Machine Learning Research Group Working Paper 91-4

The Extraction of Refined Rules from Knowledge-Based Neural Networks

Geoffrey G. Towell **Jude W. Shavlik**

University of Wisconsin — Madison
1210 West Dayton Street
Madison, Wisconsin 53706

(608) 262-6613
{towell, shavlik}@cs.wisc.edu

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 in some form must be inserted into a neural network. Second, the network must be refined. Third, 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. This 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 (and can even exceed) 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, the method demonstrates that neural networks can be an effective tool for the refinement of 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: integrated learning,
neural networks,
theory refinement,
representational shift,
rule extraction from neural networks

Running Head: Extracting Rules

Submitted (8/91) to *Machine Learning*

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; Weiss & Kapouleas, 1989; Ng & Lippmann, 1990). However, ANNs have several well-known shortcomings; perhaps the most significant is that a trained ANN is essentially a "black box." That is, determining exactly why an ANN makes a particular decision is all but impossible. Yet, without such an ability, neural networks cannot be used as the basis of a rule refinement system. Furthermore, it is hard to be confident in the reliability of a network that addresses a real-world problem. Also, the fruits of training neural networks are difficult to transfer to other neural networks (Pratt & Kamm, 1991), much less non-neural learning systems. Hence, a trained neural network is something like Pierre de Fermat and his (in)famous last theorem. Like Fermat, the network tells you that it is has discovered something "wonderful", but then does not tell you exactly what that something is.

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

The first link of the 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 allowing refinements to be made at a finer grain.

The second link of the chain is to train the KNN using a set of classified training examples and standard neural learning methods. 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 their initial comprehensibility. Taking advantage of properties of trained KNNs that step from their initial comprehensibility, we developed a method to efficiently extract rules from networks. Significantly, when evaluated in terms of the ability to correctly classify examples not seen during training, the method produces sets of rules that are superior to the networks from which they came. Moreover, the extracted rules are 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 the most widely-published algorithm for the extraction of rules from general neural networks.

The next section contains a brief overview of KBANN. Subsequent sections describe both our method, and the best reported method for the extraction of rules from trained KNNs. The succeeding section presents a series of empirical tests, using two real-world learning problems taken from molecular biology, that characterize the strengths and weaknesses of both rule-extraction method. The final sections of this paper put this work into the context of other approaches to understanding trained neural networks and discuss our future research plans.

2 The KBANN Algorithm

The KBANN algorithm 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 in the form of PROLOG-like clauses, to define what is initially known about a topic. A detailed explanation of the procedure used by KBANN to translate rules into a KNN can be found in (Towell et al., 1990); we provide a brief summary here.

As an example of the KBANN method, consider the sample knowledge base in Figure 1a, which defines membership in category A. Figure 1b represents the hierarchical structure of these rules: solid and dotted lines represent necessary and prohibitory dependencies, respectively. Figure 1c represents the KNN that results from the translation into a neural network of this knowledge base. Units X and Y in Figure 1c are introduced into the KNN to handle the disjunction in the rule set. Otherwise, each unit in the KNN corresponds to a consequent or an antecedent in the knowledge base. The thick lines in Figure 1c represent heavily-weighted links in the KNN that correspond to dependencies in the knowledge base. The thin lines represent the links added to the network to allow refinement of the knowledge base.

This example illustrates that the use of KBANN 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, thereby guiding the choice of the number and connectivity of *hidden units* in the KNN.

KBANN-DAID (Towell & Shavlik, 1990) is a preprocessing phase applied to KNNs

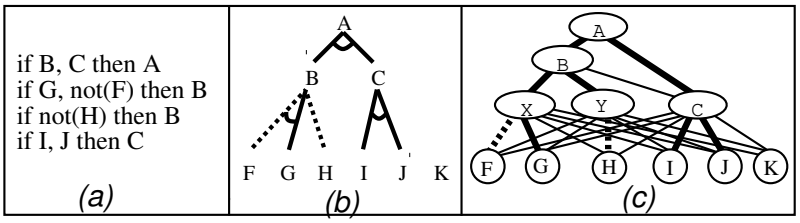


Figure 1: Translation of a knowledge base into a KNN.

to overcome some of the difficulties that KNNs present to neural-learning algorithms. Applying KBANN-DAID prior to submitting a KNN to backpropagation has been shown to both improve generalization and reduce the amount of training required by KNNs (Towell & Shavlik, 1990). Therefore, this training procedure is used for all KNNs from which we will extract rules.

3 Rule Extraction

This section presents detailed explanations of both our approach and the best reported approach to the extraction of rules from trained KNNs. Both methods make two assumptions about trained networks. The first assumption is that training does not significantly shift the meaning of units. By making this assumption, 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 the rules. If meaning shifted significantly, the utility of the labels would be minimal. Observation of trained networks indicates that the meanings usually are quite stable.[1]

The second assumption underlying the rule extraction methods is that almost all of the units in a trained KNN are either fully active (i.e., have activation near one) or inactive (i.e., have activation near zero). By making this assumption, each non-input unit in a trained KNN can be treated as a step function or a Boolean rule. Therefore, the problem for rule extraction is to determine the situations in which the "rule" is true. Again, examination of trained KNNs suggests that this assumption is valid. (Note that this second assumption is not particularly restrictive because the standard logistic activation function used in neural networks can be slightly modified to ensure that units approximate step functions.)

[1]Meanings occasionally shift. Such shifts are most common at units associated with the least certain of the initial rules. Hence, extracted rules that are inappropriately labeled can be used by humans as pointers to weak portions of the initial knowledge base.

3.1 Underpinnings of rule extraction

The method we describe for extracting rules from neural networks, as well as those previously in the literature, do so by trying to find combinations of the input values to a unit that result in it having an activation near one. Units in neural networks normally have activations defined by the functions:

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

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

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.

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 look for ways in which the weighted sum is guaranteed to exceed the bias.*

The second of the above assumptions, that all units in trained networks have activations near zero or one, simplifies this task by forcing links to either carry a signal equal to their weight or no signal at all. That is, Equation 1 reduces to:

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

So, rule extraction methods need only be concerned with the weight of links entering a unit and may ignore the precise activation of the sending unit. Therefore, the task of rule extraction reduces to searching each unit for sets of incoming links whose summed weights exceeds the bias.

Rule extraction is further simplified by the logistic activation function (Equation 2) which guarantees that units always have non-negative activations. This property allows rule-extraction methods to take the sign of a link's weight as a perfect indicator of the way in which an antecedent will be used. 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).

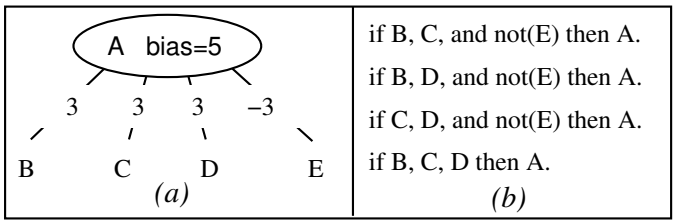


Figure 2: Rule extraction using the SUBSET algorithm.

