Originally submitted to *Machine Learning* 8/91
Accepted 11/92

Extracting Refined Rules from Knowledge-Based Neural Networks

Geoffrey G. Towell*
towell@learning.siemens.com

Jude W. Shavlik
shavlik@cs.wisc.edu

University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706
(608) 262-6613

Abstract

Neural networks, despite their empirically-proven abilities, have been little used for the refinement of existing knowledge because this task requires a three-step process. First, knowledge must be inserted into a neural network. Second, the network must be refined. Third, the refined knowledge must be extracted from the network. We have previously described a method for the first step of this process. Standard neural learning techniques can accomplish the second step. In this paper, we propose and empirically evaluate a method for the final, and possibly most difficult, step. Our method efficiently extracts symbolic rules from trained neural networks. The four major results of empirical tests of this method are that the extracted rules: (1) closely reproduce the accuracy of the network from which they are extracted; (2) are superior to the rules produced by methods that directly refine symbolic rules; (3) are superior to those produced by previous techniques for extracting rules from trained neural networks; (4) are "human comprehensible." Thus, this method demonstrates that neural networks can be used to effectively refine symbolic knowledge. Moreover, the rule-extraction technique developed herein contributes to the understanding of how symbolic and connectionist approaches to artificial intelligence can be profitably integrated.

Keywords: theory refinement
integrated learning
representational shift
rule extraction from neural networks

Running Head: Extracting Rules

* Currently at Siemens Corporate Research, 755 College Road East, Princeton, NJ, 08540. Email: towell@learning.siemens.com. Please direct all correspondence to this address.

1 Introduction

Artificial neural networks (ANNs) have proven to be a powerful and general technique for machine learning (Fisher & McKusick, 1989; Shavlik *et al.*, 1991). However, ANNs have several well-known shortcomings; perhaps the most significant of which is that a trained ANN is essentially a "black box." That is, determining exactly why an ANN makes a particular decision is a daunting task. This is a significant shortcoming, for without the ability to produce understandable decisions, it is hard to be confident in the reliability of networks that address real-world problems. Also, the fruits of training neural networks are difficult to transfer to other neural networks (Pratt *et al.* (1991) ameliorate this problem), and all but impossible to directly transfer to non-neural learning systems. Hence, a trained neural network is analogous to Pierre de Fermat's comment about his last theorem. Like Fermat, the network tells you that it is has discovered something "wonderful", but then does not tell you what it discovered.

This paper sheds light into the neural-network black box by combining symbolic, rule-based reasoning with neural learning. Our approach is to form the three-link chain illustrated by Figure 1 in which symbolic knowledge is revised and corrected using neural networks. Thus, the approach we present makes possible the use of neural networks as the empirical learning algorithm underlying a rule-refinement system.

Figure 1 approximately here.

The first link of our three-link chain is to insert knowledge, which need be neither complete nor correct, into a neural network using KBANN (Towell *et al.*, 1990). (Hereafter, networks created using KBANN will be referred to as Knowledge-based Neural Networks – KNNs.) This step changes the representation of the rules from symbolic to neurally-based, thereby making the rules refinable by standard neural learning methods.

The second link of the chain is to train the KNN using a set of classified training examples and a standard neural learning algorithm, backpropagation (Rumelhart *et al.*, 1986) or any other method for weight optimization of feedforward neural networks. In so doing, the rules upon which the KNN are based are corrected so that they are consistent with the training examples.

The final link is to extract rules from trained KNNs. This is an extremely difficult task for arbitrarily-configured networks, but it is somewhat less daunting for KNNs due to properties that stem from their initial comprehensibility. Taking advantage of these properties, we developed a method to efficiently extract intelligible rules from networks. When evaluated on two real-world test problems in terms of the ability to correctly classify examples not seen during training, the method produces sets of rules that closely approximate the networks from which they came. Moreover, the extracted rules are equal, or superior, to the rules resulting from rule-refinement methods that act directly on the rules, rather than their re-representation as a neural network. We also show that our method is superior to a previous algorithm for the extraction of rules from general neural networks (e.g., Saito & Nakano, 1988; Fu, 1991).

The next section contains a brief overview of our method for inserting rules into neural networks. The subsequent section describes both our method, and the best reported method for the extraction of rules from trained KNNs. Sections 4 and 5 present a series of empirical tests of our rule-extraction method. In Section 4 we use two real-world learning problems taken from molecular biology to determine the strengths and weaknesses of our rule-extraction method. In Section 5 we use an artificial domain to characterize, more closely than is possible using real-world domains, the abilities of our rule-extraction method. The final sections of this paper relate our approach to others that extract rules from trained neural networks and discuss our future research plans.

2 The KBANN Rules-to-Network Translator

KBANN translates symbolic knowledge into neural networks, defining the topology and connection weights of the networks it creates. It uses a knowledge base of domain-specific inference rules represented as propositional Horn clauses, to define what is initially known about a topic. Detailed explanation of the procedure used by KBANN to translate rules into a KNN are given by Towell *et al.* (1990) and Towell (1991). We provide a brief summary of this procedure in the following example.

Consider the sample knowledge base in Figure 2a, which defines membership in category A. Figure 2b represents the hierarchical structure of these rules: solid and dotted lines represent necessary and prohibitory dependencies, respectively. Figure 2c represents the KNN that results from translating this knowledge base into a neural network. Units[1] X and Y in Figure 2c are introduced into the KNN to represent the disjunction in the

[1] *Units* are the basic processing elements of neural networks. They receive real-numbered signals from other units over weighted connections (referred to as "links") and mathematically transform these signals into a real-numbered output which is sent on to other units (see Table 1). Generally, units are divided into three categories: *input*, that receive signals from the environment, *output*, that send signals to the environment, and *hidden*, that have no connection to the environment.

rule set. Otherwise, each unit in the KNN corresponds to a consequent or an antecedent in the knowledge base. Thick lines in Figure 2c represent heavily-weighted links in the KNN; they correspond to dependencies in the knowledge base. Thin lines represent the links added to the network to allow additional refinement of the knowledge base.

Figure 2 approximately here.

This example illustrates that using this procedure to initialize KNNs has two principal benefits. First, the algorithm indicates the input features that are believed to be important to an example's classification. Second, it specifies important derived features (e.g., B and C in Figure 2), thereby guiding the choice of the number and connectivity of *hidden units* in the KNN.

3 Rule Extraction

This section presents detailed explanations of both our approach and a previously reported approach for extracting rules from trained KNNs. The first algorithm is presented only for comparison. The second algorithm (introduced in Section 3.4) is the focus of this work.

3.1 Assumptions

The rule extraction methods we present make two assumptions about trained networks. The first assumption, which underlies most methods for extracting rules from trained neural networks, is that the units are either maximally active (i.e., have activation near one) or inactive (i.e., have activation near zero).[2] This assumption is not particularly restrictive because the commonly-used logistic activation function can be slightly modified to ensure that units approximate step functions. By making this assumption, each non-input unit in a trained KNN can be interpreted as a step function or a Boolean rule. Therefore, the rule extraction problem is to determine the situations in which the "rule" is true.

The second assumption is that training does not significantly alter the meaning of units. By making this assumption, the methods are able to attach labels to extracted

[2]Exceptions to this generalization include a method proposed by Bochereau and Bourgine (1990) and most methods for extracting fuzzy rules (e.g., Berenji, 1991).

$$a_i = f\left(\sum_j (w_{i,j} * a_j) - \theta_i\right) \quad (1)$$

$$f(x) = \frac{1}{1 + e^{-sx}} \quad (2)$$

$$a_i = f\left(\sum_{\{j|a_j \approx 1\}} w_{i,j} - \theta_i\right) \quad (3)$$

where	a_i	is the activation of unit i
	$w_{i,j}$	is the weight on a link from unit j to unit i
	θ_i	is the "bias" of unit i.
	s	is a parameter affecting the slope of the sigmoid. (Except when explicitly stated, $s = 1$.)

Table 1: The logistic activation function used by backpropagation.

rules that correspond to consequents in the symbolic knowledge upon which a KNN is based, thereby enhancing the comprehensibility of the rules. Our examination of trained networks indicates that the meanings usually are quite stable, although Sutton (1986) suggests that this assumption may not be generally true.

3.2 Commonalities

The method we introduce for extracting rules from neural networks, as well as those previously described (e.g., Saito & Nakano, 1988; Fu, 1991), do so by trying to find combinations of the input values to a unit that result in it having an activation near one. Hence, in this section we describe the mathematics underlying the neural networks that we use.

Units in backpropagation-trained neural networks normally have activations defined by Equations 1 and 2 in Table 1. To rephrase Equation 1, the activation of a unit is a function of the sum of weighted inputs to a unit less its bias. Broadly speaking, the result of Equation 2 is that if the summed weighted inputs exceed the bias, then the activation of a unit will be near one. Otherwise the activation will be near zero. *Hence, rule-extraction methods search for constraints on the input settings such that the weighted sum is guaranteed to exceed the bias.*

The first of the above assumptions, that all units in trained networks have activations near zero or one, simplifies this search by ensuring that links carry a signal equal to their weight or no signal at all. As a result, Equation 1 can be simplified to Equation 3 which states that rule-extraction methods need only be concerned with the weight of links entering a unit. This reduces rule extraction to a search of each unit for sets of incoming links whose summed weights exceeds guarantees that the unit's bias is exceeded, regardless of the activation carried by other incoming links.

Rule extraction is further simplified by Equation 2 which guarantees that non-input units always have non-negative activations. (In the problems we investigate, input units always have activations in $\{0, 1\}$.) Therefore, negatively-weighted links can only give rise to negated antecedents while positively-weighted links can only give rise to non-negated antecedents. This considerably reduces the size of the search space (Fu, 1991).

Finally, note that short of exactly copying a unit and its links, there is no way to limit the kinds of differences that extracted rules will have from a network of reasonable complexity. That is, extracted rules may make both errors of omission and of 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. One of the chief problems for rule extraction is to minimize such errors without simply rewriting the network.

3.3 'Subset' algorithms

The first method for rule extraction that we describe is fundamentally similar to the approach described by Saito and Nakano (1988), as well as Fu (1991). However, the particular implementation is our own. We call this general approach the 'subset' method because it explicitly searches for subsets of incoming weights that exceed the bias on a unit. While this paper uses a subset algorithm as a "straw man" to show the effectiveness of our new method for rule extraction (presented in the next section), subset algorithms represent the state of the art in the published literature.

A simple, breadth-first 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 algorithm removes subsumed and overly-general rules. For example, given that the link weights and the bias are as shown in Figure 3a, a subset algorithm would initially find five rules. After eliminating one subsumed rule, the algorithm returns the four rules listed in Figure 3b (assuming that all units have activations near zero or one).

Table 2: Specification of our SUBSET algorithm.

With each hidden and output unit:

A. Extract up to β_p subsets of the positively-weighted incoming links whose summed weight is greater than the bias on the unit.

B. With each subset $\mathcal{P}$ of the β_p subsets found in step A:

1. Extract up to 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.
2. Let $\mathcal{Z}$ be a new predicate used nowhere else.
3. With each subset $\mathcal{N}$ of the β_n subsets found in step B.1 form the rule:
   "if $\mathcal{N}$ then $\mathcal{Z}$."
4. Form the rule:
   "if $\mathcal{P}$ and not $\mathcal{Z}$ then <name of unit>."

