To appear in: *Machine Learning: An Artificial Intelligence Approach, volume IV*

Refining Symbolic Knowledge Using 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

To use artificial neural networks as part of a multistrategy learning system, there must be a way for neural networks to accept information, often expressed symbolically, from other systems. Moreover, neural networks must be able to transfer the results of their learning. A three-step process for learning using rule-based domain knowledge in combination with neural networks can address this task. Methods for performing the first two steps have been previously described. Hence, this chapter focuses on the final step, extracting symbolic rules from trained neural networks. Proposed and empirically evaluated in this chapter is a new method for rule extraction. The rules extracted by this method closely reproduce the accuracy of the network from which they came, are superior to the rules derived by a learning system that directly refines symbolic rules, and are human comprehensible. Hence this process allows the use of neural networks as a part of a multistrategy learning system. More generally, this work contributes to the understanding of how symbolic and connectionist approaches to artificial intelligence can be profitably integrated.

1 Introduction

Artificial neural networks (ANNs) have proven to be a powerful and general technique for machine learning (Fisher and McKusick, 1989; Shavlik *et al.*, 1991). Hence, it is tempting to use ANNs in combination with other, more knowledge intensive, learning strategies. However, ANNs have several well-known shortcomings. Perhaps the most significant of these shortcomings is that it is very difficult to determine why a trained ANN makes a particular decision. Yet, the extraction of accurate, comprehensible, symbolic knowledge from ANNs is necessary if ANNs are to be useful in a multistrategy framework. In addition, without the ability to explain their decisions, it is hard to be confident in the reliability of networks that solve real-world problems. The extraction of rules directly addresses both of these concerns, as rules provide a simple mechanism for explanation and transfer. Therefore, the extraction of comprehensible, symbolic rules from trained networks is desirable. That is the topic of this chapter.

The approach taken to extracting rules from trained ANNs uses KBANN; a tri-algorithmic multistrategy learning system illustrated by Figure 1. The first link in the algorithm chain creates a neural network that encodes domain knowledge; the encoded knowledge need be neither complete nor correct (Towell *et al.*, 1990). The second link trains the KBANN-net (i.e., a network created by KBANN) using a set of classified training examples and standard neural learning methods (e.g., Rumelhart *et al.*, 1986). The final link extracts rules from trained KBANN-nets.

Rule extraction is an extremely difficult task for arbitrarily-configured networks, but is somewhat less daunting for KBANN-nets due to their initial comprehensibility. The method described in this chapter takes advantage of their initial comprehensibility to efficiently extract rules from trained KBANN-nets. When evaluated in terms of their ability to correctly classify examples not seen during training, the rules produced by this method are equivalent to the accuracy of the networks from which they came. Moreover, the extracted rules are superior to the rules resulting from methods that act *directly* on the original rules (rather than their re-representation as a neural network). Also, the method is superior to the most widely-published algorithm for the extraction of rules from general neural networks.

The next section contains brief reviews of neural networks and the rules-to-network trans-

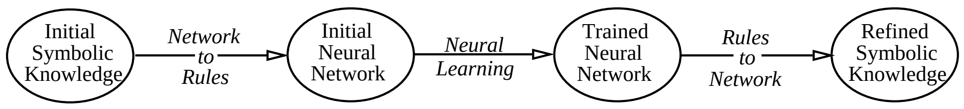


Figure 1: The flow of information through KBANN.

lator of KBANN. Subsequent sections describe two rule extraction methods, one original and one very similar to the best previously-reported method. The succeeding section presents empirical tests, using two real-world learning problems taken from molecular biology, that characterize the strengths and weaknesses of both rule-extraction methods. The final sections of this chapter put this work into the context of other approaches to understanding trained neural networks and discuss future research plans.

2 Review of Algorithms

This section briefly reviews learning in neural networks and the KBANN rules-to-network translator. An understanding of both of these topics is required to comprehend the rule-extraction techniques described in the following section.

2.1 Neural learning

Artificial neural networks are a class of learning systems inspired by the architecture of the brain and theories of learning in the brain (Rosenblatt, 1962; Hebb, 1949; McCulloch and Pitts, 1943). They are composed of cell-like entities (referred to as *units*) and connections between the units corresponding to dendrites and axons. (The connections are referred to as *links*.) Links are *weighted*; the weights roughly correspond to the efficacy of synapses in the brain. The effect of the weights is that the signal received by a unit is the product of the link weight and the signal sent across the link. Abandoning the brain metaphor, an ANN is a directed graph with weighted connections.

Units do only one thing – they compute a real-numbered output (known as an *activation*) which is a function of real-numbered inputs. Figure 2 shows the behavior of a single unit with N incoming links. The (output) *activation* of the unit is shown to be a function of the sum of the incoming signals and a "bias" term which acts as a threshold for the unit. A typical activation function (the "logistic") is illustrated by Figure 3. It shows that the activation of a unit is near zero when the net incoming activation is less than the bias and near one in the opposite case.

Units in neural networks are commonly sorted into three groups: *input*, *hidden* and *output*. These groups are shown graphically in Figure 4. Input units are so named because they receive signals from the environment. Similarly, output units are so named because their activation is available to the environment. (Generally the activation of the output units is the answer computed by the network.) Finally hidden units are so named because they have no direct interaction with the environment. They are purely internal to the network.