Finally, note that without merely 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 form 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.

3.2 The SUBSET method

The method for rule extraction we describe first is fundamentally similar to the approach described by Saito and Nakano (1988), as well as Fu (1991). (The particular implementation is our own.) We call this procedure the SUBSET algorithm because it operates by attempting to find subsets of incoming weights that exceed the bias on a unit. While SUBSET is basically a "straw man" that shows the effectiveness of our new method for rule extraction, it represents the state of the art in the published literature.

As an example of the SUBSET algorithm, consider the unit in Figure 2a. Given that the link weights and the bias are as shown, then the algorithm would extract the four rules listed in Figure 2b to replicate the behavior of the unit (under the assumption that all units have activations near zero or one).

The major problem with the SUBSET method is that the cost of finding all subsets grows as the size of the power set. Thus, the algorithm can only be expected to work on simple networks and toy domains. To sidestep these combinatorial problems, Saito and Nakano (1988) establish an *ad hoc* ceiling on the number of consequents 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

Table 1: Specification of the SUBSET algorithm.

With each hidden and output unit:

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

With each element $\mathcal{P}$ of the β_p subsets:

Extract up to β_n subsets of negatively-weighted links whose summed weights is greater than the the sum of $\mathcal{P}$ less the bias on the unit.

With each element $\mathcal{N}$ of the β_n subsets form a rule: "if $\mathcal{P}$ and not $\mathcal{N}$ then <name of unit>".

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 of the SUBSET method looks for subsets using a branch-and-bound algorithm that is limited in terms of the number of rules it may find. The algorithm for the SUBSET method is presented in pseudocode in Table 1.

SUBSET may discover up to $\beta_p * \beta_n$ rules for each non-input unit in the network to be translated. Setting β_p and β_n to find acceptably small sets of rules that reasonably approximate the network from which they came, the SUBSET algorithm finds about 300 rules in each of the domains we studied. While large, 300 is smaller than than many handcrafted expert systems (Fozzard et al., 1988; McDermott, 1982). Hence, SUBSET delivers sets of rules that are, at least potentially, tractable. However, the rules tend to hide significant structures in trained networks. For instance, in Figure 2a the links to B, D 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 2b 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.
```

This consideration of structure shared among several rules led to the development of the algorithm that is the focus of this paper.

Table 2: The NOFM algorithm for 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 links 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 remaining links.

(6) Where possible, simplify rules to eliminate spperfluous weights and thresholds.

3.3 The NofM method

Our algorithm, called NOFM, addresses both the combinatorial and presentation problems inherent to the SUBSET algorithm. It differs from the SUBSET algorithm in that it explicitly searches for rules of the form:

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

As suggested previously, this method arose because we noticed that rule sets discovered by the SUBSET method often contain N-of-M style concepts. Further support for this method comes from experiments that indicate neural networks are good at learning N-of-M concepts (Fisher & McKusick, 1989) as well as experiments with a variant of ID3 that show a bias towards N-of-M 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 $N = 1$; hence, using N-of-M rules does not restrict generality.

The idea underlying our NOFM algorithm is that individual antecedents (links) do not have 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. Unfortunately, training using backpropagation does not naturally bunch links into equivalence classes; rather link weights are distributed across a range of real numbers. Hence the first step of the NOFM algorithm, summarized in Table 2, groups links into equivalence classes using a standard clustering method (Hartigan, 1975). This equivalence-class idea is the key to the NOFM algorithm; it allows the algorithm to consider groups of links without worrying about particular links within the group.

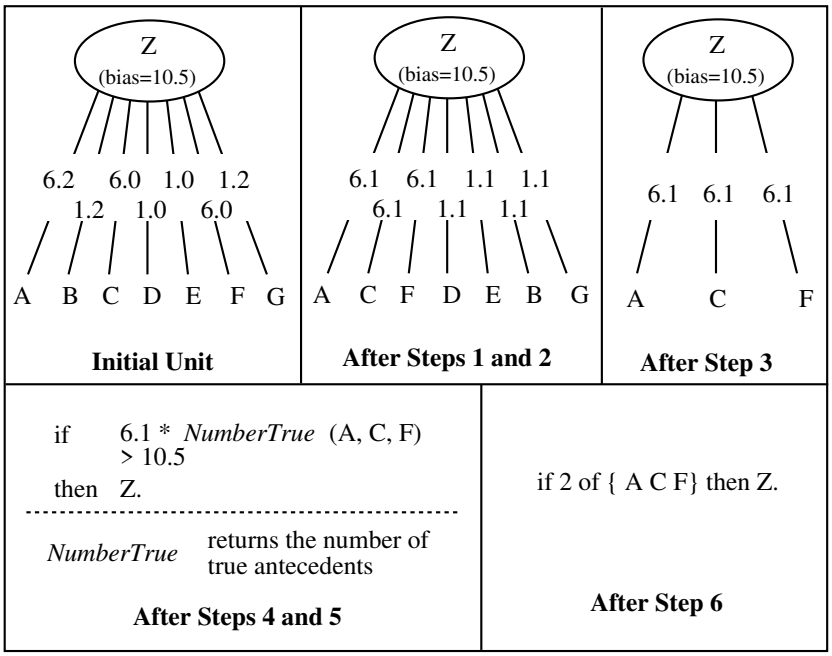


Figure 3: An example of rule extraction using NofM.

Once the groups are formed, the procedure next attempts to identify and eliminate those groups that cannot have any bearing on the calculation of the consequent. Such groups generally have low link weights and few members. With unimportant groups eliminated, the next step of the procedure is to optimize the bias on the unit. This can be done by freezing the weights on the links so that the groups stay intact and retraining the network using backpropagation with the activation function slightly modified to reflect the rule-like nature of the network; this training will only change the biases of the units.

After optimization, rules are formed that simply re-express the network. Note that these rules are considerable simpler than the original network; they have fewer antecedents and those antecedents tend to be in a few weight classes. Finally, an attempt is made to simplify the rules, eliminating the weights and the biases. Frequently such simplification results in N-of-M style rules.

As an example of the NOFM, consider Figure 3. This figure illustrates the process through which a single unit with seven incoming links is transformed by the NOFM procedure into a rule that requires two of three antecedents to be true.

The complexity of NOFM is difficult to precisely analyze as the bias-optimization phase uses backpropagation. However, 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.[2] Each of the other steps requires $O(n)$ time except for the initial clustering, which requires $O(n * log(n))$ time using an algorithm specialized for this task.

3.4 SUBSET and NOFM: Summary

Both of these rule-extraction algorithms have strengths with respect to the other. For example, the individual rules returned by SUBSET are more easily understood than those returned by NOFM. However, because SUBSET can be expected to return many more rules (which are often quite repetitive) than NOFM, the rule sets returned by NOFM may actually be easier to understand than those of SUBSET. More significantly, SUBSET is an exponential algorithm whereas NOFM is approximately cubic. Finally, results presented later in this paper indicate that the rule sets derived by NOFM equal or exceed the accuracy of the networks from which they are extracted while the rules extracted by SUBSET are significantly worse. The next section contains a series of empirical tests that demonstrate these and other differences between these two algorithms.

4 Experimental Datasets

To test the efficacy of our rule-extraction procedures, we studied two real-world problems from the domain of molecular biology. Both have been previously used to demonstrate the usefulness of the KBANN algorithm (Noordewier et al., 1991; Towell et al., 1990).

4.1 Notation

In both of these datasets, a special notation is used to simplify specifying locations in a DNA sequence (a linear string 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 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 symbol, or a sequence of symbols, 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. So, the first rule for `conformation` in Table 3 says that

[2] Training neural networks has been proven NP-complete (Blum & Rivest, 1988). However, Hinton (1989) suggests that backpropagation usually runs in $O(n^3)$ time.

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.

4.2 Promoter recognition

The first problem we investigate is *promoter recognition*, which 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 3 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 (O'Neill, 1989).

Table 3: 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'.
```