Figure 3 approximately here.

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 behavior of simple networks working on small problems. To avoid their otherwise prohibitive combinatorics, all implementations of subset algorithms use heuristics. For instance, Saito and Nakano (1988) establish a ceiling on the number of antecedents in extracted rules.

Establishing an *a priori* ceiling on the number of antecedents has several shortcommings. 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 a large number of antecedents. For instance, the initial rules for one the real-world domains we study indicate that the smallest bound on the number of antecedents that can be safely set could require considering more than 10^5 sets. Therefore, rather than setting a bound on the number of antecedents, our implementation uses a branch-and-bound algorithm that is limited in terms of the number of rules it may find.

Our SUBSET algorithm is presented in pseudocode in Table 2. (See Fu (1991) for a more complete description of a subset algorithm.) Briefly, 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 positively-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 weight of the positive subset and the unit's bias. Finally, the algorithm forms rules which result in the consequent being satisfied when all of the antecedents of a 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. Hence, we must ensure that each of these minimal subsets contains at least one unsatisfied antecedent.

This algorithm may discover up to $\beta_p * (1 + \beta_n)$ rules for each non-input unit in the network to be translated. As described above, there is a tradeoff between the number of extracted rules and accurate reproduction of the network's behavior. Hence, we set β_p and β_n to find reasonably-sized sets of rules that approximate the network from which they came. The settings we use result in finding about 300 rules in each of the real-world domains we studied. In general, more accurate reproduction of KNNs require many more rules.

While SUBSET extracts a large set of rules, it is smaller than many handcrafted expert systems (e.g., McDermott, 1982). Hence, SUBSET delivers sets of rules that are, at least potentially, tractable. However, these rules tend to hide significant structures in trained networks. For instance, in Figure 3a 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 3b 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.
```

Our recognizing the structure shared among several conjunctive rules led to the development of the algorithm described next that is the focus of this paper.

3.4 The MOFN method

Our algorithm, called MOFN, addresses both the combinatorial and presentation problems 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 suggested previously, this method arose because we noticed that rule sets discov-

Table 3: The MOFN approach to rule extraction.

1. With each hidden and output unit, form groups of similarly-weighted links.
2. Set link weights of all group members to the average of the group.
3. Eliminate any groups that do not significantly affect whether the unit will be active or inactive.
4. Holding all link weights constant, optimize biases of all hidden and output units using the backpropagation algorithm.
5. 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. Where possible, simplify rules to eliminate superfluous weights and thresholds.

ered by SUBSET often contain M-of-N style concepts. Further support for this method comes from experiments that indicate neural networks are good at learning M-of-N concepts (Fisher & McKusick, 1989), as well as experiments with a variant of ID3 that show a bias towards M-of-N style concepts is useful (Murphy & Pazzani, 1991). Finally, note that purely conjunctive rules result if $N = M$, while a set of disjunctive rules results when $M = 1$; hence, using M-of-N rules does not restrict generality.

The idea underlying MOFN, an abstracted version of which appears in Table 3, 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.

3.4.1 The MOFN algorithm

This section contains a detailed description of our rule-extraction algorithm. We illustrate the algorithms with an example in the following section.

Step 1, clustering. Backpropagation training tends to group links in KNNs into loose clusters rather than the equivalence classes assumed by the MofN algorithm.[3] Hence, the first step of MofN creates equivalence classes. We do this by clustering links using a standard clustering method; the *join* algorithm (Hartigan, 1975). This method operates by successively combining the two closest clusters; it starts which each cluster holding a single link. Clustering stops when no pair of clusters is closer than a set distance (MofN uses 0.25).

Step 2, averaging. After groups are formed, the second step of the algorithm sets the weight of all links in each group to the average of each group's weight. Thus, these first two steps force links in the network into equivalence classes as required by the rest of the algorithm.

Step 3, eliminating. With equivalence classes in place, the procedure next attempts to identify and eliminate those groups that are unlikely to have any bearing on the calculation of the consequent. Such groups generally have low link weights and few members. Elimination proceeds via two paths: one algorithmic, and one heuristic.

The first elimination procedure algorithmically attempts to find clusters of links that cannot have any effect on whether or not the total incoming activation exceeds the bias. This is done by calculating the total possible activation that each cluster can send. The total possible activation is then compared to the levels of activation that are reachable given the link weights in the network. Clusters that cannot change whether the net input exceeds the bias are eliminated. Note that this procedure is very similar to Subset. However, clustering of link weights considerably reduces the combinatorics of the problem.

The heuristic elimination procedure is based explicitly upon whether the net input received from a cluster ever is necessary for the correct classification of a training example. This procedure operates by presenting each training example to the clustered network and sequentially zeroing the input from each cluster. If doing so results in a qualitative change in the activation of the unit receiving activation from the cluster, then the cluster is marked as necessary. Clusters not marked as necessary are eliminated.

[3]Nowlan and Hinton's (1991) neural-network training algorithm draws link weights into clusters. It reduces, but does not eliminate, this problem. One of our future research plans is to combine their training algorithm with our knowledge-insertion and rule-extraction methods.

Step 4, optimizing. With unimportant groups eliminated, the fourth step of MOFN is to optimize the bias on the unit. This step is necessary because the first three steps can change the times at which a unit is active. As a result, altered networks may have error rates that are significantly higher than they were at the end of training.

Optimization can be done by freezing the weights on the links so that the groups stay intact and then retraining the biases of the network using backpropagation. To reflect the rule-like nature of the network, the activation function is slightly modified so that it more-closely resembles a step function.

Step 5, extracting. This step of the MOFN algorithm forms rules that exactly re-express the network. These rules are created by directly translating the bias and incoming weights to each unit into a rule with weighted antecedents such that the rule is true if the sum of the weighted antecedents exceeds the bias. Note that because of the equivalence classes and elimination of groups, these rules are considerably simpler than the original trained network; they have fewer antecedents and the antecedents tend to be in a few weight classes.

Step 6, simplifying. In the sixth, and final, step rules are simplified whenever possible to eliminate weights and thresholds. MOFN simplifies a rule by scanning it to determine the possible combinations of antecedents that exceed the rule's threshold (i.e., bias). This scan may result in more than one rule. Hence, there is a tradeoff in the simplification procedure between complexity of an individual rule and complexity resulting from a number of syntactically simpler rules.

For example, consider the rule[4]:

```
A :- 10.0 < 5.1 x number-true{B, C, D, E} +
            3.5 x number-true{X, Y, Z}
```

Simplification rewrites this rule with the following three rules:

```
A :- 2 of {B, C, D, E}.
A :- 1 of {B, C, D, E} and 2 of {X, Y, Z}.
A :- X, Y, Z.
```

If the elimination of weight and biases requires rewriting a single rule with more than five rules, then the rule is left in its original state.

[4]The function `number-true` returns the number of antecedents in the following set that are true.

3.4.2 Example of MofN

Figure 4 illustrates 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 four links with weight 1.1 and one of 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. Hence, the final, M-of-N can be written out by inspection.

Figure 4 approximately here.

3.4.3 Complexity analysis of MofN

The complexity of MofN cannot be precisely analyzed because the bias-optimization phase uses backpropagation. But, note that the problem addressed in bias optimization is considerably simpler than the initial training of the network. Usually, networks have more than an order of magnitude fewer links during bias optimization than during initial training. Moreover, only the biases are allowed to change. As a result, this step takes a reasonably short time.[5] The initial clustering, requires $O(u \times l^2)$ time, u is the number of units and l is the average number of links received by a unit, and cluster elimination requires $O(n \times u \times l)$ time, n is the number of training examples. Each of the other steps uses $O(u \times l)$ time.

3.5 Subset and MofN: Summary

Both of these rule-extraction algorithms have strengths with respect to the other. For example, the individual rules returned by Subset are often more easily understood than those returned by MofN. However, because Subset generates many more rules (which can be repetitive) than MofN, the rule sets returned by MofN are usually easier to understand than those of Subset. Moreover, measuring time with respect to the number of links to each unit, Subset is an exponential algorithm whereas MofN is

[5] Training some simple classes of neural networks has been proven NP-complete (Judd, 1988). Hinton (1989) suggests that, in practice, backpropagation usually runs in $O((u \times l)^3)$ time.

approximately cubic. Finally, results presented later in this paper indicate that the rule sets derived by MOFN are approximately equal to the accuracy of the networks from which they are extracted, while the rules extracted by SUBSET are significantly worse. The next sections contain a series of empirical tests that demonstrate these and other differences between these two algorithms.

4 Experiments with Real-World Datasets

4.1 Datasets

To test the efficacy of our rule-extraction procedure on real-world problems, we studied two problems from the domain of molecular biology. We use these datasets because they are members of a small collection of real-world problems for which there exist both an approximately-correct domain theory and a set of classified examples. Also, both datasets have been previously studied in the theory revision literature (e.g., Towell *et al.*, 1990; Ourston & Mooney, 1990; Thompson *et al.*, 1991; Noordewier *et al.*, 1991) making possible some of the comparisons we perform in this section.

4.1.1 Notation

In both of these datasets, a special notation is used to simplify specifying locations in a DNA sequence. (DNA sequences are linear strings of 'nucleotides'; each nucleotide is drawn from {A, G, T, C}.) The idea is to number locations with respect to a fixed, biologically-meaningful, reference point. Negative numbers indicate sites preceding (to the left of) the reference point while positive numbers indicate sites following the reference point. Hence, when a rule's antecedents refer to input features, they first state a location relative to the reference point in the sequence vector, then a DNA nucleotide, or a sequence of nucleotides, that must occur (e.g., "@3 'A' "). By biological convention, position numbers of zero are not used. In the rule specifications, `'-'` indicates that *any* nucleotide will suffice. For example, the first rule for `conformation` in Table 5 says that there must be an 'A' 45 nucleotides *before* the reference location. Another 'A' must be at position -44, then any two nucleotides can appear, and finally there must be a 'A' at location -41. Finally, we use standard ambiguity codes, given in Table 4, to refer to acceptable alternatives at a particular location.

Table 4: Ambiguity codes for DNA nucleotides.

Code	*Meaning*	*Code*	*Meaning*	*Code*	*Meaning*
M	A or C	R	A or G	W	A or T
S	C or G	Y	C or T	K	G or T
V	A or C or G	H	A or C or T	D	A or G or T
B	C or G or T	X	A or G or C or T		

4.1.2 Problem 1: promoter recognition

The first problem we investigate, (prokaryotic) *promoter recognition*, was originally described by Towell *et al.* (1990). Promoters are short DNA sequences that initiate the expression of a gene. Basically, a promoter is a site where the protein *RNA polymerase* binds to DNA. According to the rules in Table 5 there are two sites at which this binding must occur – the minus 10 and minus 35 regions. In addition, the `conformation` rules attempt to capture the "twist" of the DNA helix, thereby ensuring that the binding sites are spatially aligned.

This set of rules was derived in a straightforward manner from the biological literature (Harley & Reynolds, 1987; Hawley & McClure, 1983; Koudelka *et al.*, 1987) on "consensus sequences" for promoters. Consensus sequences describe the most probable nucleotides at a given location. However, no nucleotide appears in every promoter. For instance in the `minus-10` rules, the probability of each nucleotide in `'T A - - - T'` is about 0.85. Also, these rules were derived from multiple sources. Because we did not prune subsumed rules after extraction, some rules are strictly less general than others.

Figure 5 approximately here.

This rule set is translated by KBANN into a neural network with the topology shown in Figure 5. Recall that KBANN adds additional low-weighted links (not shown) so that if additional sequence information is relevant, the algorithm can capture that information during training.

The input features for promoter recognition are 57 sequential DNA nucleotides. (Each location in the DNA sequence is translated into 4 inputs units, one for each nucleotide. Hence, the promoter network has 228 input units.) For this problem, the reference point is the site at which gene transcription would begin if the example contained a promoter. This point is located so that there are 50 nucleotides preceding, and seven following,

Table 5: Initial rules for promoter recognition.

