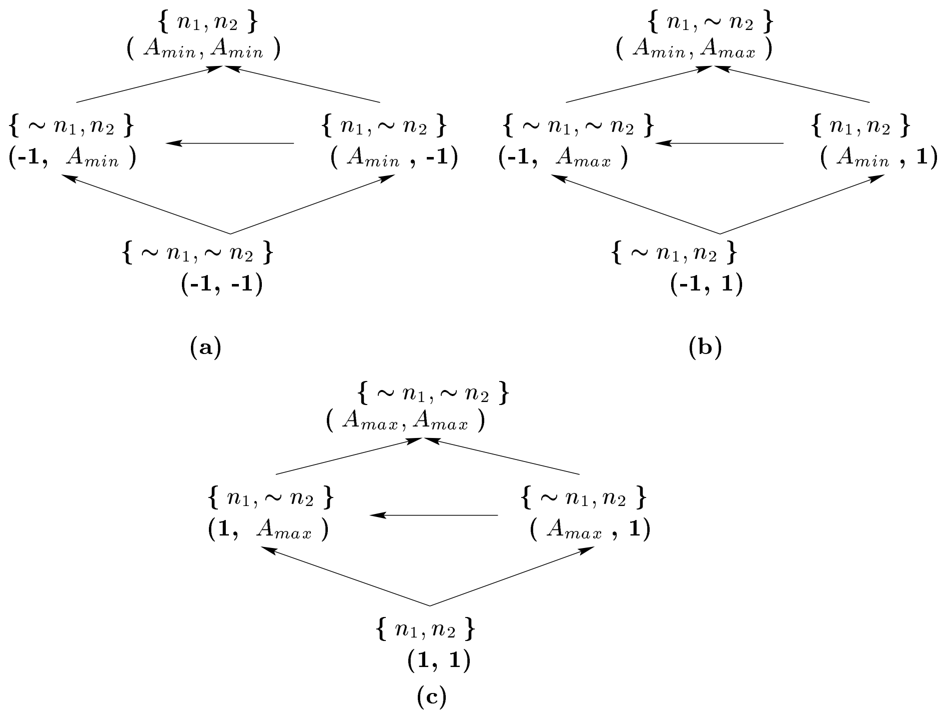


Figure 16: Orderings on *Hidden to Output BNS* (a) sup($\mathbf{I}$) $= (\mathbf{n_1}, \mathbf{n_2})$ (b) sup($\mathbf{I}$) $= (\mathbf{n_1}, \sim \mathbf{n_2})$ (c) sup($\mathbf{I}$) $= (\sim \mathbf{n_1}, \sim \mathbf{n_2})$

The mapping σ for Regular Networks and *Input to Hidden BNS* can be used for *Hidden to Output BNS* as well but with the following assumption that A_{max} = - A_{min} . Hence $\sigma_{[a,\sim b,c,\sim d]}$ (A_{min}, A_{min} ,-1,-1) = (A_{min}, A_{max} ,-1,1). Now in the sequel below, the rule extraction algorithm for non-regular networks is presented.

6.2.8 Knowledge Extraction Algorithm - General Case

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

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

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

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

/* *Recall that, now the Search Space Pruning Rules 1 and 2, have to be replaced by BNS Pruning Rules 1 and 2 respectively.*

/* *The following lines also need to be added to the extraction algorithm for regular networks*

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

3. Assemble the final Rule Set of N .

The hidden neurons do not have concepts directly associated to them, hence 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 serves to do this task.
Lemma 1 The extraction of rules from Input to Hidden BNSs is sound and complete. [5] opines that the above Lemma can be used for completion of rules extracted from *Input to Hidden* BNSs to assemble the network's rule set, i.e., it allows an extracted rule of the form $X_1, ..., X_p \rightarrow L_j$ to be substituted by the stronger $X_1, ..., X_p \leftrightarrow L_j$.