This rule set is translated by KBANN into a neural network with the topology shown in Figure 4. 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. 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

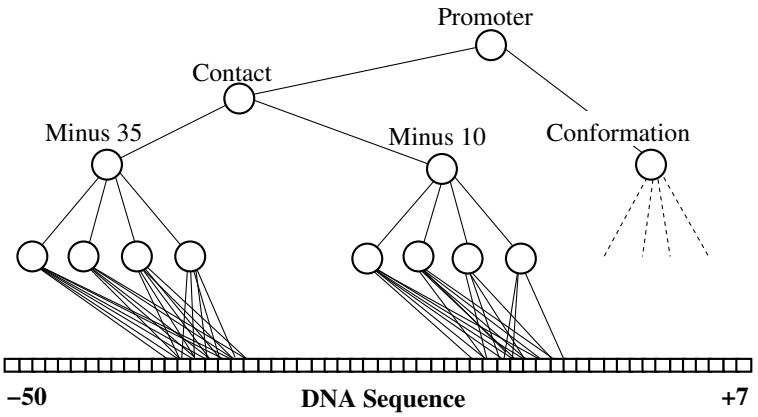


Figure 4: The initial KNN for promoter recognition.

preceding the reference point and 7 follow it. (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, 1989).

Prior to training, the rules in Table 3 do not classify any of the 106 examples as promoters. As a result, while the rules correctly classify 50% of the examples, they are only slightly more useful than a stopped clock.

4.3 Splice-junction determination

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* (the parts of the DNA sequence retained after splicing) and *introns* (the parts of the DNA sequence that are spliced out). This problem consists of two subtasks: recognizing exon/intron boundaries (referred to as E/I), and recognizing intron/exon boundaries (referred to as I/E). Figure 5 sketches how splicing occurs during the process of protein creation.

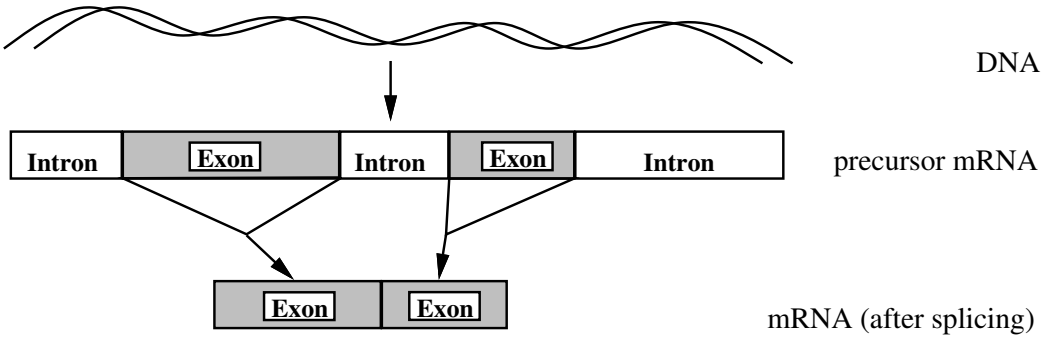


Figure 5: The expression of genes in higher organisms.

To approach this problem, we use a dataset initially described by Noordewier et al. (1991). This dataset contains 3190 examples, of which approximately 25% are *I/E*, 25% are *E/I* 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. We trained networks containing two output units: *E/I* and *I/E*. The third category (*neither*) was considered to true when neither *E/I* nor *I/E* exceeded a threshold (0.5).

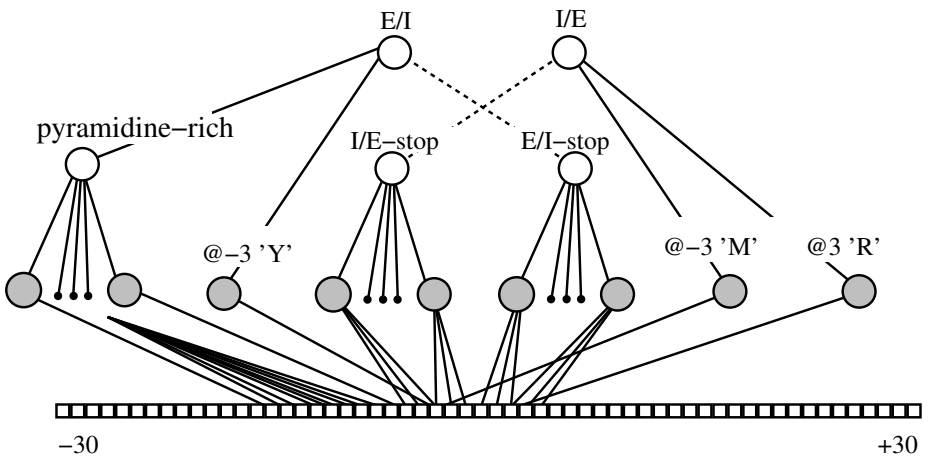
In addition to the examples, the information about splice-junctions includes a set of 21 rules appearing in Table 4. (This count does not include the rules defined by the iterative construct ``For i from ...'' which define the meaning of 'Y'.) This set of rules was derived in a straightforward fashion from the biological literature (Watson et al., 1987). Briefly, the splice-junction domain theory specifies sequences that appear near the two kinds of sites. The I/E boundary is also characterized by a *pyramidine-rich* region (i.e., a region where the majority of nucleotides are either C or T). Also, the rules check to make sure that the current location is not near the end of a gene by looking for certain sequences of letters that tell a cell to stop translating the gene into a protein. The "stop" rules disjunctively check for such indicators near the putative splice site.