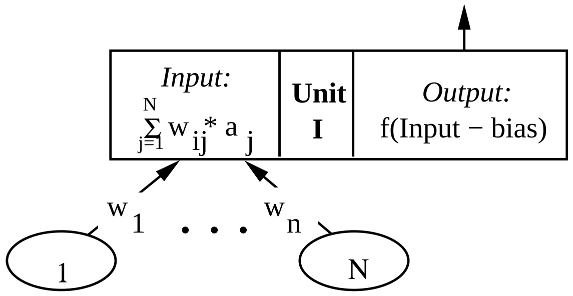


Figure 2: A single unit of an ANN.

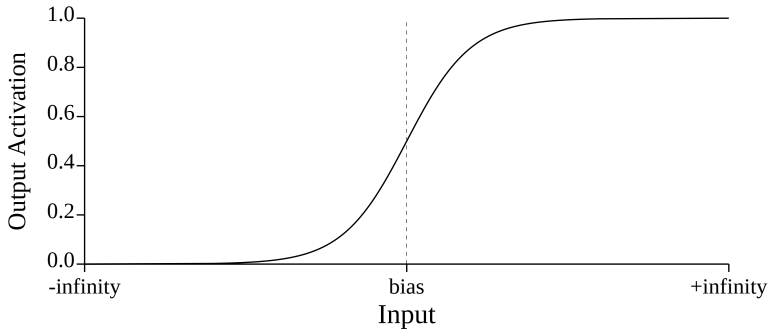


Figure 3: The activation of a unit in an ANN.

The general process of learning in a neural network, using the standard backpropagation model (Rumelhart *et al.*, 1986), is given by the five-step algorithm in Table 1. For the details of the backpropagation algorithm, see Rumelhart *et al.* (1986) or Hertz *et al.* (1991).

2.2 KBANN's 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, in the form of propositional Horn clauses, to define what is initially known about a topic. Detailed explanations of the procedure used by KBANN to translate rules into

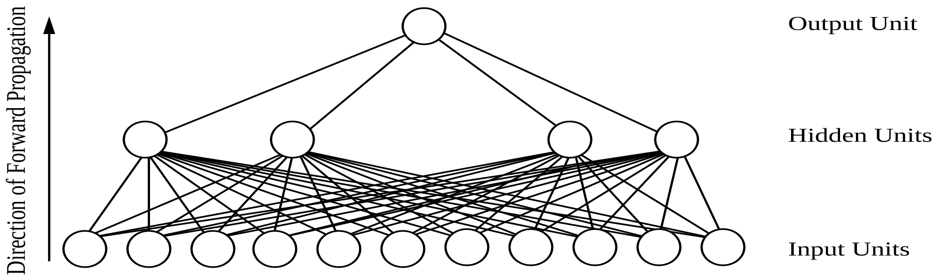


Figure 4: A prototypical artificial neural network.

Table 1: The backpropagation algorithm.

1. Activations of input units are set by the environment;
2. Activation is propagated forward along the directed connections (links) possibly through hidden units to the output units;
3. Errors are determined as a function of the difference between the computed activation of the output units and the desired activation of the output units;
4. Errors are propagated backward along the same links used to carry activations;
5. Changes are made to link weights to reduce the difference between the actual and desired activations of the output units.

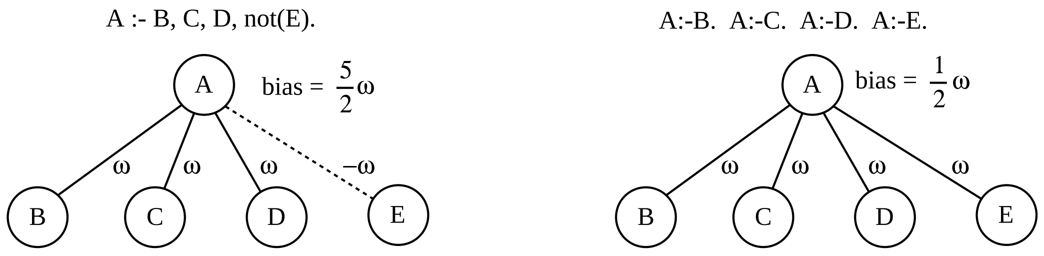


Figure 5: Translation of conjunctive and disjunctive rules into a KBANN-net.

a KBANN-net are given by Towell *et al.* (1990) and Towell (1991); the following is a brief summary.

KBANN translates a collection of rules into a neural network by individually translating each rule into a small subnetwork that accurately reproduces the behavior of the translated rule. These small subnetworks are then assembled to form a single neural network that mimics the behavior of the whole rule. The next paragraphs describe how KBANN creates subnetworks that encode conjuncts and disjuncts. Following that is an example of a full rules-to-network translation.

Conjunctive rules are translated into a neural network by setting weights on all links corresponding to positive (i.e., unnegated) antecedents to ω, weights on all links corresponding to negated antecedents to $-\omega$, and the bias on the unit corresponding to the rule's consequent to $\frac{2P+1}{2}\omega$; P is the number of positive antecedents. KBANN commonly uses a setting of $\omega = 4$, a value empirically found to work well. For example, Figure 5 shows the encoding of a conjunctive rule.