Example Assume that the extraction algorithm derives $a \rightarrow n_1$ from BNS $\mathcal{B}_1$ and $b \sim c \rightarrow n_2$ from BNS $\mathcal{B}_2$. By Lemma 1, one can have $a \leftrightarrow n_1$ and $b \sim c \leftrightarrow n_2$. By contraposition, we have $\sim a \leftrightarrow \sim n_1$ from $\mathcal{B}_1$, and $\sim b \vee c \leftrightarrow \sim n_2$ from $\mathcal{B}_2$. Having this information around , assume that one has the derived rule $n_1 \sim n_2 \rightarrow x$ from *Hidden to Output BNS* $\mathcal{B}_3$. As a result one assembles the final rule set w.r.t the output x as: $\{ a \sim b \rightarrow x, \, ac \rightarrow x \}$.

6.2.9 Some useful results

Lemma 2 *The extraction of rules from Hidden to Output BNSs is sound.*
Theorem 1 *The extraction algorithm for non-regular networks is sound.*
Theorem 2 *The extraction algorithm for non-regular networks is incomplete.*

7 Experimental Results

In this section I have tried to sketch a comparitive evaluation of the three algorithms that have been discussed above. The basis of this evaluation is the set of experiments carried out by [13] and [5]. Some of the problem domains which are used for benchmarking the performance of the algorithms is the MONKS problem and the DNA sequence analysis problem. DNA sequence analysis problem is an example of a real world problem where the rule extraction techniques have been tested, whereas the MONKS problem is the artificial domain for testing the performance. I am going to draw on the results obtained from the real world case study of the DNA sequence analysis and compare the following parameters for all the extraction techniques discussed so far. The parameters which have been monitored during the testing were (a) quality (b) comprehensibility and (c) complexity. Quality is determined by the accuracy of the rules and their fidelity to the network from which they were extracted. Comprehensibility reflects the readability of the rule sets both as a whole set of rules and as individual rules. Complexity of course refers here to the time complexity of the algorithm.

I am not going to discuss the experiments in detail here, as this is already available in the individual papers [13, 5]. Rather I am going to present the comparitive performance of these algorithms when tested on real world problems such as the DNA Sequence Analysis. Here a short desciption of the DNA sequence problem is explained.

A DNA molecule contains two strands that are linear sequences of nucleotides. The DNA is composed from four different nucleotides - *adenine*, *guanine*, *thymine* and *cytosine* - which are abbreviated by a, g, t, c respectively. Some sequences of the DNA strand, called genes, serve as a blue print for the synthesis of proteins. Interspersed among the genes are segments, called non-coding regions, that do not encode proteins. A special notation is used to identify the location of nucleotides in a DNA sequence.

Each nucleotide is numbered with respect to a fixed, biologically meaningful, reference point. For example, "@3*atcg*" states the location relative to the reference point in the DNA, followed by the sequence of symbols that must occur, i.e., an 'a' must appear three nucleotides to the right of the reference point and so on. By convention, location zero is not used, and '*' indicates that any nucleotide would suffice in a particular location. Each location is encoded in the network by four input neurons, representing nucleotides a, g, t, c in this order. Suppose that input vectors @ -1g = 1, @1c = 1 and @5t = 1 activate the output promoter. Then a desirable extracted rule would be of the form:

$$@ - 1gct * ** \rightarrow Promoter\,.$$

The extraction algorithms have been used for both the eukaryotes[3] promoter recognition and prokaryotes[4] splice junction determination. Promoters are short DNA sequences that precede the beginning of genes. The aim of "promoter recognition" is to identify the starting location of genes in long sequences of DNA. The network's input layer for this task contains 228 neurons (57 consecutive DNA nucleotides), its single hidden layer contains 16 neurons, and its output neuron is responsible for classifying the DNA sequence as promoter or non-promoter. The training examples used by [13] contain 53 sample promoters and 53 non-promoter sequences. The training examples used by [5] consist of 48 promoter and 48 non-promoter DNA sequences, while the test set contained 10 examples.