The initial set of rules classifies 61% of the examples correctly. As for promoter recognition, the success rate of the splice junction rules initial 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. Figure 6 depicts an abstracted version of the network that KBANN generates given these rules. Note that this initial set of rules creates a KNN that can do little more than perceptron (Rosenblatt, 1962) style learning.

Table 4: Initial rules for splice-junction determination.

```
E/I :- @-3 'MAGGTRAGT', not(E/I-stop).

E/I-stop :- @-3 'TAA'.     E/I-stop :- @-4 'TAA'.     E/I-stop :- @-5 'TAA'.
E/I-stop :- @-3 'TAG'.     E/I-stop :- @-4 'TAG'.     E/I-stop :- @-5 'TAG'.
E/I-stop :- @-3 'TGA'.     E/I-stop :- @-4 'TGA'.     E/I-stop :- @-5 'TGA'.

I/E :- pyramidine-rich,    @-3 'YAGG', not(I/E-stop).

pyramidine-rich :- 6 of   (@-15 'Y', @-14 'Y', @-13 'Y', @-12 'Y', @-11 'Y',
                           @-10 'Y', @-09 'Y', @-08 'Y', @-07 'Y', @-06 'Y').

For i from -30 to +30     {@<i> 'Y' :- @<i> 'C'.  @<i> 'Y' :- @<i> 'T'.}

I/E-stop :- @1 'TAA'.      I/E-stop :- @2 'TAA'.      I/E-stop :- @3 'TAA'.
I/E-stop :- @1 'TAG'.      I/E-stop :- @2 'TAG'.      I/E-stop :- @3 'TAG'.
I/E-stop :- @1 'TGA'.      I/E-stop :- @2 'TGA'.      I/E-stop :- @3 'TGA'.
```

See appendix for meanings of letters other than A, G, T, C.



Shaded units are definitional features. They are "fixed" in that neither the weights of their incoming links nor their bias is allowed to change during training.

Figure 6: The initial splice-junction KNN.

5 Experiments in the Extraction of Rules

This section presents a set of experiments designed to determine the relative strengths and weaknesses of the two rule-extraction methods described above. 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 NOFM.

5.1 Testing Methodology

Following Weiss and Kulikowski (1990), networks are trained using repeated 10-fold cross-validation[3] for assessing the quality of learning in both domains. We train networks until all training examples are correctly classified or every example has been presented 500 times. Following Hinton's (1989) suggestion for improved network interpretability, all weights are subject to gentle "decay" during training.[4] Finally, 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). This methodology is used, except when noted, in all our experiments.

5.2 Rule 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 KNNs provide 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 (e.g., Ourston & Mooney 1990; Thompson et al. 1991) to develop rules, while using KNNs 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 determining that a trained KNN is likely to behave correctly in the future. On each of these measures, we show that the NOFM method is

[3] In N-fold cross-validation, the set of examples is partitioned into N sets of equal size. Networks are trained using $N-1$ of the sets and tested using the remaining set. This procedure is repeated N times so that each set is used as the testing set once.

[4] 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 currently use a very gentle $\phi = 0.99999$.

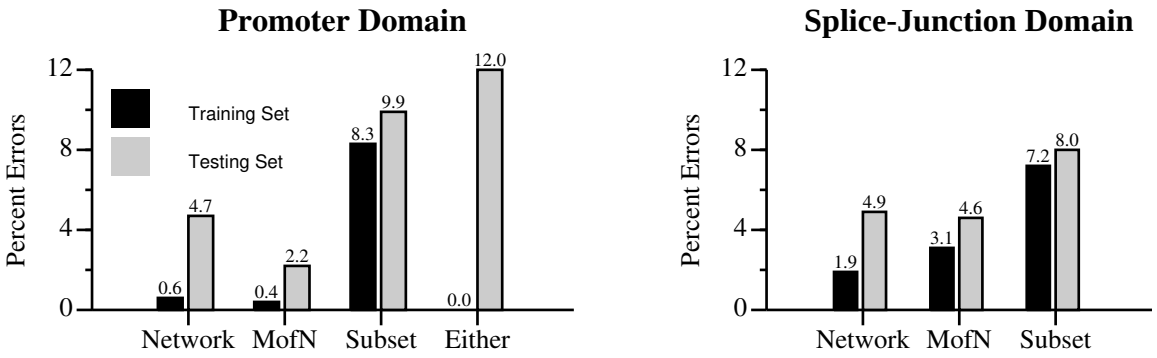


Figure 7: Error rates of extracted rules.

superior to the SUBSET method. More significantly, we show that the the NOFM method extracts rules that are superior to the networks at classifying testing examples.

5.2.1 Accuracy.

Figure 7 addresses the issue of the accuracy of extracted rules. It plots percentage of errors on the testing and training sets, averaged over ten repetitions of 10-fold cross-validation, for both the promoter and splice-junction tasks. For comparison, Figure 7 includes the accuracy of the trained KNNs prior to rule extraction (the bars labeled "Network"). Also included in Figure 7 is the accuracy of the EITHER system, an "all symbolic" method for the empirical adaptation of rules which has been tested using the promoter dataset (Ourston & Mooney, 1990). The numbers for EITHER were supplied by the authors of that system (Mooney & Ourston, 1991); they reflect a slightly different testing method. Labyrinth, (Thompson et al., 1991) another system for the symbolic refinement of rules, has also been tested on the promoter problem. Its results are superior to those of EITHER but inferior to KBANN. (Due to a different testing methodology, results for labyrinth do not appear in Figure 7). Neither EITHER nor labyrinth have been tested on the splice-junction problem.

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 7 improves upon the initial rules. Comparing only the systems that produce refined rules, the NOFM method is the clear winner. On the training examples, the error rate for rules extracted by NOFM is slightly worse than EITHER, but superior to the rules extracted using SUBSET . On the testing examples the NOFM rules are much more accurate than both EITHER and SUBSET. (One-tailed, paired-sample

t-tests indicate that for both domains the NOFM rules are superior to the SUBSET rules with 99.5% confidence.)

Perhaps the most significant result in this paper is that, on the testing examples, the error rate of the NOFM rules is superior to that of the networks from which the rules were extracted. (One-tailed, paired-sample t-tests indicate that the difference is statistically significant at the 0.005 level for the promoter domain (t=5.5, d.f.=9) and at the 0.100 level for the splice-junction domain (t=1.6, d.f.=9).) Conversely, the error rate of the SUBSET rules on testing examples is statistically worse than the networks in both problem domains. The discussion at the end of this section analyses the reasons why the NOFM rules are superior to the networks from which they are extracted.

5.2.2 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 5 indicates that NOFM is superior to SUBSET at reproducing the behavior of the network on the examples seen during training. For instance, on the splice-junction data, rules extracted using NOFM 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 5 shows that both methods of rule extraction are much better at mimicking the behavior of the network when the network generates the correct answer, than when it generates an incorrect answer. Note that the relatively poor fidelity of the NOFM rules when the networks generate an incorrect answer is due to the superior classification performance of the NOFM rules. That is, the NOFM rules often fail to repeat the errors of the networks. The fidelity of the extracted rules to the networks on examples not seen during training (data not shown) is similar.