```
promoter   :- contact, conformation.
contact    :- minus-35, minus-10.

minus-35 :- @-37 'CTTGAC-'.          minus-10 :- @-14 'TATAAT--'.
minus-35 :- @-37 '-TTGACA'.          minus-10 :- @-14 '-TA-A-T-'.
minus-35 :- @-37 '-TTG-CA'.          minus-10 :- @-14 '-TATAAT-'.
minus-35 :- @-37 '-TTGAC-'.          minus-10 :- @-14 '--TA---T'.

conformation  :-  @-45 'AA--A'.
conformation  :-  @-45 'A---A',      @-28 'T---T-AA--T-',  @-04 'T'.
conformation  :-  @-49 'A----T',     @-27 'T----A--T-TG',  @-01 'A'.
conformation  :-  @-47 'CAA-TT-AC',  @-22 'G---T-C',       @-08 'GCGCC-CC'.
```

the reference point. (Thus, positive examples contain the first seven nucleotides of the transcribed gene.)

The set of examples contains 53 sample promoters and 53 nonpromoter sequences. The 53 sample promoters were obtained from a compilation produced by Hawley and McClure (1983). We derived negative training examples by randomly selecting contiguous substrings from a 1.5 kilobase sequence provided by Prof. T. Record of the University of Wisconsin's Chemistry Department. This sequence is a fragment from *E. coli* bacteriophage *T7* isolated with the restriction enzyme *Hae*III. By virtue of the fact that the fragment does not bind RNA polymerase, it is believed to not contain any promoter sites (Record, personal communication).

Prior to training, the rules in Table 5 do not classify any of the 106 examples as promoters. As a result, the rules by themselves are not very useful; they "just say no."

4.2 Problem 2: splice-junction determination

The second problem we examine is (primate) splice-junction determination. Originally described as a learning problem by Noordewier *et al.* (1991); splice junctions are the points on a DNA sequence at which 'superfluous' DNA is removed during the process of protein creation in higher organisms. The problem posed by this dataset is to recognize in a given sequence of DNA the boundaries between *exons* (i.e., the parts of the DNA sequence retained after splicing) and *introns* (i.e., the parts of the DNA sequence that are spliced out). This problem consists of two subtasks: recognizing exon/intron boundaries, and recognizing intron/exon boundaries.

The dataset for this problem contains 3190 examples, of which approximately 25% are exon/intron boundaries, 25% are intron/exon boundaries and the remaining 50% are neither. Each example consists of a 60-nucleotide long DNA sequence categorized according to the type of boundary at the center of the sequence. (The center of the sequence is the reference location used for numbering nucleotides.) For the experiments presented in this paper, we randomly selected 1000 examples from the 3190 examples available.

In addition to the examples, the information about splice-junctions includes a set of 21 rules derived from standard biological textbooks. (The rule set is not shown; it appears in Noordewier *et al.* (1991) and Towell, 1991.) As for promoter recognition, the success rate of the splice junction rules is due largely to their tendency to "just say no"; the rules correctly classify only 40% of the I/E and 3% of the E/I examples.

4.3 Experiments

This section presents a set of experiments designed to determine the relative strengths and weaknesses of the two rule-extraction methods described in Section 3. We compare rule-extraction techniques using two measures: *quality*, which is measured both by the accuracy of the rules and their fidelity to the network from which they were extracted, and *comprehensibility* which is measured both by characterizing whole sets of rules and by looking at individual rules. The final part of this section, which examines the meanings of individual rules, includes samples of the rules extracted from trained KNNs by both SUBSET and MOFN.

4.3.1 Systems and methodology

In addition to comparing our MOFN algorithm to SUBSET, we compare MOFN to three "symbolic" algorithms: EITHER (Ourston & Mooney, 1990), C4.5 (Quinlan, 1987), and LINUS (Dzeroski & Lavrac, 1991). The first of these algorithms, EITHER (Ourston & Mooney, 1990), is a method for empirically adapting a set of propositional rules so that they are correct on the training examples. C4.5 (Quinlan, 1987), is distinct from the other algorithms we compare, in that it builds decision trees without using any background knowledge. However, it extracts rules from those trees so like all other systems we consider, the final result of C4.5 is a set of rules. The final algorithm, LINUS, augments the set of input features with Boolean features that represent the truth value of each consequent in the rule set. Hence, in the promoter domain, LINUS has 244 features available (228 original plus 16 rules). We use C4.5 as the inductive component of LINUS. Hence, the result of LINUS is a set of rule that may include some of the consequents of the original rules.

Training and testing methodology for neural networks: We train networks until either: (a) all training examples are correct to within 0.20 of the desired output activation, (b) every example has been presented 500 times, or (c) no improvement in classification occurs for five presentations of every training example. (The last termination criterion is based upon Fahlman and Lebiere's (1989) "patience" metric.) Following Hinton's (1989) suggestion for improved network interpretability, all weights are subject to gentle "decay" during training.[6] In addition, networks are trained using the *cross-entropy* error function (Hinton, 1989), as our experience indicates that it is better able to handle the type of errors that occur in KNNs than the standard error function (Rumelhart *et al.*, 1986). During testing, examples are considered correctly classified when all outputs are within 0.5 of correct.

Training and testing methodology for LINUS and C4.5: C4.5 is trained using the defaults in the code provided by R. Quinlan. Specifically, we use ten trials of the windowing procedure on each training set to build ten decision trees. Those trees are transformed into ten rule sets. Finally, an 'optimal' set of rules is derived from the ten rule sets. The optimal rule set is used to assess generalization.

Training and testing methodology for EITHER: We neither trained nor tested EITHER. Rather, the numbers we report are derived from Ourston's thesis (1991). Because EITHER is unable to work on domain theories that include negation, we can not report results for EITHER on the splice-junction domain.

4.3.2 Quality

The issue of overriding importance to this work is the quality of the extracted rules. As a measure, *quality* is at least two dimensional. First, the rules must accurately categorize examples that were not seen during training. If the rules lose the advantage in accuracy that KBANN provides over symbolic learning algorithms (Towell *et al.*, 1990), then there is little value in rule extraction. It would be simpler to use an "all symbolic" method to

[6]That is, to the standard weight change function a term ϕ is added ($0 < \phi < 1$). Thus, the weight change formula becomes: $w_{ij}(t) = \phi * (w_{ij}(t-1) + \Delta_{ij})$ where Δ_{ij} is the standard weight adjustment (Rumelhart *et al.*, 1986). Weights change after each example presentation, so we use a very gentle $\phi = 0.99999$.

develop rules, while using KBANN only to develop highly-accurate, but not understandable, classifiers. Second, the extracted rules must capture the information contained in the KNN. This is necessary if the extracted rules are to be useful for gaining an understanding of exactly what the KNN learned during training. On each of these measures, we show that MOFN is superior to SUBSET. We also show that the accuracy of the rules extracted by MOFN is equal, or superior, to the symbolic algorithms. More importantly, we show that the MOFN method extracts rules that are equivalent to the networks at classifying testing examples.

Accuracy. To assess the accuracy of each system we use ten repetitions of 10-fold cross-validation (Weiss & Kulikowski, 1990). Figure 6 plots the average of our ten-fold cross-validation runs. For comparison, Figure 6 includes the accuracy of the trained KNNs prior to rule extraction (the bars labeled "Network").

Figure 6 approximately here.

Recall that the initial rule sets for promoter recognition and splice-junction determination correctly categorized 50% and 61%, respectively, of the examples. Hence, each of the systems plotted in Figure 6 is superior to the initial rules. On the Promoter problem, C4.5, LINUS, EITHER and SUBSET are approximately equivalent in test-set accuracy. These methods lag significantly behind our MOFN method. Somewhat surprisingly, C4.5 outperforms LINUS, although the difference is not statistically significant. While, the accuracy of the rules extracted by EITHER and SUBSET are approximately equal, the method of achieving that accuracy is quite different. Whereas EITHER is perfect on the training set, SUBSET is only marginally better on the training than on the testing set. Finally, the rules extracted by MOFN are significantly superior (with at least 99% confidence) to the rules extracted by any of the other methods investigated. Moreover, the difference in accuracy between the rules extracted by MOFN and the networks from which they came is not statistically significant.

The pattern of results is somewhat different on the splice junction problem largely as a result of the relative improvement of C4.5 and LINUS. In this domain, the results for C4.5, LINUS, MOFN, and the original networks are statistically indistinguishable. The rules generated by the SUBSET method are much worse. Perhaps the most interesting result on this dataset is that the MOFN rules outperform the networks from which they came in terms of training set accuracy. Earlier tests (Towell & Shavlik, 1991) showed the error

Table 6: Fidelity of extracted rules to trained KNNs on training examples.

Rule Extraction Method	Overall Percent Agreement	Probability of agreement between KNN and extracted rules given:	
		KNN correct	KNN incorrect
Splice Junction			
SUBSET	86%	0.87	0.38
MOFN	93%	0.95	0.31
Promoter			
SUBSET	87%	0.90	0.24
MOFN	99%	0.99	0.10

rate of the extracted rules on the promoter problem was also below that of the networks from which the rules were extracted. However, alterations to the training method (to reduce overfitting) improved the accuracy of the networks without significantly affecting the accuracy of the extracted rules. In Section 4.4 we further analyze this result. The error rate of the SUBSET rules on testing examples is statistically worse than the rules from all of the other learning methods.

Fidelity. By *fidelity*, we mean the ability of the extracted rules to mimic the behavior of the network from which they are extracted. Fidelity is important in two circumstances: (1) if the extracted rules are to be as a tool for understanding the behavior of the network, and (2) if the extracted rules are to be used as a method of transferring the knowledge contained in the network. Table 6 indicates that MOFN is superior to SUBSET at reproducing the behavior of the network on the examples seen during training. (These results are obtained from the trials used in the previous section.) For instance, on the splice-junction data, rules extracted using MOFN give the same answers as trained KNNs 93% of the time, while rules extracted by SUBSET match only 86% of the time. In addition, Table 6 shows that both methods of rule extraction are much better at mimicking the behavior of the network when the network is correct, than when it is incorrect.

4.3.3 Comprehensibility

To be useful, the extracted rules must not only be accurate, they also must be *understandable*. While an ill-defined concept, there are several ways in which "understandability" might be measured. One approach is to look at statistics describing whole sets of rules.

An alternative is to examine whether individual rules are meaningful. The following sections present results on both of these measures.

Global Comprehensibility. The first dimension along which we characterize the comprehensibility of sets of rules is rule-set size. Size is a concern because sets with large number of rules can be difficult, if not impossible, to understand. Figure 7 addresses the issue of rule-set size by plotting each rule-extraction method in a space spanned by the number of extracted rules and total number of antecedents in those rules. (The data in Figure 7 represents an average over the cross-validation study reported in Figure 6.) Unfortunately, counting rules and antecedents for MOFN and for networks is not straightforward. For MOFN, the simplification phase may increase the number of rules and require the reuse of some antecedents. The data in both Figures 7 and 8 blithely ignore these complications, reflecting only the rule and antecedent counts of unsimplified rule sets. For trained networks, the count of antecedents includes each link whose weight is within two orders of magnitude of the bias on the unit receiving the link. Weights of lesser size are not counted because they have little effect. Finally, rule and antecedent counts for LINUS include the antecedents of the derived features which LINUS used in its rules. Had the derived features been counted as a single antecedent, the rule sets extracted by LINUS would be about the same size as those extracted by C4.5.

Figure 7 approximately here.

Taking the initial rule sets in each domain as a standard for interpretability, the rules refined by the 'symbolic' methods are likely to be easy to understand. (Recall, however, that these rules are significantly less accurate on the promoter domain.) Rule sets extracted by MOFN are also likely to be easily understood; they contain fewer rules and approximately the same number of antecedents as the initial rule sets. On the other hand, the rules SUBSET extracts are much less likely to be comprehensible.

A second important global statistic is the number of antecedents per rule. If this value is large, individual rules are unlikely to be understandable (Bruner *et al.*, 1956). Furthermore, negated antecedents add to the difficulty of evaluating rules (Nessier & Weene, 1962). However, the effects of negated antecedents are difficult to quantify. Weighted antecedents, such as those appearing in MOFN rules and which implicitly appear in trained networks, cloud the picture further. Rather than arbitrarily assigning a difficulty ranking to each type of antecedent, in Figure 8, we simply show the number of negative and positive antecedents to an average rule.

Figure 8 approximately here.

On this measure, as with rule-set size, the symbolic methods are the clear winners. The MOFN rules are slightly larger than SUBSET rules and contain more negative antecedents. Still, both methods return rules that are well within the limits of human comprehensibility (Miller, 1956) but recall MOFN extracts many fewer rules.

Individual Comprehensibility. Global statistics, such as those just discussed, are sufficient to indicate whether or not a whole rule set is likely to be comprehensible. Once it has been determined that a rule set is potentially comprehensible, it is necessary to look at individual rules and assess their meaning.

To make this assessment, we examine rule sets extracted by the MOFN and SUBSET methods after training a KNN on the entire set of examples from each problem domain. Tables 7 and 8 present the rules MOFN and SUBSET extract from the same network for promoter recognition. (The rules extracted from the splice-junction domain are not in this paper because they have much the same character as those of the promoter domain. They appear in Towell, 1991.) Because rule sets extracted by SUBSET can be very large, we set β_p and β_n to values too small to deliver reasonably accurate rule sets. As a result, the rules in Table 8 have about a 25% error rate on the training set.

While the rules extracted by MOFN in Table 7 are somewhat murky, they are vastly more comprehensible than the network of 3000 links from which they were induced. Moreover, these rules can be rewritten in a form very similar to one used in the biological community, namely weight matrices (Stormo, 1990).

One major pattern is apparent in the rules extracted by both MOFN and SUBSET. Specifically, the KNN learned to disregard `conformation`. The `conformation` rules are also dropped EITHER and are not often used by LINUS. This suggests that dropping these rules is not an artifact of our method but rather that DNA nucleotides outside the *minus35* and *minus10* regions are less important than the conformation hypothesis (Koudelka *et al.*, 1987) suggests. Hence, we demonstrate that machine learning methods can provide valuable evidence confirming or refuting biological theories.

In general, the rules MOFN extracts confirm the importance of the bases identified in the initial rules. However, whereas the initial rules required matching every base, the extracted rules allow a less than perfect match. In addition, the extracted rules point to

Table 7: Promoter rules extracted by MOFN.