The second application is eukaryotic splice-junction determination. Splice junctions are points on a DNA sequence at which the non-coding regions are removed during the process of protein synthesis. The aim of "splice-junction" determination is to recognize the boundaries between the part of the DNA retained after splice - called exons and the part that is spliced out - called introns. The task consists therefore of recognizing exon/intron (E/I) boundaries and intron/exon (I/E) boundaries. Each example is a DNA sequence with 60 nucleotides (240 input neurons), where the center is the reference point. The network contains 26 neurons in its single hidden layer, while two output neurons are responsible for classifying the DNA sequences into E/I or I/E. The third category (neither E/I nor I/E) is considered true when neither output neurons are active. [13] report that they used the data set comprising of 3190 examples, of which approximately 25% are exon/intron boundaries,

[3]Eukaryotic cells contain a nucleus

[4]Prokaryotes are single-celled organisms without a nucleus, e.g. E. Coli

25% are intron/exon boundaries and the remaining 50% are neither, further they used randomly selected 1000 examples from the 3190 examples available. [5] uses a test set of 100 examples. Towell in [13] uses cross-validation to test the network and the accuracy of the set of rules. [5] suggests in his report that the results obtained for the promoter problem do not have statistical significance due to the reduced number of examples available for testing. However the accuracy of the set of rules w.r.t the network's training set was 90.6%, this was similar to that obtained for the test set. Figures 17 and Figure 18 illustrate the comparative accuracy and fidelity of the algorithms.

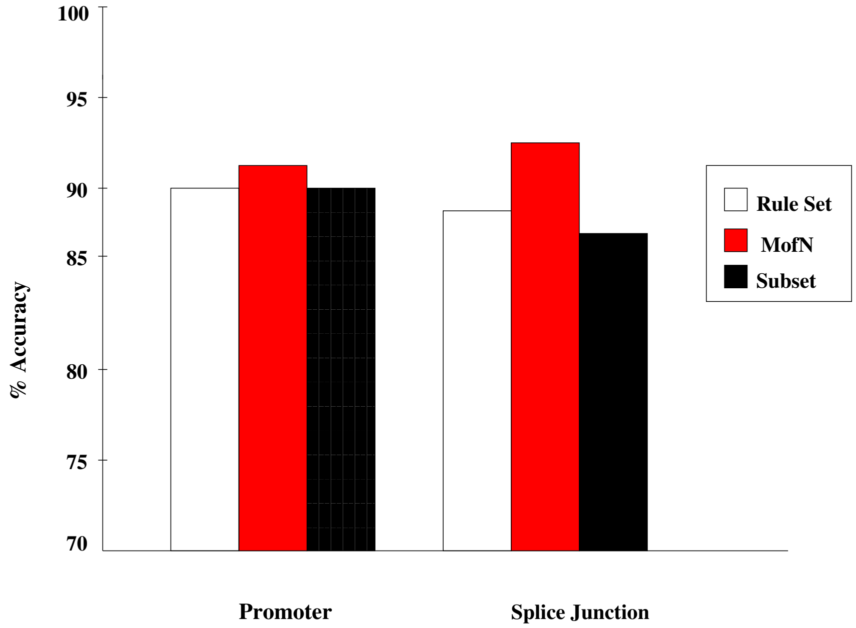


Figure 17: Comparison of accuracies obtained by the various rule extraction techniques in a typical real world example

Comprehensibilty is still an issue which remains largely the focus of research in rule extraction, and considering just the size of the rule set that is finally extracted, it could be mind boggling to read and understand all the rules. Secondly, consider a rule set small enough to be read and understood, but do the structure of individual rules allow an easy understanding? The two different perspectives of understanding the rule set gives the intuition