5.3 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 look at individual rules and try to determine if they are meaningful. The following sections present results on both of these measures.

Table 5: 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
NOFM	93%	0.95	0.31
Promoter			
SUBSET	87%	0.90	0.24
NOFM	99%	0.99	0.10

5.3.1 Global Comprehensibility.

We characterize the comprehensibility of sets of rules using three measures: *size*, *number of antecedents per rule* and, *consistency*.

Size. Certainly size is a major concern in determining whether or not a rule set is comprehensible, for sets with large number of rules can be difficult, if not impossible, to understand. Figure 8 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. Counting rules and antecedents for the rules extracted by NOFM is less than straightforward, as simplification may increase the number of rules and require the reuse of some antecedents. The data in both Figures 8 and 9 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. Also included in Figure 8 are the initial set of rules in each domain and the set of useful rules produced by EITHER after training on all 106 promoter examples (Mooney & Ourston, 1991).

Taking the initial rule sets in each domain as a standard for interpretability, the rules refined by EITHER are likely to be easy to understand. (Recall, however, that these rules are not particularly accurate.) Rules extracted by NOFM are also likely to be easily understood; they have virtually the same number of rules and antecedents as the initial rule sets. On the other hand, the rules SUBSET extracts are much less likely to be comprehensible. (See Tables 7 and 8 for examples of rules extracted by NOFM and

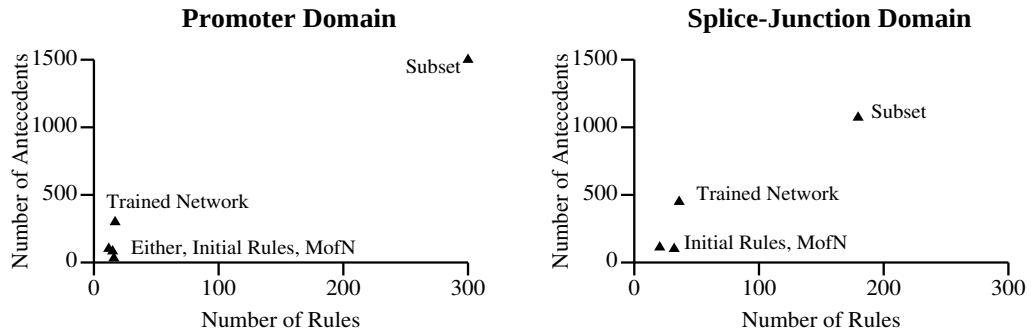


Figure 8: Comprehensibility of extracted rules.

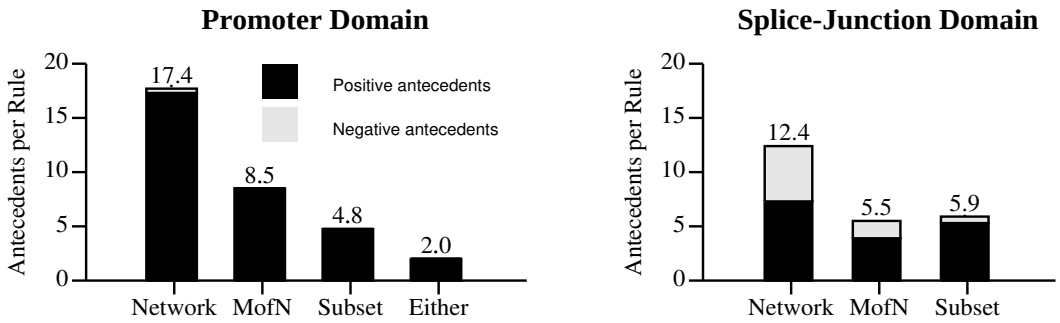


Figure 9: Average number of antecedents per rule.

SUBSET, respectively.)

Antecedents per rule. A second important global statistic is the number of antecedents per rule. If this value is large, individual rules are unlikely to be understandable (Brunner 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 NOFM rules and which implicitly appear in trained networks, cloud the picture further. Thus, in Figure 9, we take a very simple approach to judging rules on the basis of antecedents. Each bar consists of a stack that represents the simple sum of the negative and positive antecedents to an average rule.

On this measure, as with rule-set size, EITHER is the clear winner. The NOFM 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 that NOFM extracts many fewer rules).

Table 6: Number of classification differences among the rules extracted from three trained networks.

Problem	NOFM	SUBSET
Promoter	1	3
Splice-Junction	76	193

Consistency. Finally, the "consistency" of rules is important because if the extracted rules vary considerably between training sessions, it is difficult to ascribe any significance to a particular rule set. To test this issue, we abandon ten-fold cross-validation and instead train three networks using the complete dataset. Hence, the only differences between the trained networks are (a) the initial starting configurations of the networks (which vary only by small random perturbations to the network) and (b) the order in which examples are presented.

There are several ways to determine the consistency of two or more sets of rules. Arguably the best way, and certainly the most thorough, is to test each rule set with every possible input and compare the actions of each set. However, this approach is impossible for it would require testing about 10^{34} combinations for the promoter problem and about 10^{36} combinations for the splice-junction problem.[5]

Because of these combinatoric problems, we measure consistency by training three networks, differing only by slight randomness in initial link weights, with all of the examples. We then extract rules form those networks, test the extracted rules against the examples, and count the number of examples for which the three sets of extracted rules do not agree. Note that this measure is blind to whether or not the rules are correct, it is concerned only with whether the rules make the same decision. The results of this test appear in Table 6. On both domains, NOFM extracts rules that are considerably more consistent than SUBSET, averaging about one inconsistency for every three made by the SUBSET rules.

A slightly different approach to determining consistency looks at the errors made during repeated 10-fold cross-validation runs. Specifically, if the rule-extraction method is consistently pulling out the same information, then the extracted rules should make the same number of errors on each cross-validation run. Hence, a consistent method should have a low standard deviation in the number of errors. This is exactly what is

[5]Let $\Omega = 15 * 10^9$years (approximately the age of the universe). Then, assuming that you could evaluate one million combinations per second, it would take $2.8 * 10^{12}$ Ω to evaluate all the possible splice-junction combinations.

observed for the NOFM method; its standard deviation is lower than both SUBSET and the trained KNNs.

5.3.2 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 looked at rule sets extracted by the NOFM and SUBSET methods after training networks on the whole set of examples from each problem domain. Tables 7 and 8 present the rules NOFM and SUBSET extract from the same network for promoter recognition. The rules extracted from the splice-junction domain have much the same character as those of the promoter domain; hence, they are included only as an appendix. Because rule sets extracted by SUBSET can be very large, we set $\beta_p = 5$ and $\beta_n = 1$ for these tables. These values are too small to deliver reasonably accurate rule sets; they each make about 25% more errors than the networks from which they are extracted.