To translate a set of rules encoding a disjunct, KBANN sets the weight of each link corresponding to a disjunctive antecedent to ω and the bias on the unit encoding the consequent

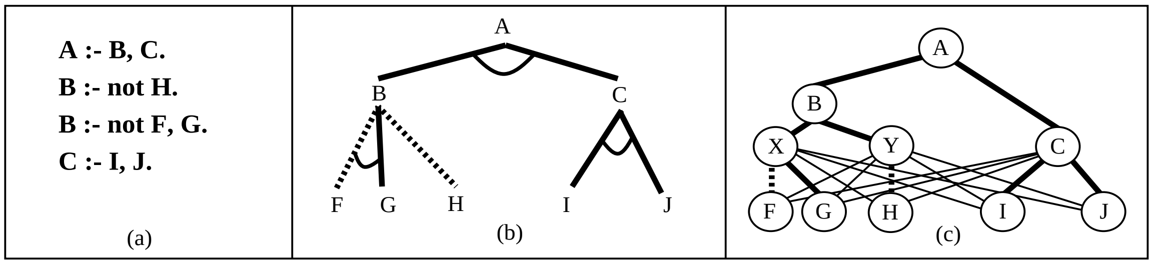


Figure 6: Translation of a knowledge base into a KBANN-net.

to $\frac{\omega}{2}$. For example, Figure 5 shows the network that encodes a four-rule disjunct.

Intuitively, these strategies for translating conjunctive and disjunctive rules are reasonable. In both cases, the incoming activation overcomes the bias only when the consequent is satisfied.

As an example of the KBANN method, consider the sample knowledge base in Figure 6a, which defines membership in category A. Figure 6b represents the hierarchical structure of these rules: solid and dotted lines represent necessary and prohibitory dependencies, respectively. Figure 6c represents the KBANN-net that results from the translation into a neural network of this knowledge base. Units X and Y in Figure 6c are introduced into the KBANN-net to handle the disjunction in the rule set. Otherwise, each unit in the KBANN-net corresponds to a consequent or an antecedent in the knowledge base. The thick lines in Figure 6c represent heavily-weighted links (i.e., links with weight ω) in the KBANN-net; they 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 neural networks 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 KBANN-net.

3 Extracting Rules from Trained Networks

After KBANN-nets have refined, they can be used as highly accurate classifiers (Towell *et al.*, 1990). However, trained KBANN-nets provide no explanation of how an answer was derived. Nor can the results of their learning be shared with humans or transferred to related problems. (The work of Pratt *et al.* (1991) partially ameliorates this problem.)

The extraction of symbolic rules directly addresses both of these problems. It makes the information learned by the KBANN-net accessible for human review and justification of answers. Moreover, the modified rules can be used as a part of knowledge bases for the solution of related problems.

3.1 Underpinnings of rule extraction

3.1.1 Assumptions

Almost every method of rule extraction makes one assumption about trained networks. Specifically, that the units in trained nets are always either fully active (≈ 1) or inactive (≈ 0). This assumption is not particularly restrictive; the standard logistic activation function can be slightly modified to ensure that units approximate step functions.

The methods described in this chapter also assume that training does not significantly shift the meaning of units. By making this assumption, the methods are able to attach labels to rules that correspond to terms in the domain knowledge upon which the network is based. These labels enhance the comprehensibility of the rules.

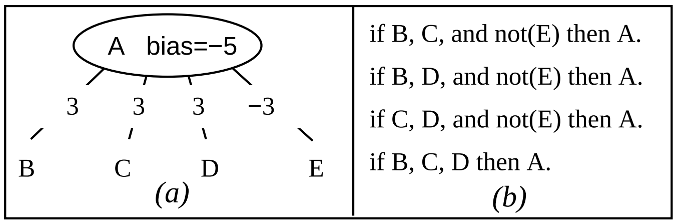
3.1.2 Commonalities

The methods for extracting rules from neural networks described below, as well as those 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. Broadly speaking, if the summed weighted inputs to a unit exceeds its bias, then its activation will be near one. Otherwise its activation will be near zero. *Hence, rule extraction methods need only look for ways in which the weighted sum of inputs exceeds the bias.*

The first of the above assumptions, that all units in trained networks have activations near zero or one, simplifies this task by ensuring that links carry a signal equal to their weight or no signal at all. As a result, rule extractors need only be concerned with the weight of links entering a unit and may ignore the activation of the sending unit.

Rule extraction is further simplified by the fact that, in the networks created by KBANN, units always have non-negative activations. Therefore, negatively-weighted links can only give rise to negated antecedents, while positively-weighted links can only give rise to un-negated antecedents. This considerably reduces the size of the search space (Fu, 1991).

The following subsections present two methods of rule extraction. This first method has been previously described (Saito and Nakano, 1988). It is presented here for purposes of comparison.



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.

Figure 7: Rule extraction using the SUBSET algorithm.

3.2 The SUBSET method

Given the above assumptions, the simplest method for extracting rules can be referred to as the SUBSET method. This method operates by exhaustively searching for subsets of the links into a unit such that the sum of the weights of the links in the subset guarantees that the total input to the unit exceeds its bias. In the limit, SUBSET extracts a set of rules that reproduce the behavior of the network. However, the combinatorics of this method render it impossible to execute except on small networks that solve toy problems. Heuristics can be added to reduce the complexity of the search at some cost in the accuracy of the resulting rules. Unfortunately, even using heuristic search, SUBSET tends to produce repetitive rules whose preconditions are difficult to interpret.

As a brief example of the SUBSET method, consider the unit in Figure 7a. Given that the link weights and the bias are as shown, the four rules listed in Figure 7b are extracted by the algorithm. To find these four rules, SUBSET must check all possible rules with 1, 2, 3 or 4 antecedents. (See papers by Saito and Nakano (1988) or Fu (1991) for more detailed explanations of SUBSET.)