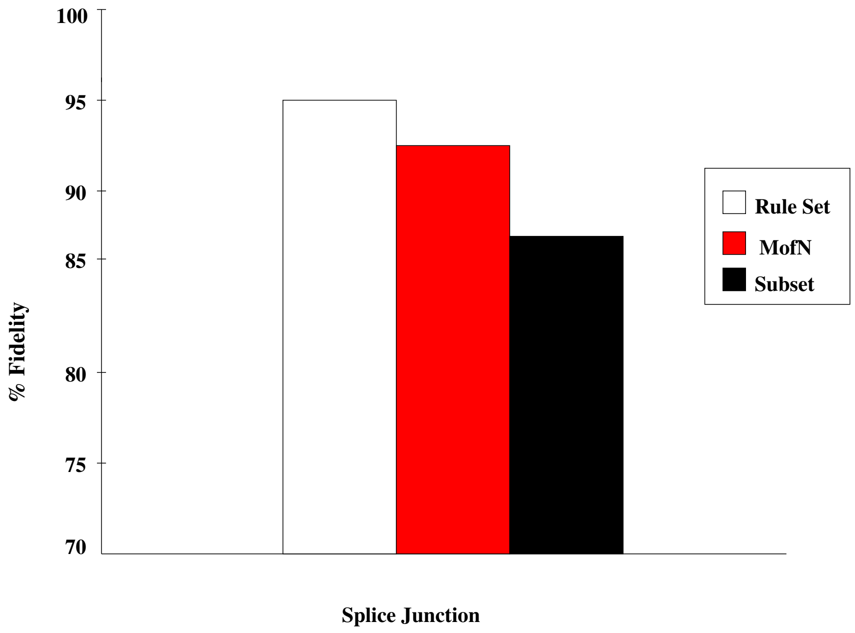


Figure 18: Comparison of fidelity achieved by the various rule extraction techniques in the Splice Junction determination problem

of global comprehensibility and local comprehensibility. Infact none of the above extraction schemes tend to fair quite well on this, however depending upon the simplification rules applied, one could have an optimized level of comprehensibility. In [5], though the extracted rule set is provably sound, the size of the rule set before simplification is just too large to be understood for any meaningful purpose. In the Subset and the MofN algorithms, the former produces 300 rules only in the promoter and the splice junction problem, however the accuracy is only 86%, however the latter in the promoter recognition problem Figure 19 part(a) has larger number of antecedents per rule as compared to the former, although in part(b) the Subset has larger number of antecedents than MofN though the difference is very less, however the number of negative antecedents per rule is larger for MofN than for Subset. As compared to the Partial Order approach the MofN seems to be better in terms of comprehensibility, as regards global and local perspectives. But as [5] indicate that the strength of their rule extraction scheme

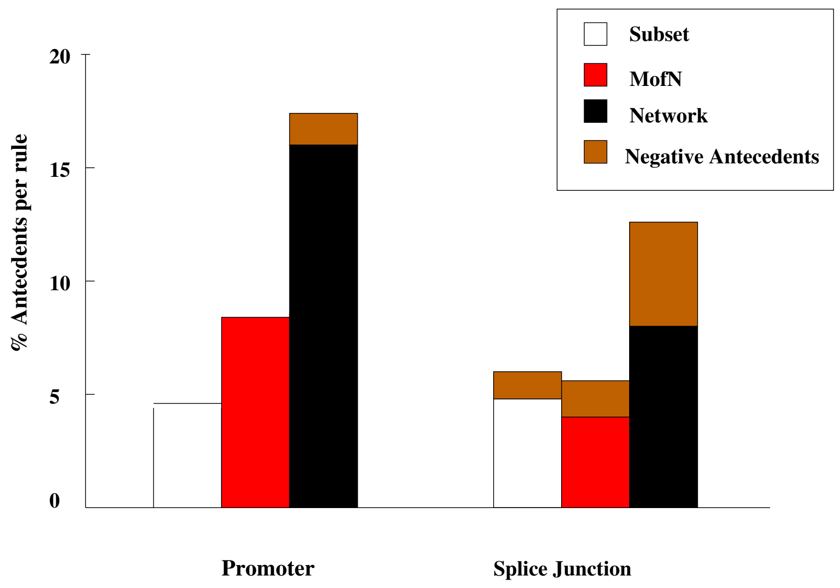


Figure 19: Number of rule antecedents per rule (a) Promoter recognition (b) Splice Junction determination