```
Promoter :- Minus35, Minus10.

Minus-35:-10 < 4.0 x nt(@-35 ‘TTGAT-’)+
               1.5 x nt(@-35 ‘--TCC-’)+
               0.5 x nt(@-35 ‘-MC---’)-
               1.5 x nt(@-35 ‘GGAGG-’).

Minus-35:-10 < 5.0 x nt(@-35 ‘T-G--A’)+
               3.1 x nt(@-35 ‘-GT---’)+
               1.8 x nt(@-35 ‘-CC-MT’)+
               1.8 x nt(@-35 ‘CAA-GG’)-
               3.1 x nt(@-35 ‘A----C’).

Minus-35:-             @-36 ‘C-TGAC-’.

Minus-35:-              @-35 ‘TTD-CA’.

Minus-10:-             @-13 ‘TAWA-T--’.

Minus-10:-          2 of @-12 ‘-CA---T’ and
               not (1 of @-12 ‘-RB---S’).

Minus-10:-10 < 3.0 x nt(@-12 ‘TAT--T-’) +
               1.8 x nt(@-12 ‘---GA--’) +
               0.7 x nt(@-12 ‘--GAT--’) -
               0.7 x nt(@-12 ‘GKCCCS-’).

Minus-10:-10 < 3.7 x nt(@-12 ‘KA-M-T-’) +
               1.0 x nt(@-12 ‘-T---A-’) -
               1.0 x nt(@-12 ‘CS-G-S-’) -
               3.0 x nt(@-12 ‘A--T---’).
```

See Table 4 for meanings of letters other than A, G, T, and C.

"nt()" returns the number of named antecedents that match the given sequence. So, nt(@-14 ‘- - - C - - G - -’) would return 1 when matched against the sequence @-14 'AAACAAAAA'.

places in which changes to the sequence are important. For instance, in the final `minus10` rule, a ‘T’ in position 8 is a strong indicator that the rule is true. However, replacing the ‘T’ with a ‘G’ prevents the rule from ever being satisfied.

It is more difficult to get a clear picture of a promoter from the SUBSET rules (Table 8). The three `minus35` rules in the left column encode a simple 2-of-3 concept while the `minus10` rules approximate a 3-of-7 concept. Note that these patterns support the idea implemented in the MOFN method that a bias towards M-of-N style concepts is useful.

4.4 Discussion

The results presented in this section indicate that the MOFN method is able to extract a good set of rules from trained networks. In particular, the data supports our contentions that (1) the MOFN method is able to extract comprehensible rules from trained KNNs that reproduce the behavior of the KNN from which they were extracted and (2) the extracted rules are equivalent, or superior, to rules obtained through other neurally-based or symbolic rule-refinement methods. While rule sets produced by the MOFN algorithm

Table 8: Promoter rules extracted by SUBSET.

```
Promoter :- Minus35, Minus10.

Minus35  :- Minus35b, Minus35d.
Minus35  :- Minus35a, Minus35b.
Minus35  :- Minus35a, Minus35d.

Minus10  :- @-14 '-ATA----'.
Minus10  :- @-14 '-T--A-A-'.
Minus10  :- @-14 '--A-T-T-'.
Minus10  :- @-14 '-TA-----'.
Minus10  :- @-14 '-T------'.
Minus10  :- @-14 '---A---T'.
Minus10  :- @-14 '--T----T'.
Minus10  :- @-14 '--TA----'.
Minus10  :- @-14 'T-T-A---'.
Minus10  :- @-14 '-AT--T--'.
Minus10  :- @-14 '--TAA---'.
Minus10  :- @-14 '--TA-T--'.

Minus35  :- @-37 '-T-T--A'.
Minus35  :- @-37 '-TT---A'.
Minus35  :- @-37 '---G-CA'.
Minus35  :- @-37 '--T--C-'.
Minus35  :- @-37 '-T---CA'.

Minus35a :- @-37 'CT--A--'.
Minus35a :- @-37 'C--G-C-'.
Minus35a :- @-37 'C-T--C-'.
Minus35a :- @-37 'CT---C-'.
Minus35a :- @-37 'C---AC-'.

Minus35b :- @-37 '---GA--'.
Minus35b :- @-37 '--T--CA'.
Minus35b :- @-37 '-T---C-'.
Minus35b :- @-37 '---G-CA'.
Minus35b :- @-37 '----ACA'.