3.3 The NOFM method

The second algorithm for rule extraction, referred to as NOFM, explicitly searches for antecedents of the form:

```
if (N of these M antecedents are true) then ...
```

This approach was taken because the rule sets discovered by the SUBSET method often contain "N-of-M" style concepts. Furthermore, experiments indicate that ANNs are good at learning N-of-M concepts (Fisher and McKusick, 1989) and that searching for N-of-M concepts is a useful inductive bias (Murphy and 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 NOFM, an abstracted version of which appears in Table 2, is that individual antecedents (links) do not have a unique importance. Rather, groups of antecedents

Table 2: The NofM 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 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 the remaining links.
6. Where possible, simplify rules to eliminate weights and thresholds.

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 NofM algorithm; it allows the algorithm to consider groups of links without worrying about the particular links within the group.

The next subsection contains a detailed description of each step of the NofM algorithm. A detailed example appears in the subsequent subsection.

3.3.1 Step-by-step description of the NofM algorithm

Step 1, clustering. Backpropagation training tends to group links of Kbann-nets into loose clusters rather than tight equivalence classes as assumed by the NofM algorithm. Hence, the first step of NofM is to group links into equivalence classes. This grouping can be done in either of two ways.

First, clustering may be done using a standard clustering method such as the *join* algorithm (Hartigan, 1975). This method clusters by joining the two closest clusters starting with n clusters of size 1. Clustering stops when no pair of clusters is closer than a set distance (Kbann uses 0.25).

An alternative method for clustering links that takes advantage of the constraints on this particular clustering problem is also available. This method is used almost exclusively as its complexity is $O(n \times log(n))$ while the complexity of the *join* procedure is $O(n^2)$.

Step 2, averaging. After groups are formed, the second step of the algorithm is to set the weight of all links in each group to the average of each group's weight. Thus, the first two steps of the algorithm 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 heuristic, and one algorithmic.

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 (taking into account properties of units — e.g., that only one unit related to each nominal input feature may be active at any time). This total possible activation is then compared to the levels of activation that are reachable given link weights in the network. Clusters that cannot change whether the net input exceeds the bias are eliminated.

This procedure is very similar to SUBSET. However,clustering of link weights considerably reduces the combinatorics of the problem. For instance, consider a unit with a bias of -10 and three clusters of links: (A) two links of weight 7, (B) three links of weight 4, and (C) ten links of weight 0.1. In this case, cluster (C) is eliminated. Its total possible activation is 1.0 and the only reachable activations that are less than the bias using the clusters (A) and (B) are 7 and 8. Neither of these activation levels, when combined with the 1.0 total from group C, exceeds 10.0. Hence, the links in cluster (C) can have no effect on the unit into which they feed. So, the cluster is safely eliminated.

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 a training example to the clustered network and sequentially zeroing the input from each cluster. If doing so results in a change in the activation of the unit receiving activation from the cluster, then the cluster is marked as necessary. After doing this for every example, clusters not marked as necessary are eliminated.

Step 4, optimizing. With unimportant groups eliminated, the fourth step of NOFM is to optimize the bias on the unit. This step is necessary because the averaging of the link weights in a cluster and elimination of links can change the times at which a unit is active. As a result, prior to optimization, networks on which the first three steps of the NOFM algorithm have been applied 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 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 NofM algorithm forms rules that simply re-express the network. That is, 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 those antecedents tend to be in a few weight classes. (See the extracted rules presented later in this chapter for examples of unsimplified rules.)

Step 6, simplifying. Finally, rules are simplified whenever possible to eliminate the weights and thresholds. Simplification is accomplished by scanning each restated rule to determine the possible combinations of group items 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 individual rules and complexity resulting from a number of simple rules.

For example, consider the rule[1]:

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

The simplification procedure would simplify this rule by rewriting it as 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.

3.3.2 Example of the NofM method

As an example of NofM, consider Figure 8. This figure illustrates the process through which a single unit with seven incoming links is transformed by the NofM procedure into a rule

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

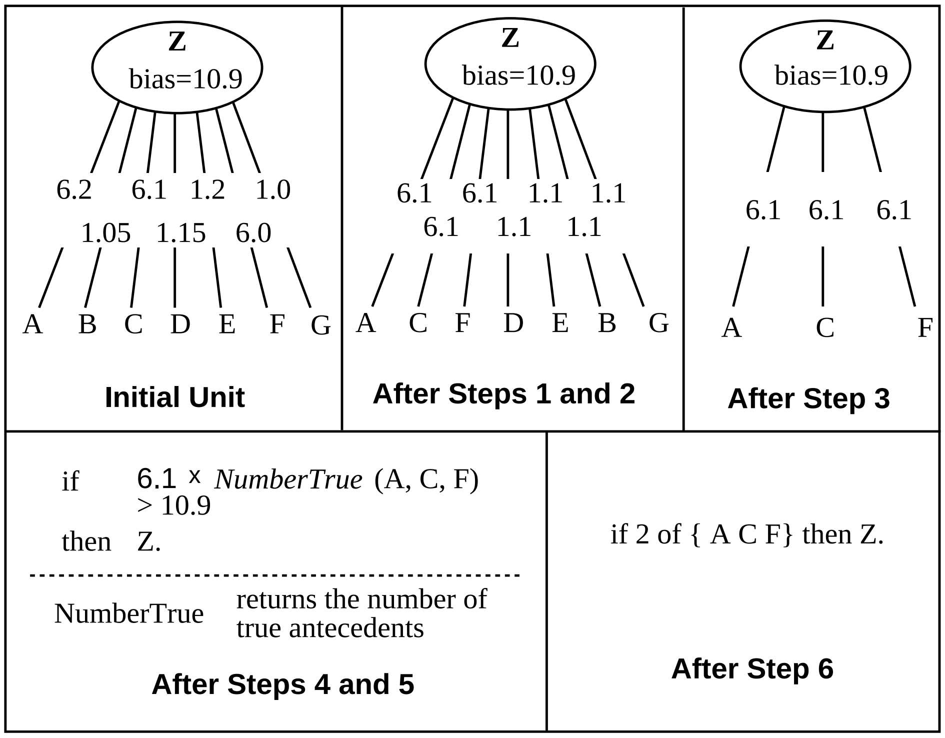


Figure 8: An example of rule extraction using NofM.

that requires two of three antecedents to be true.