lies in the incorporation of default negation in the extracted rule set and that the rules are sound, it seems that the Partial Order approach needs to work more on the efficient simplification and pruning schemes in order to reduce the overall size of the rule set. Infact in both the above applications due to the intractability of the set of input vectors (2^{228} and 2^{240} elements each), the authors indicate that they had to limit the maximum number of rules to 50,000 per hidden neuron. They also mention that they had to speed up the search process by doing a jump as in a binary search, from the ordering's minimum element to a new minimal element in the frontier at which input vectors start to generate rules. The authors state that the M-of-N simplification rules were not implemented and they could be very powerful in reducing the size of the initial rule set. Complexity is the next important issue, the complexity of MofN and even the Partial Order approach could not be precisely determined because the bias-optimization phase uses backpropagation. Training some simple classes of neural networks has been proven to be NP-complete [38]. Hinton suggests that, in practice, backpropagation usually runs in $\mathcal{O}((u \times l)^3)$ where u is the number of units and l is the average number of links received by a unit. In the MofN algorithm, the initial clustering, requires $\mathcal{O}((u \times l)^2)$ time, and cluster elimination requires $\mathcal{O}(n \times u \times l)$ time, where n is the number of training examples. Each of the other steps use $\mathcal{O}(u \times l)$ time. On the other hand the complexity in [5] depends mostly on the search strategy which can be facilitated by efficient search mechanisms such as by imposing efficient orderings guided by heuristics.

8 Related Work

In this report I sketched the general hybrid learning framework as propounded by Towell and Shavlik and thereby examined some of the well known rule extraction systems, some of the key features that each offers and the limitations that each have. But my survey has not dealt with some of the other extraction techniques like TREPAN, a rule extraction algorithm given by Mark Craven, who deals with issues like generality and scalability of rule-extraction algorithms. Briefly, TREPAN is an approach to rule extraction where the problem of extracting a comprehensible hypothesis form a trained network is viewed as an inductive learning task. In this learning task, the target concept is the function represented by the network, and the hypothesis produced by the learning algorithm is a decision tree that approximates the network. Unlike most other inductive learning problems, in this task access to an *oracle* is provided to answer *queries* during the learning process. The advantage with the oracle is that the learner can make as many queries as desired, and it can make queries for those instances that provide most information about the target function. He claims in [48] that TREPAN is general enough that it can be applied to a wide variety of learned models and not just to neural networks.

Thrun's Validity Interval Analysis is also an approach to rule extraction. Thrun in [43] developed a method called *validity interval analysis* (VIA). VIA tests rules by propagating activation intervals through a network after constraining some of the input and output units. Thrun frames the problem of determining *validity intervals* (i.e., valid activation ranges for each unit) as a linear programming problem. This is an important insight because it allows activation intervals to be propagated *backward*, as well as forward through the network, and it allows arbitrary linear constraints to be incorporated into the computation of validity intervals. Backward propagation of activation intervals enables the calculation of tighter validity intervals than forward propagation alone.

An exhaustive survey covering some decompositional approaches is presented in [2].

9 Conclusion

As a conclusion of this survey, it needs to be mentioned that there is still a plethora of research going on in this area and several groups are focusing on diverse issues like generality, scalability, comprehensibility, complexity, that there should be no constraints on the architecture of the network, that there should be little constraints on the training of the network and so on. Some of the known limitations that the MofN and Subset algorithms have is that they need the network to be knowledge based, and MofN for example introduces shifts in the meanings of units as a result of training, which could make comprehensibility difficult, domain theories may not always provide a sufficiently rich vocabulary to allow a knowledge based neural network to accurately learn a concept. Some of the drawbacks the Partial Order approach seem to have is that the size of the rule sets is quite large thereby offering poor comprehensibility. A possible extension to this extraction algorithm could be the extraction of meta-level priorities directly from the network's *Hidden to Output BNSs*. Negative weights from the hidden to output neurons implement a preference relation. This could be used to extract directly a set of meta-level priorities together with the object level rules from the network. Alternatively this could be done after the extraction, when the rules are assembled to derive the final rule set. This would lead to enhancement of the rule set readability and compactness. Besides this the search process itself could be improved, by adding some new heuristics, and exploring new orderings on input vectors.