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

One major pattern is apparent in the rules extracted by both NOFM and SUBSET. Specifically, the network learned to disregard `conformation`. The `conformation` rules are dropped by other rule-refinement systems (e.g., Mooney & Ourston 1991), which suggests that dropping these rules is not an artifact of our method but rather that DNA bases 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 NOFM 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 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 even more difficult to get a clear picture of a promoter from the SUBSET rules

Table 7: Promoter rules NOFM extracts.

```
Promoter :- Minus35, Minus10.

Minus-35
 :- 10 < 4.0 * nt(@-37 '--TTGAT-') +
         1.5 * nt(@-37 '----TCC-') +
         0.5 * nt(@-37 '---MC---') -
         1.5 * nt(@-37 '--GRAGG-').
Minus-35
 :- 10 < 2.4 * nt(@-37 '--TTG-CA') +
         1.2 * nt(@-37 '----W---').
Minus-35
 :- 10 < 5.0 * nt(@-37 '--T-G--A') +
         3.1 * nt(@-37 '---GT---') +
         1.9 * nt(@-37 '----C-CT') +
         1.5 * nt(@-37 '---C--A-') -
         1.5 * nt(@-37 '------GA') -
         1.9 * nt(@-37 '--CGW---') -
         3.1 * nt(@-37 '--A----C').
Minus-35
 :- 10 < 2.5 * nt(@-37 '-C-TGAC-') +
         0.7 * nt(@-37 '--H-TT--') -
         0.7 * nt(@-37 '---C----').

Minus-10
 :- 10 < 5.0 * nt(@-14 '---CA---T') +
         1.8 * nt(@-14 '---T----A') -
         1.8 * nt(@-14 '---RB---S').
Minus-10
 :- 10 < 3.0 * nt(@-14 '--TAT--T-') +
         1.8 * nt(@-14 '-----GA--') +
         0.7 * nt(@-14 '----GAT--') -
         0.7 * nt(@-14 '--GKCCCS-').
Minus-10
 :- 10 < 3.8 * nt(@-14 '--TA-A-T-') +
         3.0 * nt(@-14 '--G--C---') +
         1.0 * nt(@-14 '---T---A-') -
         1.0 * nt(@-14 '--CS-G-S-') -
         3.0 * nt(@-14 '--A--T---').
Minus-10
 :- 10 < 2.2 * nt(@-14 '-TA-A-T--') +
         1.5 * nt(@-14 '---W-----') +
         0.8 * nt(@-14 '-G---G---') -
         0.8 * nt(@-14 '-CBGKT---') -
         1.5 * nt(@-14 '------A--') -
         2.2 * nt(@-14 '---C--G--').
```

See appendix 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'.

(Table 8). The first three *minus35* encode a simple 2-of-3 concept while the *minus10* rules seem to approximate a 3-of-7 concept. Note that these patterns support the idea implemented in the NOFM method that a bias towards N-of-M style concepts is useful. Other than these patterns, it is difficult to say anything about the SUBSET rules.

5.4 Discussion

Our results indicate that upon any of the measures defined above, the NOFM method is able to extract a good set of rules from trained networks. Specifically, with the NOFM

Table 8: Promoter rules extracted by SUBSET.

```
Promoter :- Minus35, Minus10.

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

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-'

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--'.
```

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.

method not only can meaningful, symbolic rules be extracted from trained KNNs, the extracted rules can be superior to the KNN at classifying testing examples. Additionally, the NOFM method produces refined rule sets whose accuracy is substantially better than EITHER, an approach that directly modifies the initial set of rules (Ourston & Mooney, 1990; see Figure 7). While the rule set produced by the NOFM algorithm is

slightly larger than that produced by the "all symbolic" approach of EITHER, the sets of rules produced by both of these algorithms is small enough to be easily understood. Hence, although weighing the tradeoff between accuracy and understandability is problem and user-specific, the NOFM approach combined with the KBANN algorithm offers an appealing mixture.

Two hypotheses explain the superiority of NOFM over rule-refinement method 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. Hence, it may be possible to more closely fit the target concept than when taking the large steps required by direct rule-refinement. Second, DNA sequence-analysis problems have aspects that can be perspicuously captured by N-of-M style rules. 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. As neither EITHER nor Labyrinth (nor the SUBSET method) can easily express N-of-M rules, these algorithms may be at a disadvantage on our test domains.

The superiority of the NOFM rules over the networks from which they are extracted occurs because the rule-extraction process reduces overfitting of the training examples. The principle evidence in support of this hypothesis is that the difference in ability to correctly categorize testing and training examples is smaller for NOFM rules than for trained KNN. In other words, the rules that the NOFM method extracts are only slightly better at classifying training examples than at classifying testing examples. (While these differences are smaller, they are statistically significant at the 0.005 level using a one-tailed, paired-sample, t-test.) The rules SUBSET extracts also have this property, but they are worse on the training set than both NOFM rules and trained KNNs on the testing set.

Another piece of evidence in support of the overfitting hypothesis comes from the work in pruning of neural networks to eliminate superfluous parts (Mozer & Smolensky, 1988; Le Cun et al., 1989; Weigand et al., 1990). Rule extraction is, in some sense, 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 the reseachers attribute these gains to reduced overfitting of the training examples.

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). These methods avoid having to extract rules after learning by performing symbolic manipulations directly on the rules. For instance, Ourston and Mooney's EITHER (1990) operates by identifying examples incorrectly classified by the initial rule set and attempting to find a way of altering the rule set so that the previously-incorrect example is corrected, without creating new errors among the correct examples. While these methods are appealing because they do not require a shift between symbolic and neural representations, the results presented in Figure 7 indicate that they are not as accurate as the methods presented here.

The second set contains attempts to understand trained neural networks. One common approach is to attempt to present the information in a network in a compact, graphical form that enables a user to analyze what was learned. "Hinton diagrams" are the most common effort in this direction (Hinton, 1989). Presenting a reasonably complex network in human-comprehensible form requires either a considerably simplified network (Hinton, 1989) or interactive methods for presenting only small portions of the network (Wejchert & Tesauro, 1989). For example, to make "Hinton diagrams" of reasonable size, networks may be configured with few or no hidden units (Tesauro & Sejnowski, 1989). A different approach is to use statistical clustering software to identify commonalities in the input/output mapping of ANNs (e.g., Sejnowski & Rosenberg 1987). This approach only appears to be useful for understanding the "high-level" decisions of the network.