The first two steps of the algorithm (Table 2) transform the unit with seven unique inputs into a unit with two classes of inputs, one with three links of weight 6.1 and one with for links of weight 1.1. Step 3 of the algorithm eliminates the group with weight 1.1 from consideration because there is no way that these links – either alone or in combination with links in the other group – can affect whether or not the sum of the incoming activation to unit Z exceeds the bias on Z. This step takes advantage of the assumption that units necessarily have activations near zero or one. Figure 8 does not illustrate bias optimization (Step 4).

The lower left panel of Figure 8 shows the re-expression of the simplified unit as a rule. The sixth and final step of the algorithm is illustrated by the bottom right panel of Figure 8, in which the rule with weighted antecedents and a threshold is transformed into a simple N-of-M style rule. (The Subset algorithm would find the three rules that are the expansion of the 2-of-3 rule found by NofM. However, to find these three rules Subset would have had to consider as many as 125 possibilities.)

3.3.3 Algorithmic complexity of NofM

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. This considerably speeds the simulation of neural networks on a serial machine. 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 $O(m \times n)$ time — m is the number of units, n is the average number of links received by a unit — except for the initial clustering, which requires $O(m \times n \times log(n))$ time and cluster elimination which requires $O(x \times m \times n)$ time — x is the number of training examples.

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 are easier to understand than those of Subset. More significantly, Subset is an exponential algorithm whereas NofM is approximately cubic. Finally, results presented in the next section indicate that the rules derived by NofM are more accurate than the rules derived by Subset.

4 Datasets

Before presenting the experiments, this section briefly describes the two real-world datasets from the domain of molecular biology used for testing Kbann. Both datasets have been previously used to demonstrate the usefulness of the Kbann algorithm (Noordewier *et al.*, 1991; Towell *et al.*, 1990). They are more thoroughly described by Towell (1991).

4.1 Background and notation

The two biological datasets in this chapter are taken from the domain of DNA sequence analysis. DNA, the "blueprint" of almost all living organisms, is a linear sequence from the alphabet {A, G, T, C}. Each of these characters is referred to as a *nucleotide*. Human DNA

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

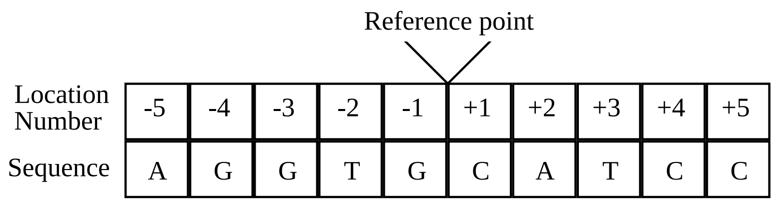


Figure 9: The numbering of nucleotide locations in DNA sequences.

Table 3: 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		

consists of approximately $3 * 10^9$ nucleotides. By contrast, the DNA of *E. coli*, a common intestinal bacterial, contains about $5 * 10^6$ nucleotides.

This paper uses a special notation for specifying locations in a DNA sequence. The idea is to number locations with respect to a fixed, biologically-meaningful, reference point. Negative numbers indicate sites preceding the reference point (by biological convention, this appears on the left) while positive numbers indicate sites following the reference point. (Zero is not used.) Figure 9 illustrates this numbering scheme. Rules use this referencing scheme by stating a position with respect to the reference location, denoted by '@', and the giving a subsequence in the positive direction. For example `@-4'GGT'` refers to the three nucleotide long sequence in Figure 9 that begins at position -4 and ends at position -2.

In addition to this notation for specifying locations a DNA sequence, Table 3 specifies a standard coding scheme for referring to any possible combination of nucleotides using a single letter (IUB Nomenclature Committee, 1985). This scheme is compatible with the codes used by the EMBL, GenBank, and PIR data libraries, three major collections of data for molecular biology.

4.2 Promoter recognition

The first problem is *promoter recognition*. Briefly, promoters are short DNA sequences that precede the beginnings of genes. The input features for promoter recognition are a sequence of 57 DNA nucleotides. Following biological convention, the reference point for promoter recognition is the site at which gene transcription begins (if the example is a promoter). The

Table 4: The initial rules for promoter recognition.

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

minus35 :- @-37 ‘CTTGAC-’.                minus10 :- @-14 ‘TATAAT--’.
minus35 :- @-37 ‘-TTG-CA’.                minus10 :- @-14 ‘-TA-A-T-’.
minus35 :- @-37 ‘-TTGACA’.                minus10 :- @-14 ‘-TATAAT-’.
minus35 :- @-37 ‘-TTGAC-’.                minus10 :- @-14 ‘--TA---T’.