References

[1] Srinivasan A., Muggleton S., King R. D, and Sternberg M. J. E. Mutagenesis: ILP experiments in a non-determinate biological domain. In *Proceedings of the Fourth Inductive Logic Programming Workshop*, 1994.

[2] 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*, 8(6), 1995.

[3] Dolsak B. and Muggleton S. The application of inductive logic programming to finite element mesh design. In *In S. Muggleton (Ed.), Inductive Logic Programming*. London:Academic Press, 1992.

[4] A. E. Bryson and Y. C. Ho. Applied optimal control. *Blaisdell*, 1969.

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

[6] Rumelhart D. E., G. E. Hinton, and R. J. Williams. Learning internal representations by error backpropagation. In Rumelhart D. and McClelland J., editors, *In Parallel Distributed Processing: Explorations in the microstructure of cognition*, volume 1. MIT Press, Cambridge, 1986.

[7] Rumelhart D. E., G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 1986a. Reprinted in Anderson and Rosenberg[1988].

[8] Rumelhart D. E., G. E. Hinton, and R. J. Williams. Learning internal representations by error backpropagation. In *In Parallel Distributed Processing*, chapter 8. 1986b.

[9] Rumelhart D. E., Durbin R., Golden R., and Chauvin Y. Backpropagation: The basic theory. In Chauvin Y. and Rumelhart D. E., editors, *Backpropagation: Theory, Architectures, and Applications*. Lawrence Erlbaum, Hillsdale, NJ, 1995.

[10] Buchanan B. G., Smith D. H.and White W. C., Gritter R., Feigenbaum E. A., Lederberg J., and Djerassi C. Applications of artificial intelligence for chemical inference, XXII: Automatic rule formation in mass spectrometry by means of the meta-dendral program. *Journal of the American Chemical Society*, 98, 1976. 6168.

[11] Hinton G. Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 1–12, Amherst, MA, 1986. Erlbaum.

[12] Hinton G. Connectionist learning procedures. *Artificial Intelligence*, 40:185–234, 1989.

[13] Towell G. and Shavlik J. Extracting refined rules from knowledge based neural networks. *Machine Learning*, 13(1):71–101, 1993.

[14] Fisher D. H and McKusick K. B. An empirical comparison of id3 and backpropagation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 788–793. Detroit, MI: Morgan Kaufmann, 1989.

[15] Kramer A. H. and Sangiovanni-Vincentelli A. Efficient parallel learning algorithms for neural networks. In Tourtezky D., editor, *Advances in Neural Information Processing Systems*, volume 1, pages 40–48. Morgan Kaufmann, San Mateo, CA, 1989.

[16] Wolberg W. H., Street W. N., and Mangasarian O. L. Machine learning techniques to diagnose breast cancer from fine needle aspirates. *Cancer Letters*, 77:163–171, 1994.

[17] S. Hölldobler and Y. Kalinke. Toward a new massively parallel computational model for logic programming. In *Workshop on Combining Symbolic and Connectionist Processing, ECAI 94*. 1994.

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

[19] Gallant S. I. *Neural Network Learning and Expert Systems*. MIT Press, 1993.

[20] McDermott J. R1: A rule-based configurer of computer systems. *Artificial Intelligence*, 19, 1982.

[21] Moody J. and Darken C. Learning with localized receptive fields. In Hinton G. E., Sejnowski T. J., and Toretzky D. S, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 133–143. Morgan Kaufmann, 1988.

[22] M. Jabri, S. Pickard, and P. Leong. ANN based classification for heart defibrillators. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems*, pages 637–644, San Mateo, CA, 1992.