A still different approach to understanding neural networks attempts to prune away parts of the networks that are unimportant to the calculations of the network. In so doing, the complexity of the problem can be significantly reduced, thereby making the network more easily understood. For instance, Le Cun et al. (1989) describe the *OBD* method that eliminates as much as 30% of the number of free parameters in a network without increasing error. Experiments with OBD on the promoter domain indicate that up to 90% of the links can be removed from KNNs without increasing error on this problem (Craven, 1990). (We performed similar experiments using a closely-related technique described by Weigand, Rumelhart and Huberman (1990) and obtained almost identical results.) Mozer and Smolensky (1988) a related technique; their "skeletonization" method eliminates whole units rather than just individual free parameters. While the authors of

these papers make claims that their procedures increase the human comprehensibility of networks, no evidence is presented to support the claim. Thus, while the rule-extraction techniques we describe are similar in spirit to these pruning systems, this paper differs significantly in its emphasis on comprehensible output.

Several researchers have reported 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. 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. Moreover, the method of both Saito and Nakano, and Fu find only rules that map directly form inputs to outputs.

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. When similarities are found, rules are created. As this method can also look for similarities within the links from the K additional units, it can discover that some of the outputs provide information about other outputs. Hence, the method can discover hierarchical rule sets. However, to discover the relationship (robin *isa* bird *isa* animal) the network must have outputs for robin, bird and animal. Other methods for the extraction of rules from specially-configured networks include McMillan, Mozer and Smolensky's *connectionist scientist game* (1991) which uses neural networks to iteratively learn sets of propositional rules.

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 the extraction of fuzzy rules from neural networks have also been reported (e.g., Hayashi 1990; Masuoka et al. 1990).

7 Limitations and Future Work

The results presented in Section 5 indicate that the NOFM method is able to effectively extract rules from trained KNNs, thereby making KBANN an excellent method for the refinement of existing rules. However, the 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* NOFM *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 NOFM method is effective.

- *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. Such a method might be based upon the "feature evaluation" system described by Matheus (1990).

- *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 above). In such cases is is necessary to augment the initial network with additional hidden units (Towell et al., 1991). However, adding hidden units opens many of the issues raised in the above discussion 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 previously, sequence analysis problems often contain aspects that are N-of-M-like. So, the NOFM method may be ideally suited to sequence-analysis problems. (Murphy and Pazzani (1991) suggest that a bias towards N-of-M concepts may be generally useful for real-world 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 NOFM 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 rounds each link weight into a member of a predetermined, small set of values. Training thus consists of alternating cycles of (a) standard weight adjustment and (b) rounding. The form of the rules produced by this method will be similar to NOFM rules. We expect that the search paths through weight space taken by this algorithm, however, will be quite different than the paths taken by the NOFM 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 rounding 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.

We are also investigating ways of enhancing the comprehensibility of the rules returned by NOFM. We believe that an N-of-M style representation is a useful inductive bias for many problems, both inside and outside the domain of DNA sequence analysis (Murphy & Pazzani, 1991). While the rules extracted from networks by NOFM 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 or some slight enhancements of the NOFM algorithm will improve the comprehensibility of the resulting rules.

8 Conclusions

This paper presents and empirically validates the NOFM method for the extraction of rules from knowledge-based neural networks. We have shown that the NOFM method, as a part of the KBANN system, forms a rule-refinement algorithm that results in rules which are better able to generalize to examples not seen during training than "all-symbolic" rule refinement algorithms (e.g., Ourston & Mooney 1990; Thompson et al. 1991). We attribute this generalization ability to the shift of representation of the initial domain theory from symbolic to neurally-based. This shift allows a learning system to make smaller increments towards the correct concept.

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 the

plethora of comparisons among empirical learning systems. However, domain-specific knowledge is much more difficult to shift between representations. Our initial KBANN paper provided a method for shifting domain-specific knowledge from PROLOG-like rules into neural networks (Towell et al., 1990). This shift made the knowledge available to neural learning algorithms such as backpropagation (Rumelhart et al., 1986). 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 results are a dead end, for the refined knowledge in the trained networks is inaccessible; the previous papers provide a method to shift 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 NOFM method can extract concise, comprehensible rules from a trained KNN without a serious loss of accuracy. In fact, the extracted rules are more accurate at classifying testing examples than 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 and Mark Craven are also gratefully acknowledged.

Both the promoter recognition and splice-junction determination datasets are a part of the collection of datasets at UC-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. In *Proceedings of the Eighth International Machine Learning Workshop*, (pp. 475–479), Evanston. IL.

Blum, A. & Rivest, R. L. (1988). Training a 3-node neural network is NP-complete. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, (pp. 9–18), Cambridge, MA.

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

Craven, M. W. (1990). Experiments in optimal brain damage. Unpublished manuscript.

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

Fozzard, R., Bradshaw, G., & Ceci, L. (1988). A connectionist expert system that actually works. In *Advances in Neural Information Processing Systems (volume 1)*, (pp. 248–255), Denver, CO. Morgan Kaufmann.

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

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

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. In *Advances in Neural Information Processing Systems (volume 3)*, (pp. 578–584), Denver, CO. Morgan Kaufmann.

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

IUB Nomenclature Committee (1985). Ambiguity codes. *European Journal of Biochemistry*, 150:1–5.

Koudelka, G. B., Harrison, S. C., & Ptashne, M. (1987). *Nature*, 326:886–888.

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

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

Matheus, C. J. (1990). *Feature Construction: An Analytic Framework and an Application to Decision Trees*. PhD thesis, University of Illinois at Urbana-Champaign.

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. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, Chicago, IL.

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.

Mooney, R. J. & Ourston, D. (1991). Personal communication.

Mozer, M. C. & Smolensky, P. (1988). Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in Neural Information Processing Systems (volume 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. In *Proceedings of the Eighth International Machine Learning Workshop*, (pp. 183–187), Evanston, IL.

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

Ng, K. & Lippmann, R. P. (1990). A comparative study of the practical characteristics of neural networks and conventional pattern classifiers. In *Advances in Neural Information Processing Systems (volume 3)*, (pp. 970–976), Denver, CO. Morgan Kaufmann.

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

O'Neill, M. C. (1989). *Escherichia coli* promoters: I. Consensus as it relates to spacing class, specificity, repeat substructure, and three dimensional orgainzation. *Journal of Biological Chemistry*, 264:5522–5530.

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

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

Record, T. (1989). Personal communication.

Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan, New York.

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

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

Sejnowski, T. J. & Rosenberg, C. (1987). Parallel networks that learn to pronounce English text. *Complex Systems*, 1:145–168.

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

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. In *Methods in Enzymology (volume 183)*. Academic Press.

Tesauro, G. & Sejnowski, T. J. (1989). A parallel network that learns to play backgammon. *Artificial Intelligence*, 39:357–390.

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

Towell, G. G. (1991). *Symbolic Knowledge and Neural Networks: Insertion, Refinement, and Extraction*. PhD thesis, University of Wisconsin, Madison, WI. In preparation.

Towell, G. G. & Shavlik, J. W. (1990). Directed propagation of training signals through knowledge-based neural networks. Technical Report 989, University of Wisconsin, Computer Sciences Department, Madison, WI.