conformation  :-  @-49 ‘----AA--A’.
conformation  :-  @-49 ‘----A---A------------TT---T-AA---T-T---------T’.
conformation  :-  @-49 ‘A----T----------------T----A--T-TG--------------A’.
conformation  :-  @-49 ‘--CAA-TT-AC----------------G---T-C-------GCGCC-CC’.
```

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

Table 4 contains the initial rule set used in the promoter recognition task. According to these rules, promoters are defined by `contact` and `conformation`. `Contact` requires matching short DNA sequences in two regions, one 35 nucleotides before the reference point and the other ten nucleotides before the reference point.

The training examples include 53 sample promoters and 53 nonpromoter sequences. Prior to training, the rules in Table 4 do not classify any of the 106 examples as promoters. Thus, the rules are useless as a classifier. Nevertheless, they do capture a significant amount of information about promoters.

4.3 Splice-junction determination

The second dataset is for *splice-junction determination.* Splice junctions are points on a DNA sequence at which ‘superfluous’ DNA is removed during the process of protein creation in higher organisms.

In this dataset, there are 1000 examples, each 60 nucleotides long, divided among three categories. In addition, there is an approximately-correct set of 21 rules (not shown) derived from the biological literature (Watson *et al.*, 1987) by Noordewier *et al.* (1991). This set of rules classifies 61% of the examples correctly.

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. These are evaluated 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 assessed by analyzing individual extracted rules. The part includes a set of the rules extracted from trained KBANN-nets by the NOFM method.

5.1 Testing methodology

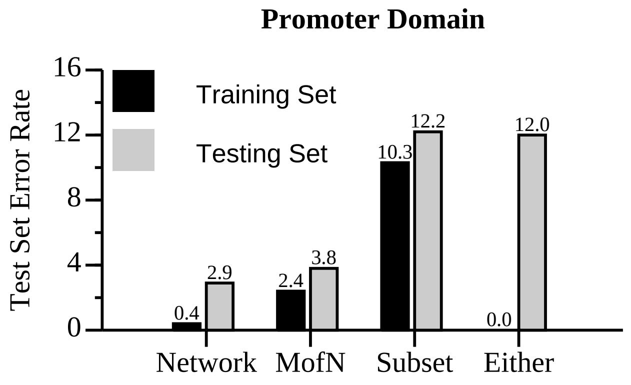
Following Weiss and Kulikowski (1990), networks are trained using repeated 10-fold cross-validation for assessing the quality of learning in both domains. Following Hinton's (1989) suggestion for improved network interpretability, all weights are subject to gentle "decay" during training.[3] Finally, networks are trained using the *cross-entropy* error function (Hinton, 1989).

5.1.1 Accuracy of the extracted rules

Figure 10 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 10 includes the accuracy of the trained KBANN-nets prior to rule extraction (the bars labeled "Network"). Also included in Figure 10 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 and Mooney, 1990). The numbers for EITHER are derived from Ourston's thesis (Ourston, 1991); they reflect a slightly different testing method. (EITHER has not been tested on the splice-junction problem.)

Recall 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 10 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

[3]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 ϕ is set to a very gentle 0.99999. Weight decay was not used in previously reported experiments simply because it added unnecessary complexity to these experiments. It has little or no measurable effect of the generalization ability of KBANN-nets.



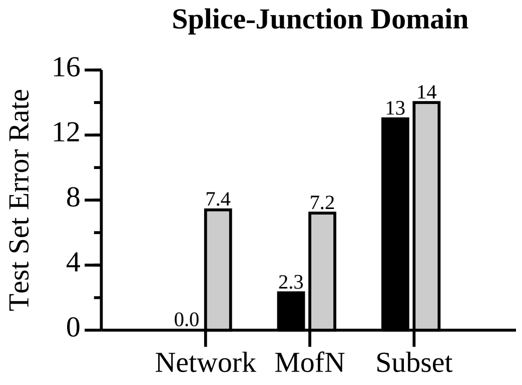


Figure 10: The error rates of extracted rules.

the rules extracted using SUBSET. More importantly, 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. Statistical comparisons to EITHER are not made due to lack of compatible data.)

One of the more interesting results in this chapter is that, on the testing examples in the splice-junction domain, the error rate of the NOFM rules is superior to that of the networks from which the rules are extracted. (One-tailed, paired-sample t-tests indicate that the difference is statistically significant with 90% confidence t=1.6, d.f.=9.) Earlier tests (Towell and Shavlik, 1991) showed the error 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.

Conversely, the error rate of the SUBSET rules on testing examples is statistically worse than the networks in both problem domains. Discussion at the end of this section analyzes reasons why NOFM rules can be superior to the networks from which they are extracted.

5.2 Comprehensibility

To be useful, the extracted rules must not only be accurate, they also must be *understandable.* While understandability is an ill-defined concept, there are several ways in which it might be measured. Several quantitative measures are presented and analyzed by Towell (1991). This section merely presents and discusses the extracted rules.

Table 5 presents the rules NOFM extracts from a KBANN-net trained for promoter recognition. While these rules are somewhat murky, they are vastly more comprehensible than the network of 3000 links from which they came. Moreover, the rules in this table can be

Table 5: The promoter rules NOFM extracts.

```
Promoter :- Minus35, Minus10.

Minus-35 :-10 < 4.0 x nt(@-35 'TTGAT-')+     Minus-10 :-           @-13 'TAWA-T--'.
                1.5 x nt(@-35 '--TCC-')+
                0.5 x nt(@-35 '-MC---')-     Minus-10 :-        2 of @-12 '-CA---T' and
                1.5 x nt(@-35 'GGAGG-').                  not (1 of @-12 '-RB---S').