Minus35d :- @-37 '-TT--C-'.
Minus35d :- @-37 '-T-G-C-'.
Minus35d :- @-37 '--T-AC-'.
Minus35d :- @-37 '---GAC-'.
Minus35d :- @-37 '-T--AC-'
```

This abbreviated set of rules extracted by SUBSET has a test set accuracy of 75%. Sets whose statistics are reported previously in this section contain about 300 rules.

are slightly larger than those produced by the "all symbolic" approaches, MOFN's rule sets are small enough to be comprehensible by domain experts and are much more accurate. Hence, although weighing the tradeoff between accuracy and understandability is problem and user-specific, the MOFN algorithm for network-to-rules translation offers an appealing mixture.

Two of the results from this section deserve additional consideration: the superiority of MOFN's rules to those obtained by symbolic rule-refinement techniques, and the occasional superiority of MOFN's rules to the networks from which they were extracted. At first glance, it might seem that some of the power of our MOFN algorithm derives from its use of a more expressive language than other rule-refinement systems. This is not true because the antecedents and consequents in MOFN's rules are all Boolean. Hence, MOFN's rules can be written as a disjunction of conjunctive rules, at a great increase in the number of rules. So, this hypothesis must be rejected.

Still, two hypotheses explain the superiority of MOFN over rule-refinement methods that directly modify rules. First, the re-representation of a rule set as a neural network allows for more fine-grained refinement. When cast as a neural network, rules can be modified in very small steps. This may make it possible to more closely fit a target concept than when taking the large steps required by direct rule refinement.

A second hypothesis to explain the relative superiority of MOFN to symbolic methods of rule refinement is that MOFN may better fit the nature of the problem at hand. For instance, in the promoter problem, there are several potential sites at which hydrogen bonds can form between DNA and a protein; if enough of these bonds form, promoter activity can occur. The symbolic methods we investigated can not easily express M-of-N rules. Hence, some of the advantage of the MOFN method may result from its output language better fitting that of the "natural" language of DNA sequence-analysis problems. From a very general viewpoint, each problem may have some "natural" language in which it can be most parsimoniously solved. If the language used by the learning system is similar to the "natural" language, then the learning system should be able to effectively learn to solve the problem. If this hypothesis is correct, then we expect that problems which closely fit the inductive bias of symbolic rule-refinement systems will outperform our MOFN algorithm.

The observation that MOFN rules can be superior to the networks from which they are extracted cannot be attributed to a language bias; M-of-N style rules are only a subset of the languages expressible using neural networks. Instead, the advantage of the MOFN rules most likely occurs because the rule-extraction process reduces overfitting of the training examples. Several pieces of evidence support this hypothesis. First, we noted previously that revising network training procedures to reduce overfitting in the promoter domain eliminated the advantage that MOFN rules had over the networks from which they were extracted. That is, by reducing the degree to which the network fit the training data we eliminated the advantage that MOFN rules had over the networks from which they were extracted. Second, the difference in ability to correctly categorize testing and training examples is smaller for MOFN rules than for trained KNNs. In other words, the rules that the MOFN method extracts are only slightly better at classifying training examples than at classifying testing examples. (While the differences between training and testing set performance are smaller, they are statistically significant with 99.5% confidence using a one-tailed, paired-sample, *t*-test.) The rules SUBSET extracts also have this property, but they are much worse on the training set than both MOFN rules and trained KNNs on the testing set. As an aside, we do not expect that subset algorithms will be able to exceed the performance of MOFN on the two problems we investigated. However, subset algorithms might be expected to equal the accuracy of MOFN on problems whose solution matches their bias towards sets of conjunctive rules.

A third piece of evidence in support of the overfitting hypothesis comes from the work on pruning neural networks to eliminate superfluous parts (Mozer & Smolensky, 1988; Le Cun *et al.*, 1989).[7] Rule extraction is an extreme form of pruning in which links and units are pruned and actions are taken on the remaining network to transform the network into a set of rules. Pruning efforts have also led to gains in test set performance. Generally these researchers attribute these gains to reduced overfitting of the training examples.

5 Tests on Artificial Problems

In the previous section we explored the utility of our MOFN method in the context of two real-world problems. While the advantages of real-world problems are self-evident, there are disadvantages to real-world problems. In particular, it can be difficult to closely control a study that investigates the abilities of an algorithm. For instance, the tests in the previous section left unaddressed questions concerning the robustness of the MOFN method to flaws in the initial domain theory because we do not know the ways in which the domain theories are incorrect. Therefore, in this section we step away from the real-world, and look at two very artificial problems, the "MONKS problems" (Thrun *et al.*, 1991). In so doing, we gain control over the distance between the correct theory and the theory provided to the learning system. This control allows us to make additional predictions about the ability of our rule-extraction algorithm.

5.1 The MONKS Problems

We chose the MONKS problems as a testing domain because it is a widely tested, thoroughly documented problem which affords easy replication. The MONKS problems were originally described by Thrun *et al.* (1991). In their report, the authors describe tests of 25 machine learning algorithms.

The MONKS problems consist of a population of 432 examples defined over the six

[7] There is also a large body of literature in "symbolic" machine learning which suggests that methods to reduce overfitting of the training set can improve generalization (e.g., Quinlan, 1987).

features appearing at the top of Table 9. The first MONKS problem is to learn to classify the examples according to the domain theory appearing in the middle of Table 9. This domain theory is simply a disjunction of four conjunctions. This theory can be easily expressed by rules extracted from a KNN using either SUBSET or MOFN. The second MONKS problem, given by the rule near the bottom of Table 9 is considerably more complex. To express this concept as a disjunction of conjunctions requires 15 rules each of which has six antecedents. Alternately, this concept can be expressed by a single conjunctive rule with two clauses, each of which expresses an M-of-N concept. This rule appears at the bottom of Table 9.

The additional complexity of the second problem had a significant effect on the ability of the 25 algorithms tested by Thrun *et al.* (1991) to learn the concept. On the first problem, the tested algorithms were supplied with 124 training examples. Nine of the 23 systems tested on this problem correctly classified the whole population. Overall, the systems averaged 88.8% correct. Despite being supplied with more examples (169) for the second problem, only four of the 25 systems tested correctly classified the population. Overall, the systems averaged 76.3% correct; no system did better on the second problem than on the first.

5.2 Experiments

The question addressed by the experiments in this section is 'how difficult is it to recover the correct domain theory when provided with a theory that is not quite correct?' We define difficulty as the average size of the training set needed to correctly identify the entire population. (This definition of difficulty is similar to the *teaching dimension* (Goldman & Kearns, 1991) except that teaching dimension is not learning-system specific.) These experiments compare only the ability of a KNN to learn a concept to the ability of MOFN and SUBSET to extract rules that express the concept.

We modify the domain theories in three ways. First, we delete antecedents from rules. This tests for robustness to missing antecedents. Second, we add both negated and unnegated antecedents to rules. This tests for robustness to rules containing unnecessary antecedents. Third, we swap unnecessary antecedents for correct antecedents. This tests for the ability to simultaneously add and delete antecedents. We modified every rule in the theory. For instance, adding one antecedent to the rules for the first MONKS problem (see Table 9) creates a rule set with three rules which have three antecedents and one rule which has two antecedents.

In addition to these three types of changes, we modified each domain theory in accor-

feature name		*values*
head-shape	$\in$	{round, square, octagon}
body-shape	$\in$	{round, square, octagon}
smiling	$\in$	{yes, no}
holding	$\in$	{sword, balloon, flag}
jacket-color	$\in$	{red, yellow, green, blue}
has-tie	$\in$	{yes, no}

The features and values for the MONKS problems.

```
if (head-shape round)   and (body-shape round)   then MONK.
if (head-shape square)  and (body-shape square)  then MONK.
if (head-shape octagon) and (body-shape octagon) then MONK.
if (jacket-color red)                            then MONK.
```

The correct theory for the first MONKS problem.

```
if exactly two of {(head-shape   round) (body-shape round)
                   (smiling      yes)   (holding    sword)
                   (jacket-color red)   (has-tie    yes)}
then MONK.
```

The correct theory for the second MONKS problem.