[23] Lindsay R. K., Buchanan B. G., Feigenbaum E. A., and Lederberg J. *Applications of Artificial Intelligence for organic chemistry*. New York:McGraw-Hill, 1980.

[24] Saito K. and Nakano R. Medical diagnostic expert system based on pdp model. In *Proceedings of IEEE International Conference on Neural Networks*, volume 1, pages 255–262, 1988.

[25] Hunter L. and Klein T. Finding relevant biomolecular features. In *In Proceedings of the First International Conference on Intelligent Systems for Molecular Biology*, pages 190–197. AAAI Press, Bethesda, MD, 1993.

[26] Mangasarian O. L. and Solodov M. V. Nonlinear complimentarity as unconstrained and constrained minimization. *Mathematical Programming, Series B*, 62:277-297, 1994.

[27] Mangasarian O. L. and Solodov M. V. Serial and parallel backpropagation convergence via nonmonotone perturbed minimization. *Optimization Methods and Software*, 4(2):103-116, 1994.

[28] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. BACKPROPAGATION applied to handwritten zip code recognition. *Neural Computation*, 1(4), 1989.

[29] Fu. L. M. Rule learning by searching on adapted nets. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 590–595, Anaheim, CA: AAAI Press., 1991.

[30] Minsky M. and Papert S. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, 1969.

[31] Murphy P. M. and Pazzani M. J. Id2-of-3: Constructive induction of n-of-m concepts for discriminators in decision trees. In *Proceedings of the Eighth International Machine Learning Workshop*, pages 183–187. Evanston, IL: Morgan Kaufmann, 1991.

[32] Tom M. Mitchell. *Machine Learning.* WCB/McGrawHill, 1997.

[33] Lavrăc N. and Džeroski S. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, 1994.

[34] D. B. Parker. Learning logic. Technical report, Center for Compuatational Research in Economics and Management Science, MIT,MA, 1985.

[35] D. A. Pomerlau. ALVINN: an autonomous land vehicle in a neural network. Technical report, Computer Science Dept., Carnegie Mellon University, Pittsburgh PA, 1989.

[36] Quinlan J. R. Induction of decision trees. *Machine Learning*, 1986.

[37] Becker S. and LeCun Y. Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 Connectionist Models Summer School.* Morgan Kaufmann, 1988.

[38] Judd S. On the complexity of loading shallow neural networks. *Journal of Complexity*, 4:177–192, 1988.

[39] Kedar-Cabelli S. and McCarty T. Explanation-based generalization as resolution theorem proving. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 383–389, San Francisco, 1987. Morgan Kaufmann.

[40] Michalski R. S. A theory and methodology of inductive learning. *Artificial Intelligence*, 20:111–161, 1983.

[41] Sutton R. S. Two problems with backpropagation and other steepest descent learning procedures for networks. In *Program of the Eighth Annual Conference of the Cognitive Science Society*, pages 823–831, Amherst, MA:Erlbaum, 1986.

[42] Thrun S. The extraction of refined rules from knowledge based neural networks. Technical report, Institut für Informatik, Universität Bonn, 1994.

[43] Thrun S. Extracting rules from artificial neural networks with distributed representations. In Tesauro G., Touretzky D., and Leen T., editors, *Advances in Neural Information Processing Systems*, volume 7. MIT Press, Cambridge, MA, 1995.

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

[45] Jude W. Shavlik and Geoffrey G. Towell. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1,2):119–165, 1994.

[46] Pggio T. and Girosi F. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247:978–982, 1990.

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

[48] Craven M. W. *Extracting Comprehensible Models from Trained Neural Networks*. PhD thesis, Computer Sciences Department, University of Wisconsin Madison, WI, 1996.

[49] Craven M. W and Shavlik J. W. Using sampling and queries to extract rules from trained neural networks. In *In Proceedings of the Eleventh International Conference on Machine Learning*, pages 37–45, New Brunswick, N.J, 1994. Morgan Kaufmann.

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

[51] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, 1974.