Minus-35 :-10 < 5.0 x nt(@-35 'T-G--A')+     Minus-10 :-10 < 3.0 x nt(@-12 'TAT--T-') +
                3.1 x nt(@-35 '-GT---')+                     1.8 x nt(@-12 '---GA--') +
                1.8 x nt(@-35 '-CC-MT')+                     0.7 x nt(@-12 '--GAT--') -
                1.8 x nt(@-35 'CAA-GG')-                     0.7 x nt(@-12 'GKCCCS-').
                3.1 x nt(@-35 'A----C').
                                             Minus-10 :-10 < 3.7 x nt(@-12 'KA-M-T-') +
Minus-35 :-          @-36 'C-TGAC-'.                         1.0 x nt(@-12 '-T---A-') -
                                                             1.0 x nt(@-12 'CS-G-S-') -
Minus-35 :-           @-35 'TTD-CA'.                         3.0 x nt(@-12 'A--T---').
```

"nt()" returns the number of nucleotides that match the parenthesized sequence.

rewritten in a form very similar to one used in the biological community (Stormo, 1990), namely weight matrices.

The major pattern in the extracted rules is that KBANN learned to disregard `conformation`. The `conformation` rules are also dropped by EITHER (Ourston, 1991), which suggests that dropping these rules is not an artifact of KBANN but rather that DNA bases outside the *minus35* and *minus10* regions are less important than the conformation hypothesis (Koudelka *et al.*, 1987) suggests. Hence, machine learning methods can provide valuable evidence confirming or refuting biological theories.

In general, the rules NOFM extracts confirm the importance of the nucleotides 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 -12 is a strong indicator that the rule is true. Replacing the 'T' with an 'A' prevents the rule from ever being satisfied.

The rules extracted by NOFM for the splice-junction domain are not shown. They are very similar to those extracted for promoter recognition. The rules extracted by SUBSET are also not shown; they would cover several pages. (See (Towell, 1991) for details.)

5.3 Discussion of rule extraction

The results presented in this section indicate that the NofM method is able to extract a good set of rules from trained networks. Specifically, with the NofM method, not only can comprehensible, symbolic rules be extracted from trained Kbann-nets, the extracted rules can be superior (at classifying testing examples) to the Kbann-net from which they are extracted. 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 and Mooney, 1990, see Figure 10). 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 algorithm for network-to-rules translation offers an appealing mixture.

Two hypotheses explain the superiority of NofM 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. Hence, it may be possible to more closely fit a target concept than when taking the large steps required by direct rule-refinement.

Second, the relative success of the rule-extraction method proposed in this chapter, might result from its better fitting the nature of the investigated problems. That is, 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 successful multistrategy learning may involve searching "language space" as well as the space of solutions possible within a language.

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. Neither Either nor the Subset method can easily express N-of-M rules. As a result, these algorithms may be at a disadvantage on the molecular biology test domains. Hence, some of the advantage of the NofM method may result from its output language better fitting that of the "natural" language of DNA sequence-analysis problems.

The observation that NofM rules can be superior to the networks from which they are extracted cannot be attributed to a language bias, as the NofM rules are only a subset of the languages expressible using neural networks. Instead, the advantage of the NofM rules most likely occurs because the rule-extraction process reduces overfitting of the training examples. Several pieces of evidence support this hypothesis. First, as noted previously, revising training procedures to reduce overfitting in the promoter domain eliminated the advantage that NofM rules had over the networks from which they were extracted (Towell and Shavlik, 1991). Second, the difference in ability to correctly categorize testing and training examples is smaller for NofM rules than for trained Kbann-nets. In other words, the rules that the NofM 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 worse on the training set than both NofM rules and trained Kbann-nets on the testing set.

A third piece of evidence in support of the overfitting hypothesis comes from the work on pruning neural networks to eliminate superfluous parts (Mozer and Smolensky, 1988; Le Cun *et al.*, 1989). Note that 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.

6 Related Work

There are three distinct sets of work that are closely related to the rule-extraction methods presented in this chapter. The first set contains "all symbolic" methods of learning from rules and examples (e.g., Ourston and 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 direct symbolic methods are appealing because they do not require a shift between symbolic and neural representations, the results presented in Figure 10 indicate that they are not as accurate as the methods presented here.

The second set of ideas related to those in this chapter is made up of methods for understanding neural networks that attempt to prune away unimportant parts of network. In so doing, the complexity of the solution is reduced, thereby making the network more easily understood. For instance, Le Cun *et al.* (1989) describe a method that eliminates as much as 30% of the free parameters in a network without increasing error. Experiments with

Le Cun's method, and a related technique described by Weigand, Rumelhart and Huberman (1990), on the promoter domain indicate that, on this problem, a majority of the links can be removed from KBANN-nets without increasing error. Mozer and Smolensky (1988) report a related technique which 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 described in this chapter are similar in many ways to these pruning systems, this work differs significantly in its emphasis on comprehensible output.

The third group of work related to this chapter are attempts to extract rules from randomly-weighted ANNs. Both Saito and Nakano (1988) and Fu (1991) report a method similar to the SUBSET algorithm. Saito and Nakano's method looks at the input/output behavior of trained networks to form rules that map directly from inputs to outputs. To control 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 users in a sea of rules.

Several groups have reported attempts to extract rules from networks that, like KBANN-nets, 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. The method is able to 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 iteratively learns sets of propositional rules.

7 Future Work

One of the principle areas of future work is in the training of networks to encourage the clustering required by the NOFM method. Doing so could reduce the complexity of the clustering and possibly eliminate the need for optimizing the rule thresholds. One of the methods being investigated is to adapt a training technique described by Nowlan and Hinton (1991). Their method trains networks so that link weights fall into a small number of Gaussian clusters. The network is able to learn the center locations and distributions of these clusters. This idea might be adapted to training KBANN-nets by specifying the starting values of the Gaussians so that they cover the initial link weights. During training, these Gaussians will tend to encourage link weights to stay near their initial values. However, when differentiation among the link weights is required, the Gaussians can adapt to allow it. Networks so trained might be quite amenable to NOFM-style rule extraction.