```
if two of {(head-shape   round) (body-shape round)
           (smiling      yes)   (holding    sword)
           (jacket-color red)   (has-tie    yes)}
   AND not three of {(head-shape   round) (body-shape round)
                     (smiling      yes)   (holding    sword)
                     (jacket-color red)   (has-tie    yes)}
then MONK.
```

Reformulation of the second MONKS problem.

Table 9: The domain and correct theories for the MONKS problems.

dance with its representation. To the first domain theory we negated antecedents. In this case, the systems must learn that an antecedent has an effect that is exactly opposite to effect indicated by the provided domain theory. To the second domain theory, we altered the requirements on the number of present and absent antecedents. For instance, rather than requiring exactly two antecedents, we provided a domain theory that required exactly N antecedents ($N \in \{1, 3, 4, 5\}$). We also provided a domain theory that required N or more antecedents ($N \in \{1, 2, 3, 4, 5\}$) i.e., if $N = 3$ then matching 3, 4, 5, or 6 of the antecedents would satisfy the rule. In all, we examined five variants of the first MONKS problem and fifteen variants of the second MONKS problem.

Figures 9 and 10 summarize our results. These results are obtained by creating a KNN based on a corrupted domain theory. A copy of the KNN is trained using a small percentage of the population. After learning to correctly classify the whole training set, rules are extracted from the KNN using both MOFN and SUBSET. Then the KNN and the two sets of extracted rules are tested on the entire population. If all three correctly classify the population, then the trial is finished. Otherwise, the procedure is repeated using a new copy of the KNN and a slightly larger training set (which is chosen so that is includes all member of the previously training set). The size of the training set is increased until all three systems correctly identify the entire population or until the entire population is used for training. The result of this procedure is the size of the smallest training set required by the KNN and the two sets of extracted rules to correctly classify the entire population. Figure 9 plots an average over 25 repetitions of this procedure for five corruptions of the first MONK theory and eight corruptions of the second MONK theory. Figure 10, which uses data from the same trials as Figure 9, plots the average size, in terms of number of antecedents, of the rule sets extracted by SUBSET and MOFN. Note that in both of these figures, short bars are preferred.

KNNs were unable to completely learn the training data for the second MONKS problem under several of the noise conditions described above. In particular, KNNs failed when even a single required antecedent was swapped with an unnecessary antecedent. KNNs also failed to learn the second problem when told that the exact number of matched antecedents was more than three or when told that N or more was sufficient. In total, seven of our fifteen corruptions of the second domain theory resulted in KNNs unable to learn the training data. These seven cases are not plotted in Figures 9 and 10.

Figure 9 approximately here.

Figure 10 approximately here.

5.3 Discussion

The most significant trend in Figure 9 is that, in all but one case, MOFN learns the concept from fewer examples than the KNN requires. This result supports the suggestion made in Section 4 that some of the effectiveness of MOFN can be attributed to its action as a post-pruning method for the network. We make this suggestion because, in all cases where MOFN learned the concept while the KNN did not, the KNN was perfect on the training examples. The KNN's problem in classifying the full population was that spurious correlations among lowly-weighted links impacted the categorization of examples not seen during training. The rule-extraction process used by MOFN acts to eliminate the links that cause the KNN problems.

On the other hand, SUBSET does not explicitly prune the network. It merely attempts to reproduce the behavior of the network using Boolean logic. Hence, it is unsurprising that, on most problems, SUBSET requires more of the population to correctly identify the concept than either MOFN or the KNN. Yet, SUBSET occasionally does quite well, on several problems it requires fewer training examples than KNNs and on one problem fewer than MOFN. Although it is occasionally successful, it is often spectacularly bad, in one case it was complete unable to learn the first problem despite the KNN reliably acquiring the concept after seeing only 50% of the population.

Looking at the size of the concept identified (plotted in Figure 10), it is clear that in most cases MOFN exactly identifies the concept. For the first problem, MOFN extracts the original rules in four of the five trials. In the fifth trial, MOFN includes several useless antecedents in its extracted rules. Interestingly, this was one of the problems on which SUBSET failed. In every case which MOFN was able to learn the second problem, it extracted the correct rules.

Finally, it is moderately surprising that the KNN was unable to learn the second problem after corrupting the rule set because none of the corruptions were particularly severe. We had hoped that the network would simply resort to a knowledge-free start, at which point it would be expected to perform along the lines of a standard neural network. (A fully-connected randomly-initialized neural network with a single layer of hidden units trained using backpropagation requires 39% of the dataset to reliably learn the first problem and 47% percent of the data to reliably learn the second problem.) Instead, in each of the problems which the KNN was unable to learn, it fell into a local minima from which it was unable to escape. While the observation that prior knowledge can hinder learning is significant, the observation has been previously made (Pazzani, 1992) and is not the focus of this paper. (Not surprisingly, when the KNN was unable to learn the concept, MOFN and SUBSET rarely extracted rules that expressed the concept.) In conclusion, these experiments show that when a KNN is capable of learning the rules that underlie a concept, the MOFN method is able to extract that set of rules, or a close approximation thereof, from KNN.

6 Related Work

There are two distinct sets of work that are closely related to the rule-extraction methods we present in this paper. The first set contains "all symbolic" methods of learning from rules and examples (e.g., Ourston & Mooney, 1990; Thompson *et al.*, 1991; Dzeroski & Lavrac, 1991). These methods avoid having to extract rules after learning by performing symbolic manipulations directly on the rules. While these methods are appealing because they do not require a shift between symbolic and neural representations, the results presented in Figure 6 indicate that they are not as accurate on the problems investigated as the MofN method presented here.

The second set of related research attempts to extract rules from randomly-weighted ANNs with a single layer of hidden units. Both Saito and Nakano (1988) and Fu (1991) report a method similar to our Subset algorithm. Saito and Nakano looked at the input/output behavior of trained networks to form rules that map directly from inputs to outputs because they have no domain theory to provide names for hidden units. To limit the combinatorics inherent to algorithms like Subset, Saito and Nakano limited rules to four antecedents. However even with this limitation, they extracted more than 400 rules for just one output unit in an ANN with 23 outputs. Thus, while their method is potentially useful for understanding networks, it may drown researchers in a sea of rules.

Several groups have reported attempts to extract rules from networks that, like KNNs, have a well-defined architecture. Sestito and Dillon (1990) avoid combinatorial problems in their approach to rule extraction by transforming a network with J inputs and K outputs into a network with $J + K$ inputs. After training, they look for links from the original J inputs that have a similar weight pattern to one of the K additional inputs. While their method can discover hierarchical rule sets, to discover the relationship (robin *isa* bird *isa* animal) the network must have outputs for robin, bird and animal. Another method for extracting rules from specially-configured networks is McMillan, Mozer and Smolensky's (1991) *connectionist scientist game*, which uses neural networks to iteratively learn sets of propositional rules. This method is similar to our MofN algorithm except that it assumes that there are only two groups, one with a large positive weight, and one with zero weight. For many problems, including the MONKS problem studied in Section 5, this assumption is quite useful because it significantly constrains the search space. However, in real-world problems this assumption may prove overly restrictive.