Towell, G. G., Shavlik, J. W., & Craven, M. W. (1991). Constructive induction in knowledge-based neural networks. In *Proceedings of the Eighth International Machine Learning Workshop*, (pp. 213–217), Evanston, IL.

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

Watson, J. D., Hopkins, N. H., Roberts, J. W., Steitz, J. A., & Weiner, A. M. (1987). *Molecular Biology of the Gene*.

Weigand, A. S., Rumelhart, D. E., & Huberman, B. A. (1990). Generalization by weight-elimination with application to forecasting. In *Advances in Neural Information Processing Systems (volume 3)*, (pp. 875–882), Denver, CO. Morgan Kaufmann.

Weiss, S. M. & Kapouleas, I. (1989). An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, (pp. 688–693), Detroit, MI.

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

Wejchert, J. & Tesauro, G. (1989). Neural network visualization. In *Advances in Neural Information Processing Systems (volume 2)*, (pp. 465–472), Denver, CO. Morgan Kaufmann.

A Appendices

A.1 Ambiguity codes of DNA nucleotides

The following table gives the letter codes for amino acid codes and nucleotide ambiguity proposed by IUB (IUB Nomenclature Committee, 1985). Recall that the four nucleotides in DNA are A, G, T and C.

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 T or C

A.2 Extracted rules for splice-junction determination.

The splice-junction rules NOFM extracts.

```
I/E :- 10 < 2.8 * nt(@-06 'Y--YAG', pyramidine-rich) +
            1.5 * nt(@-10 'Y', I/E-stop1, E/I-stop7) +
           -1.5 * nt(I/E-stop3, E/I-stop5) +
           -2.8 * nt(@-12 'Y', I/E-stop4, E/I-stop4, E/I-stop6, E/I-stop) +
           -4.7 * nt(E/I-stop3) +
           -7.5 * nt(I/E-stop8).

pyramidine-rich :- @-14 '--YY----', 1 <= nt(@-14 'YY---Y-Y').
pyramidine-rich :- @-14 'YY-Y-Y-Y'.
pyramidine-rich :- @-14 '--Y-----', 2 <= nt(@-14 'YY---Y--').

I/E-stop  :- 1 <= nt(I/E-stop1, I/E-stop2, I/E-stop3, I/E-stop4, I/E-stop5,
                     I/E-stop6, I/E-stop9).

I/E-stop1 :- @1 'TAA'.      I/E-stop4 :- @2 'TAA'.      I/E-stop7 :- @3 'TAA'.
I/E-stop2 :- @1 'TAG'.      I/E-stop5 :- @2 'TAG'.      I/E-stop8 :- @3 'TAG'.
I/E-stop3 :- @1 'TGA'.      I/E-stop6 :- @2 'TGA'.      I/E-stop9 :- @3 'TGA'.

E/I  :- 10.0 < 4.2 * nt(@-02 '---T----') +
               2.2 * nt(@-02 '-G--R-G-') +
               1.2 * nt(@-02 'AA---M-T') +
               0.6 * nt(@-02 'G--A---C') +
              -0.6 * nt(@-02 '-C---T--') +
              -1.2 * nt(@-02 '---S----') +
              -2.2 * nt(E/I-stop).

E/I-stop :- 1 <= nt(E/I-stop1, E/I-stop2, E/I-stop5, E/I-stop8).

E/I-stop1 :- @-3 'TAA'.     E/I-stop4 :- @-4 'TAA'.     E/I-stop7 :- @-5 'TAA'.
E/I-stop2 :- @-3 'TAG'.     E/I-stop5 :- @-4 'TAG'.     E/I-stop8 :- @-5 'TAG'.
E/I-stop3 :- @-3 'TGA'.     E/I-stop6 :- @-4 'TGA'.     E/I-stop9 :- @-5 'TGA'.

For i from -30 to +30     {@<i> 'Y' :- @<i> 'C'.  @<i> 'Y' :- @<i> 'T'.}
```

"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'.

The splice-junction rules SUBSET extracts.

```
E/I :- @-02 'A--TR-G', not(E/I-stop).
E/I :- @-02 '---TRAG', not(E/I-stop).
E/I :- @-02 'AG-TR--', not(E/I-stop).
E/I :- @-02 '-G-TRA-', not(E/I-stop).
E/I :- @-02 '-G-TR-G', not(E/I-stop).

E/I-stop  :- E/I-stop9, E/I-stop6, not(E/I-stop7).
E/I-stop  :- E/I-stop5, not(E/I-stopA).
E/I-stop  :- E/I-stop2, not(E/I-stop7).
E/I-stop  :- E/I-stop1, not(E/I-stop7).
E/I-stop  :- E/I-stop8, not(E/I-stop7).
E/I-stopA :- E/I-stop7, E/I-stop4, E/I-stop3.

E/I-stop1 :- @-3 'TAA'.     E/I-stop4 :- @-4 'TAA'.     E/I-stop7 :- @-5 'TAA'.
E/I-stop2 :- @-3 'TAG'.     E/I-stop5 :- @-4 'TAG'.     E/I-stop8 :- @-5 'TAG'.
E/I-stop3 :- @-3 'TGA'.     E/I-stop6 :- @-4 'TGA'.     E/I-stop9 :- @-5 'TGA'.

I/E  :- @-06 'Y--YA-', pyramidine-rich, not(I/E-stop8).
I/E  :- @-06 'Y--YAG', not(I/E-stop8).
I/E  :- @-06 'Y---AG', pyramidine-rich, not(I/E-stop8).
I/E  :- @-06 'Y--Y-G', pyramidine-rich, not(I/E-stop8).
I/E  :- @-06 '---YAG', pyramidine-rich, not(I/E-stop8).

pyramidine-rich  :- @-14 '-YY--Y-Y'.
pyramidine-rich  :- @-14 'YYY--Y--'.
pyramidine-rich  :- @-14 'Y-YY----', not(pyramidine-rich0).
pyramidine-rich  :- @-14 '--YY-Y--'.
pyramidine-rich  :- @-14 '-YYY----'.
pyramidine-rich0 :- @-14 '----Y-Y-'.

I/E-stop1 :- @1 'TAA'.      I/E-stop4 :- @2 'TAA'.      I/E-stop7 :- @3 'TAA'.
I/E-stop2 :- @1 'TAG'.      I/E-stop5 :- @2 'TAG'.      I/E-stop8 :- @3 'TAG'.
I/E-stop3 :- @1 'TGA'.      I/E-stop6 :- @2 'TGA'.      I/E-stop9 :- @3 'TGA'.

For i from -30 to +30     {@<i> 'Y' :- @<i> 'C'.  @<i> 'Y' :- @<i> 'T'.}
```

This abbreviated set of rules extracted by SUBSET has a test set accuracy of 75%. Sets whose statistics are reported in Section 5 contain about 300 rules with a test set accuracy of 92%.