In addition, an N-of-M style representation appears to be a useful inductive bias for many problems (Murphy and Pazzani, 1991). Hence, it is desirable to enhance the comprehensibility of the rules returned by NOFM. While the rules extracted from networks by NOFM are much more comprehensible than the networks from which they are extracted, they are often not easily comprehended. The hope is that alternate forms of presentation or some slight modifications to the NOFM algorithm will improve the comprehensibility of the resulting rules.

A final path for future work is the investigation of the training and interpretation of networks that have units not directly specified by the initial rule set (Towell *et al.*, 1991). To this point, efforts have concentrated upon extending networks for the splice-junction problem as the KBANN-net for this problem lacks sufficient freedom to find a highly-accurate solution. Unfortunately, expanding networks by the addition of hidden units hampers rule extraction as the hidden units often violate both the assumption that their activations will be near zero or one after training and that they have a meaningful name. Hence, efforts to this time have focused on methods to satisfy these assumptions.

8 Conclusions

Multistrategy learning often requires multiple representations of knowledge. Training examples are often amenable to such representational shifts. However, domain-specific knowledge can be more difficult to shift between representations. The initial paper describing KBANN 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 these initial efforts were classifiers more accurate than those obtainable using only training examples (Towell *et al.*, 1990; Noordewier *et al.*, 1991).

In one sense these results are a dead end, for the refined knowledge in the trained networks is inaccessible for further learning by symbolically-oriented methods. Hence, this chapter presents a method for completing the *symbolic* $\Rightarrow$ *neural* $\Rightarrow$ *symbolic* circle. By so doing, the dead end is opened, making the highly-accurate neural classifiers that result from KBANN accessible to further learning. Access to the results of neural learning is gained by the extraction of rules from trained networks, for which this chapter has presented the NOFM algorithm. This technique takes advantage of some of the properties of the KBANN-nets to reduce the problem of the extraction of rules from neural networks to a manageable scale.

Experimental results indicate that concise, comprehensible rules can be extracted by the described NOFM algorithm without a serious loss of accuracy. In fact, the extracted rules can be more accurate at classifying testing examples than the networks from which they are extracted. By comparison, the state-of-the-art technique for rule extraction (i.e., the SUBSET method) is significantly worse than the rules extracted by NOFM and the networks from which the rules were extracted. In addition, these results indicate that the shift between symbolic and connectionist representations improves the accuracy of the resulting rules. Moreover, the extracted rules show the utility of machine learning techniques as a method of validating biological theories. Finally, the extra work required by this three-step system, by comparison to "all symbolic" rule-refinement methods (e.g., Ourston and Mooney, 1990; Thompson *et al.*, 1991), is shown to be worthwhile, in that the 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 and several anonymous reviewers 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

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

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

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

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

Hebb, D. O., *The Organization of Behavior*, Wiley, New York, 1949.

Hertz, J., Krogh, A., and Palmer, R. G., *Introduction to the Theory of Neural Computation*, Addison Wesley, Redwood City, CA, 1991.

Hinton, G. E., "Connectionist learning procedures", *Artificial Intelligence*, Vol. 40, pp. 185–234, 1989.

IUB Nomenclature Committee, "Ambiguity codes", *European Journal of Biochemistry*, Vol. 150, pp. 1–5, 1985.

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

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

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

McCulloch, W. S. and Pitts, W. A., "A logical calculus of ideas immanent in nervous activity", *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115–133, 1943.

McMillan, C., Mozer, M. C., and Smolensky, P., "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, 1991.

Mozer, M. C. and Smolensky, P., "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, 1988.

Murphy, P. M. and Pazzani, M. J., "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, 1991.

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

Nowlan, S. J. and Hinton, G. E., "Simplifying neural networks by soft weight-sharing", *Advances in Neural Information Processing Systems*, Vol. 4, Denver, CO, Morgan Kaufmann, 1991.

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

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

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

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

Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning internal representations by error propagation", *Parallel Distributed Processing: Explorations in the microstructure of cognition. Volume 1: Foundations*, Rumelhart, D. E. and McClelland, J. L., (eds.), pp. 318–363, MIT Press, Cambridge, MA, 1986.

Saito, K. and Nakano, R., "Medical diagnostic expert system based on PDP model", *Proceedings of IEEE International Conference on Neural Networks*, Vol. 1, pp. 255–262, 1988.

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

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

Stormo, G. D., "Consensus patterns in DNA", *Methods in Enzymology*, Vol. 183, pp. 211–221, Academic Press, Orlando, FL, 1990.

Thompson, K., Langley, P., and Iba, W., "Using background knowledge in concept formation", *Proceedings of the Eighth International Machine Learning Workshop*, pp. 554–558, Evanston, IL, 1991.

Towell, G. G. and Shavlik, J. W., "Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules", *Advances in Neural Information Processing Systems*, Vol. 4, Denver, CO, Morgan Kaufmann, 1991.

Towell, G. G., Shavlik, J. W., and Noordewier, M. O., "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, 1990.

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

Towell, G. G., *Symbolic Knowledge and Neural Networks: Insertion, Refinement, and Extraction*, PhD thesis, Computer Sciences Department, University of Wisconsin, Madison, WI, 1991.

Watson, J. D., Hopkins, H. H., Roberts, J. W., Steitz, J. A., and Weiner, A. M., *The Molecular Biology of the Gene*, Benjamin-Cummings, Menlo Park, CA, 1987.

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

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