Finally, there have recently been attempts to extract *fuzzy* rules from neural networks.

For instance, Berenji (1991) describes a method much like KBANN except that it begins and ends with fuzzy rules. Several other techniques for extracting fuzzy rules from neural networks have also been reported (Hayashi, 1990; Masuoka *et al.*, 1990; Bochereau & Bourgine, 1990).

7 Limitations and Future Work

The results presented in Section 4 indicate that the MOFN method is able to effectively extract rules from trained KNNs, thereby making KBANN an excellent method for refining existing rules. However, this method also has several limitations which have heretofore been only briefly mentioned. Some of the more significant limitations, along with a brief discussion of our plans for addressing these limitations, are:

- *The* MOFN *method requires that the network be knowledge based.* That is, networks must be initially comprehensible. We are investigating different methods of training networks that are not knowledge-based so that they yield networks upon which the MOFN method is effective. For instance, Nowlan and Hinton's (1991) training method which groups links into loose clusters may result in networks amenable to MOFN.

- *Large shifts in the meanings of units as a result of training can make extracted rules difficult to comprehend.* At a minimum, the system should flag such rules for review. A better solution would be to give KBANN some way of analyzing the extracted rules.

- *Domain theories may not provide a sufficiently-rich vocabulary to allow a KNN to accurately learn a concept.* When a KNN is missing terms that are required to express a concept, it will modify existing terms to cover the vocabulary shortfall. This often leads to large shifts in the meaning of terms (discussed in the previous point). In such cases is necessary to augment the KNN with additional hidden units. However, adding hidden units opens many of the issues raised in Section 6 of extracting rules from networks that are not initialized with a domain theory.

- *The system is yet to be tested on a broad range of problems.* As discussed above, sequence analysis problems often contain aspects that are M-of-N-like. So, the MOFN method may be ideally suited to sequence-analysis problems. The two problems also share aspects that are not related to sequence analysis. For example, the initial domain theory of both problems is overly specific. Empirical tests suggest that KBANN is slightly more effective when the domain theory is overly specific (Towell, 1991). Hence, tests on a broader range of datasets are needed to prove the generality of the method.

In addition to extending KBANN and the MOFN method to address these limitations, we are currently working on a rule-extraction algorithm that operates *during* the training of a KNN. Rather than allowing link weights to freely take on arbitrary values, this algorithm periodically clusters links weights. Training thus consists of alternating cycles of (a) standard weight adjustment and (b) rounding. (This is quite similar to the approach to rule extraction taken by McMillan, Mozer and Smolensky, 1991.) The form of the rules produced by this method will be similar to MOFN rules. However, the search paths through weight space taken by this algorithm will be quite different than the paths taken by the MOFN algorithm during training. Instead of undergoing the transitions from an interpretable set of rules to a black-boxish KNN and back to an interpretable set of rules, the on-line clustering algorithm should preserve the comprehensibility of the knowledge base during training. We also expect that insight into the training process can be gained by being able to inspect the KNN's knowledge base at various points during training.

Lastly, we are investigating ways of enhancing the comprehensibility of the rules returned by MOFN. While the rules extracted from networks by MOFN are much more comprehensible than the networks from which they are extracted, we are not completely satisfied with their comprehensibility. Our hope is that alternate forms of presentation, some enhancements of the MOFN algorithm, or different network training methods will improve the comprehensibility of the resulting rules.

8 Conclusions

This paper presents and empirically validates the MOFN method for the extraction of rules from knowledge-based neural networks. We have shown that the MOFN method, as a part of the KBANN system, forms a rule-refinement algorithm that results in rules which generalize better to examples not seen during training than rules produced by "all-symbolic" rule refinement algorithms. We attribute this generalization ability to the shift of representation of the initial domain theory from symbolic to neurally-based. This shift alters the bias of the system in ways that enhance its ability of accurately refine the initial rules. In particular, neural representations naturally admit M-of-N style concepts. In addition, neural representations allow gradations in the importance of antecedents which are difficult to achieve in symbolic learning systems.

Representation shifts such as the one used by the KBANN system can be difficult to effect, but can pay off handsomely in improved performance. The easiest representation shifts involve only training examples. The relative ease of this shift has resulted in a plethora of comparisons among empirical learning systems. Domain-specific knowledge is much more difficult to shift between representations. Our initial work with KBANN provided a method for shifting domain-specific knowledge in the form of propositional Horn clauses into neural networks (Towell *et al.*, 1990). This shift made the knowledge available to neural learning algorithms such as backpropagation. The result of our initial efforts were classifiers more accurate than those obtained using only training examples (Towell *et al.*, 1990; Noordewier *et al.*, 1991).

In one sense, those earlier results are a dead end, for the refined knowledge in the trained networks is inaccessible; the previous papers provide a method for shifting knowledge from symbolic to neural, but do not provide a method for shifting back to symbolic. Hence, in this paper we present methods for completing the *symbolic* $\Rightarrow$ *neural* $\Rightarrow$ *symbolic* circle. By so doing, we open up the dead end, making our highly-accurate neural classifiers accessible to both human-inspection and further learning by symbolically-oriented systems.

Experimental results indicate that our MOFN method can extract concise, intelligible rules from a trained KNN without a serious loss of accuracy. In fact, the extracted rules can be as accurate at classifying testing examples as the networks from which they came. Moreover, the extracted rules show the utility of machine learning techniques as a method of validating and refining real-world (in this case, biological) theories. Finally, the extra work required by our system, by comparison to "all symbolic" rule-refinement methods, is shown to be worthwhile in that our method provides superior rules at a small cost in comprehensibility. Hence, this work shows the value of shifting to a fine-grained representation for detailed corrections, and back out again for communication of those corrections.

9 Acknowledgments

This work is partially supported by Office of Naval Research Grant N00014-90-J-1941, National Science Foundation Grant IRI-9002413, and Department of Energy Grant DE-FG02-91ER61129. We wish to thank Michiel Noordewier for his construction of the two biological rule and data sets, and for general comments on this work. Comments by

Richard Maclin, Mark Craven, Denis Kibler and the three anonymous reviewers are also gratefully acknowledged. Finally, we thank Ross Quinlan for providing the code for C4.5.

Both the promoter recognition and splice-junction determination datasets are a part of the collection of datasets at U.C.-Irvine available via anonymous ftp from `ics.uci.edu` in the directory `pub/machine-learning-databases`.

References

Berenji, H. R. (1991). Refinement of approximate reasoning-based controllers by reinforcement learning. *Proceedings of the Eighth International Machine Learning Workshop* (pp. 475–479). Evanston, IL: Morgan Kaufmann.

Bochereau, L. & Bourgine, P. (1990). Extraction of semantic features and logical rules from a multilayer neural network. *International Joint Conference on Neural Networks* (Vol. 2) (pp. 579–582). Washington, D.C.: Erlbaum.

Bruner, J. S., Goodnow, J. J., & Austin, G. A. (1956). *A Study of Thinking.* New York: Wiley.

Dzeroski, S. & Lavrac, N. (1991). Learning relations from noisy examples: An empirical comparison of LINUS and FOIL. *Proceedings of the Eighth International Machine Learning Workshop* (pp. 399–402). Evanston, IL: Morgan Kaufmann.

Fahlman, S. E. & Lebiere, C. (1989). The cascade-correlation learning architecture. *Advances in Neural Information Processing Systems* (Vol. 2) (pp. 524–532). Denver, CO: Morgan Kaufmann.

Fisher, D. H. & McKusick, K. B. (1989). An empirical comparison of ID3 and backpropagation. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 788–793). Detroit, MI: Morgan Kaufmann.

Fu, L. M. (1991). Rule learning by searching on adapted nets. *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 590–595). Anaheim, CA: AAAI Press.

Goldman, S. A. & Kearns, M. J. (1991). On the complexity of teaching. *Proceedings of the Fourth Annual Workshop on Computational Learning Theory* (pp. 303–314). Santa Cruz, CA: Morgan Kaufmann.

Harley, C. B. & Reynolds, R. P. (1987). Analysis of *E. coli* promoter sequences. *Nucleic Acids Research*, 15, 2343–2361.

Hartigan, J. A. (1975). *Clustering Algorithms.* New York: Wiley.

Hawley, D. K. & McClure, W. R. (1983). Compilation and analysis of escherichia coli promoter DNA sequences. *Nucleic Acids Research*, 11, 2237–2255.

Hayashi, Y. (1990). A neural expert system with automated extraction of fuzzy if-then rules. *Advances in Neural Information Processing Systems* (Vol. 3) (pp. 578–584). Denver, CO: Morgan Kaufmann.

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence*, 40, 185–234.

Judd, S. (1988). On the complexity of loading shallow neural networks. *Journal of Complexity*, 4, 177–192.

Koudelka, G. B., Harrison, S. C., & Ptashne, M. (1987). Effect of non-contacted bases on the affinity of 434 operator for 434 repressor and Cro. *Nature*, 326, 886–888.

Le Cun, Y., Denker, J. S., & Solla, S. A. (1989). Optimal brain damage. *Advances in Neural Information Processing Systems* (Vol. 2) (pp. 598–605). Denver, CO: Morgan Kaufmann.

Masuoka, R., Watanabe, N., Kawamura, A., Owada, Y., & Asakawa, K. (1990). Neurofuzzy system — fuzzy inference using a structured neural network. *Proceedings of the International Conference on Fuzzy Logic & Neural Networks* (pp. 173–177). Iizuka, Japan.

McDermott, J. (1982). R1: A rule-based configurer of computer systems. *Artificial Intelligence*, 19.

McMillan, C., Mozer, M. C., & Smolensky, P. (1991). The connectionist scientist game: Rule extraction and refinement in a neural network. *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society.* Chicago, IL: Erlbaum.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63, 81–97.

Mozer, M. C. & Smolensky, P. (1988). Skeletonization: A technique for trimming the fat from a network via relevance assessment. *Advances in Neural Information Processing Systems* (Vol. 1) (pp. 107–115). Denver, CO: Morgan Kaufmann.

Murphy, P. M. & Pazzani, M. J. (1991). ID2-of-3: Constructive induction of N-of-M concepts for discriminators in decision trees. *Proceedings of the Eighth International Machine Learning Workshop* (pp. 183–187). Evanston, IL: Morgan Kaufmann.

Nessier, U. & Weene, P. (1962). Hierarchies in concept attainment. *Journal of Experimental Psychology*, 64, 640–645.

Noordewier, M. O., Towell, G. G., & Shavlik, J. W. (1991). Training knowledge-based neural networks to recognize genes in DNA sequences. *Advances in Neural Information Processing Systems* (Vol. 3) (pp. 530–536). Denver, CO: Morgan Kaufmann.

Nowlan, S. J. & Hinton, G. E. (1991). Simplifying neural networks by soft weight-sharing. *Advances in Neural Information Processing Systems* (Vol. 4) (pp. 993–1000). Denver, CO: Morgan Kaufmann.

Ourston, D. (1991). *Using Explanation-Based and Empirical Methods in Theory Revision.* PhD thesis, Austin, TX: Deptartment of Computer Sciences, University of Texas.

Ourston, D. & Mooney, R. J. (1990). Changing the rules: A comprehensive approach to theory refinement. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 815–820). Boston, MA: AAAI Press.

Pazzani, M. (1992). When prior knowledge hinders learning. *Workshop Notes of Constraining Learning with Prior Knowledge* (pp. 44–52). San Jose, CA.

Pratt, L. Y., Mostow, J., & Kamm, C. A. (1991). Direct transfer of learned information among neural networks. *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 584–589). Anaheim, CA: AAAI Press.

Quinlan, J. R. (1987). Simplifying decision trees. *International Journal of Man-Machine Studies*, 27, 221–234.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the microstructure of cognition. Volume 1: Foundations.* Cambridge, MA: MIT Press.

Saito, K. & Nakano, R. (1988). Medical diagnostic expert system based on PDP model. *Proceedings of IEEE International Conference on Neural Networks* (Vol. 1) (pp. 255–262). San Diego, CA: IEEE.

Sestito, S. & Dillon, T. (1990). Using multi-layered neural networks for learning symbolic knowledge. *Proceedings of the Fourth Australian Joint Conference on Artificial Intelligence.* Perth, Australia: World Scientific.

Shavlik, J. W., Mooney, R. J., & Towell, G. G. (1991). Symbolic and neural net learning algorithms: An empirical comparison. *Machine Learning*, 6, 111–143.

Stormo, G. D. (1990). Consensus patterns in DNA. *Methods in Enzymology* (Vol. 183). Orlando, FL: Academic Press.

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

Thompson, K., Langley, P., & Iba, W. (1991). Using background knowledge in concept formation. *Proceedings of the Eighth International Machine Learning Workshop* (pp. 554–558). Evanston, IL: Morgan Kaufmann.

Thrun, S., Bala, J., Bloedorn, E., Bratko, I., Cestnik, B., Cheng, J., De Jong, K., Dzeroski, S., Fahlman, S., Fisher, D., Hamann, R., Kaufman, K., Keller, S., Kononenko, I., Kreuziger, J., Michalski, R., Mitchell, T., Pachowicz, P., Reich, Y., Vafaie, H., Van de Welde, W., Wenzel, W., Wnek, J., & Zhang, J. (1991). *The MONK's Problem: A Performance Comparison of Different Learning Algorithms.* (Technical Report CMU-CS-91-197), Pittsburgh, PA: Carnegie Mellon.

Towell, G. G. (1991). *Symbolic Knowledge and Neural Networks: Insertion, Refinement, and Extraction.* PhD thesis, Madison, WI: Computer Sciences Department, University of Wisconsin. (Available as Technical Report 1072).

Towell, G. G. & Shavlik, J. W. (1991). Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules. *Advances in Neural Information Processing Systems* (Vol. 4) (pp. 977–984). Denver, CO: Morgan Kaufmann.

Towell, G. G., Shavlik, J. W., & Noordewier, M. O. (1990). Refinement of approximately correct domain theories by knowledge-based neural networks. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 861–866). Boston, MA: AAAI Press.

Weiss, S. M. & Kulikowski, C. A. (1990). *Computer Systems that Learn.* San Mateo, CA: Morgan Kaufmann.

List of Figures

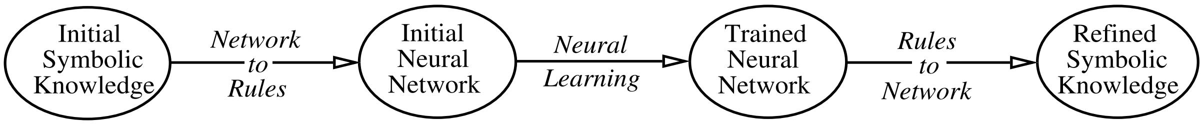


Figure 1: Rule refinement using KBANN.

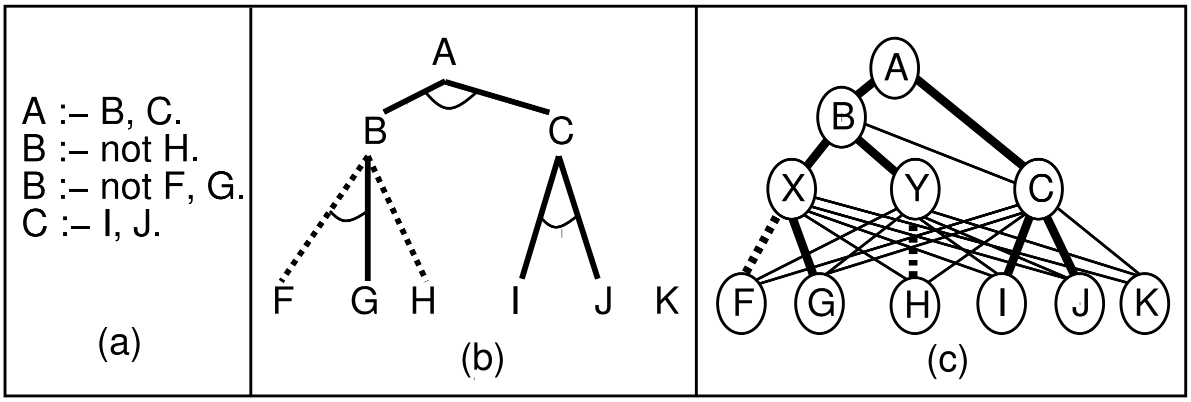
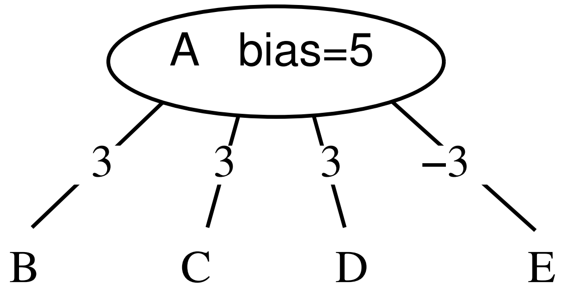


Figure 2: Translation of a knowledge base into a KNN.



(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 3: Rule extraction using the SUBSET algorithm.

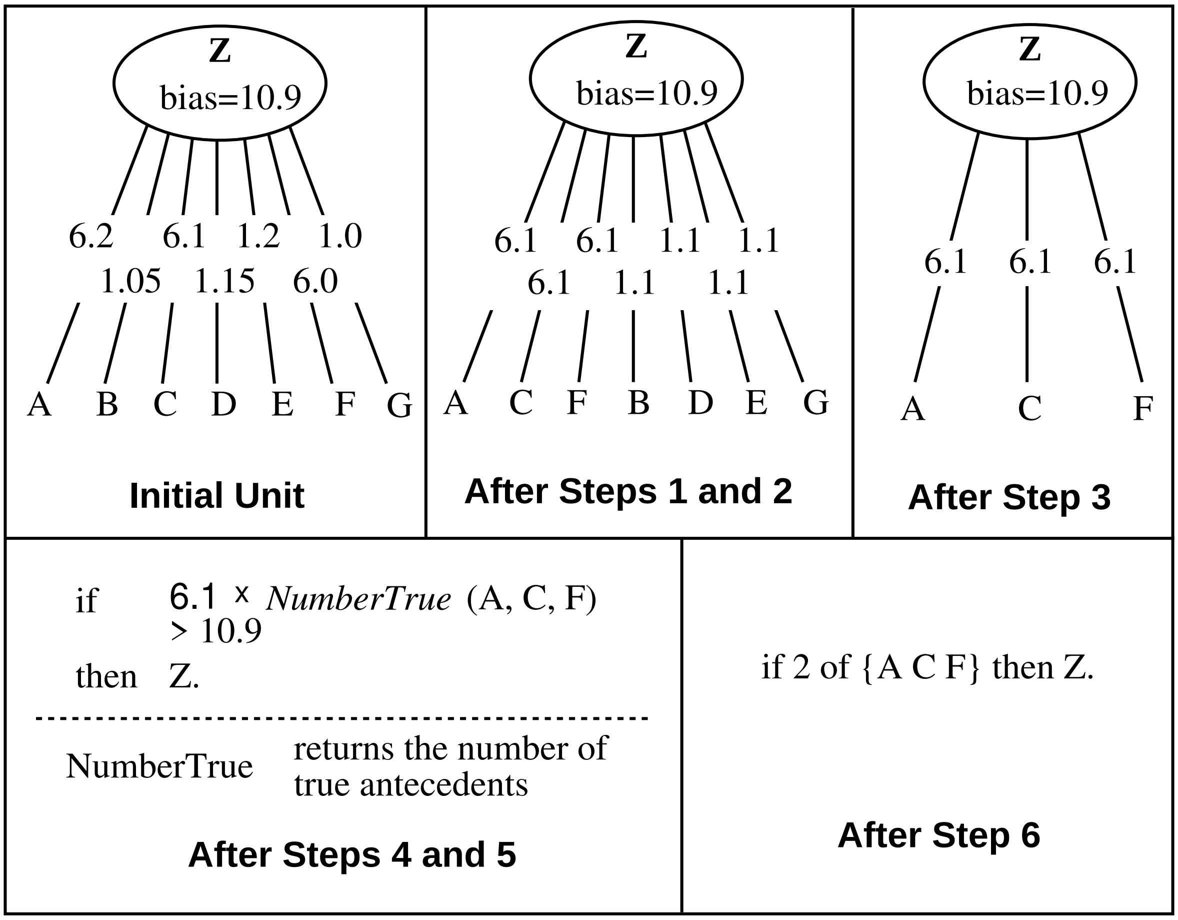


Figure 4: Rule extraction using the MofN algorithm.

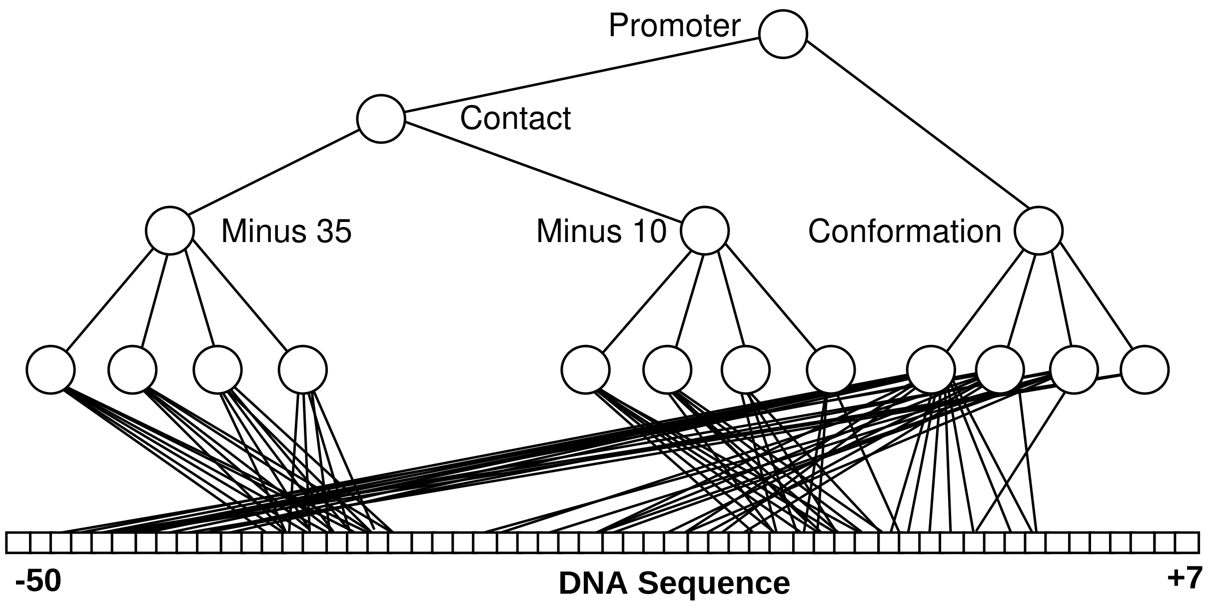


Figure 5: The initial KNN for promoter recognition (low-weighted links are not shown).

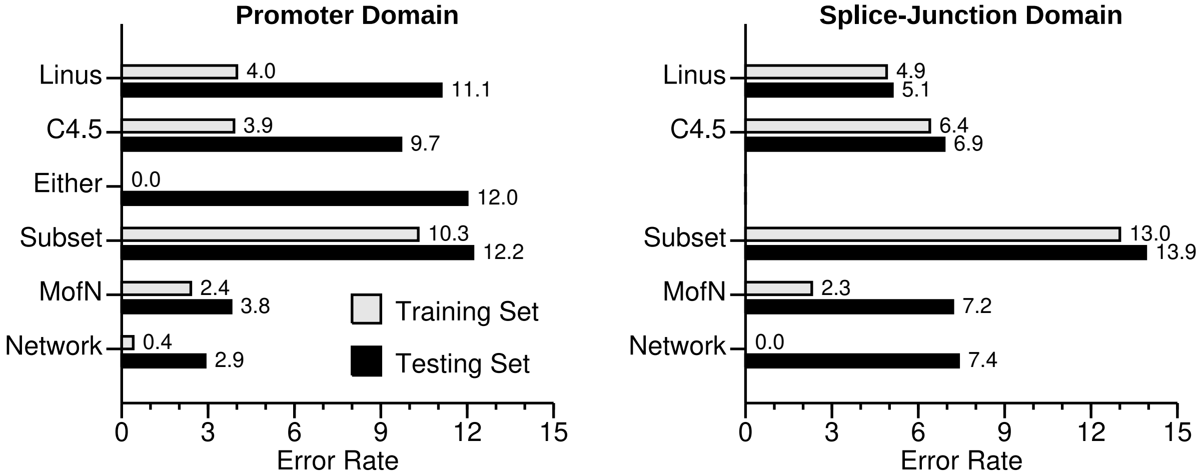


Figure 6: Error rates of extracted rules.

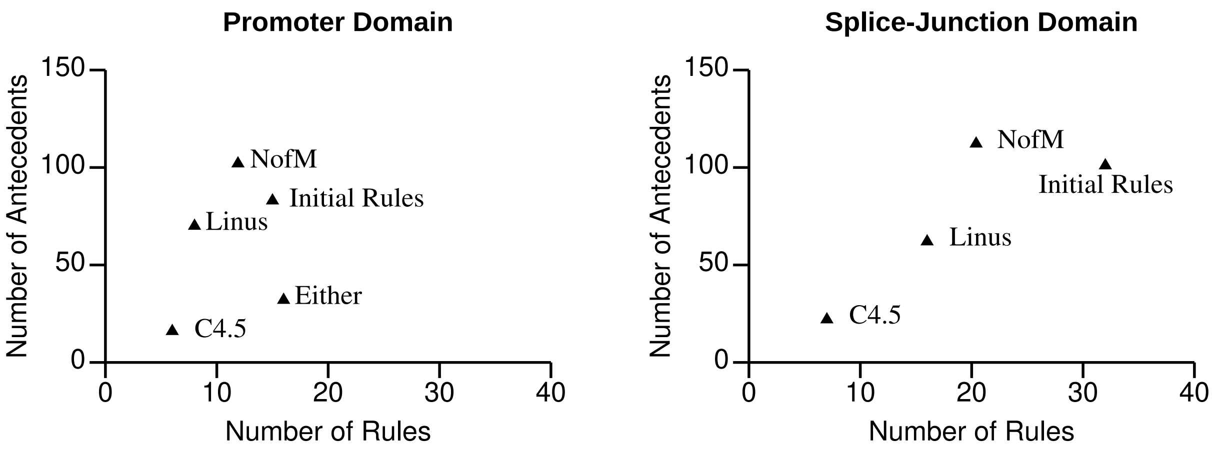


Figure 7: Comprehensibility of extracted rules. The rule sets extracted by SUBSET are too large to fit in the charted space. For the promoter domain, SUBSET extracts 322 rules with 1582 antecedents, while in the splice-junction domain it extracts 364 rules with 1072 antecedents.

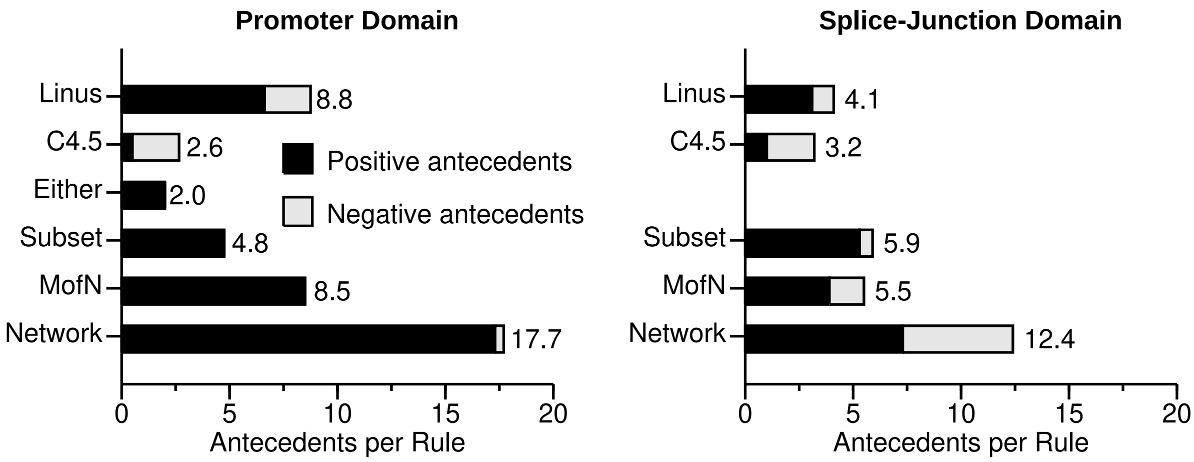


Figure 8: Average number of antecedents per rule.

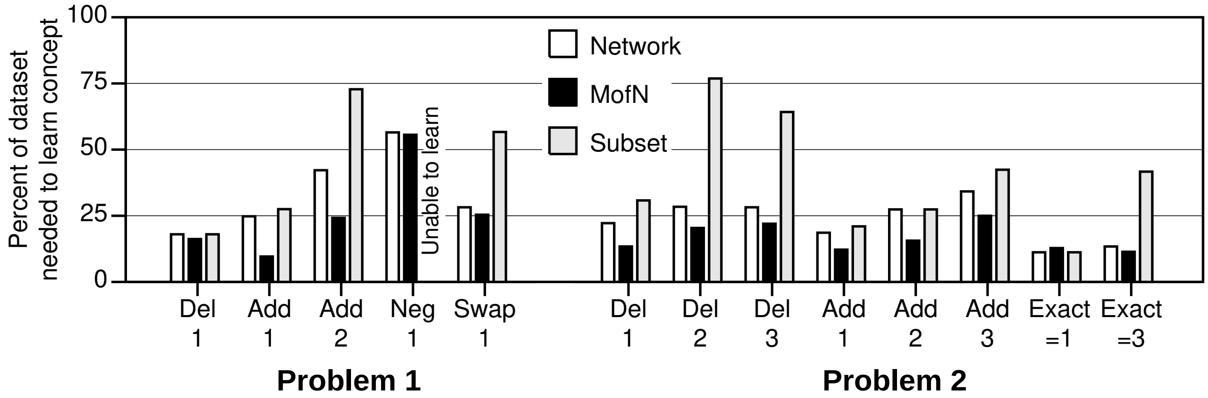


Figure 9: Percentage of the population required to learn the MONKS problems following corruption of the domain theory in various ways.

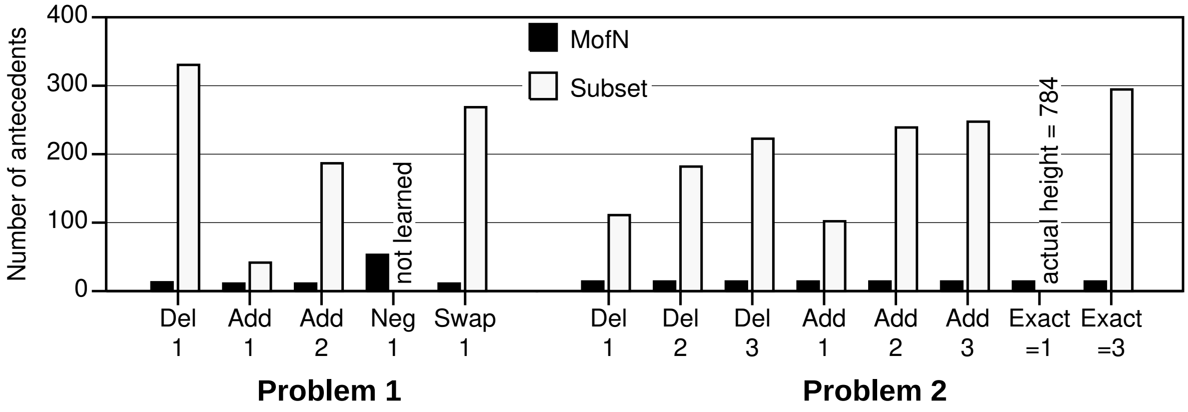


Figure 10: Average size (in number of antecedents) of rule sets that perfectly classify the entire population. The rule sets are extracted from trained KNNs initially derived from corrupted domain theories.