Appears in *Machine Learning: Proceedings of the Eleventh International Conference*,
W. W. Cohen & H. Hirsh, eds., Morgan Kaufmann, San Francisco, CA, 1994

Using Sampling and Queries to Extract Rules from Trained Neural Networks

Mark W. Craven and Jude W. Shavlik
Computer Sciences Department
University of Wisconsin
1210 West Dayton St.
Madison, WI 53706
craven@cs.wisc.edu, shavlik@cs.wisc.edu

Abstract

Concepts learned by neural networks are difficult to understand because they are represented using large assemblages of real-valued parameters. One approach to understanding trained neural networks is to extract symbolic rules that describe their classification behavior. There are several existing rule-extraction approaches that operate by searching for such rules. We present a novel method that casts rule extraction not as a search problem, but instead as a learning problem. In addition to learning from training examples, our method exploits the property that networks can be efficiently queried. We describe algorithms for extracting both conjunctive and M-of-N rules, and present experiments that show that our method is more efficient than conventional search-based approaches.

1 INTRODUCTION

A problem that arises when neural networks are used for supervised learning tasks is that, after training, it is usually difficult to understand the concept representations formed by the networks. To address this limitation, a number of approaches have been developed for extracting symbolic representations from trained networks (Craven & Shavlik, 1993; Fu, 1991; Gallant, 1993; Saito & Nakano, 1988; Thrun, 1993; Towell & Shavlik, 1993). These approaches cast the rule-extraction task as a search problem, where the search involves finding rules that explain the activations of output and hidden units in the network. In this paper we present a novel approach to extracting symbolic rules from neural networks that frames the problem not as a search task, but instead as a supervised learning task. The target concept, in this learning task, is the function computed by the network. In addition to learning from training examples, our method exploits the property that networks can be *queried*. That is, they can be used to answer questions about whether specific instances are covered by the target concept. We present experiments that illustrate that our method has a significant efficiency advantage over search-based approaches.

A concept representation learned by a neural network is usually difficult for humans to understand because the representation is encoded by a large number of real-valued parameters. It is often important, however, to be able to inspect a learned concept definition. For domains such as medical diagnosis, the users of a learning system must understand how the system makes its decisions in order to be confident in its predictions (e.g., Wolberg et al., in press). Learning systems can also play an important role in the process of scientific discovery. A system may discover salient features in the input data whose importance was not previously recognized. If the representations formed by the learner are comprehensible, then these discoveries can be made accessible to human review (e.g., Hunter & Klein, 1993).

There are several existing search-based approaches for extracting propositional *if-then* rules from trained networks (Fu, 1991; Gallant, 1993; Saito & Nakano, 1988; Thrun, 1993). These approaches find conjunctive rules by searching for combinations of input values that, when satisfied, guarantee that a given unit is active, regardless of the state of the other inputs to the unit. A limitation of these methods, however, is that the computational complexity of the search is exponential in the number of input features. Since many real-world problems involve a large number of features (e.g., Lapedes et al., 1989; Sejnowski & Rosenberg, 1987) this problem is a significant one. We present experiments that demonstrate that, in some cases, our learning-based approach requires much less computation than do these search-based methods to get rule sets that model networks to same degree of accuracy. Significantly, our method is likely to be more efficient than search-based approaches for problems that involve a large number of features.

In contrast to methods that extract *conjunctive* rules, Towell and Shavlik (1993) developed an approach for extracting *M*-of-*N* rules from *knowledge-based* neural networks.[1] In later work, we generalized this approach to ordinary neural networks by employing a special training procedure (Craven & Shavlik, 1993). The advantages of describing networks using *M*-of-*N* rules are that the search can be made more efficient than for conjunctive rules, and the extracted rule sets are usually more concise. We describe how our learning-based approach can be used to extract *M*-of-*N* rules in addition to conjunctive rules. Our learning-based approach offers four distinct advantages over our previously-described method for extracting *M*-of-*N* rules from ordinary networks: (a) it does not require a special training regime for the network, (b) it can efficiently extract rules which describe a network to an arbitrary degree of accuracy, (c) it does not require that hidden units be approximated as threshold units, and (d) it does not require that the extracted rules use an intermediate term to represent each hidden unit.

The next section of this paper describes how rule extraction has been viewed as a search problem in previous work. Section 3 introduces our learning-based approach to rule extraction and empirically compares it to a search-based method. Section 4 describes how our approach can be generalized to extract *M*-of-*N* rules from trained networks. Section 5 discusses future work, and Section 6 provides concluding remarks.

2 RULE EXTRACTION AS SEARCH

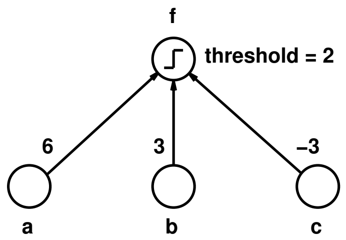
In this section, we briefly define the task of rule extraction and then discuss previous approaches to this task. There are two dimensions along which we characterize rule-extraction methods: the strategy used to explore a space of possible rules, and the method used to test hypothesized rules. The work described in this paper investigates the former dimension. In this section, we describe how a rule space can be explored using top-down search, and in Section 3 we describe our bottom-up learning approach. This section also provides a brief description of two approaches to testing hypothesized rules, either of which can be used with our learning-based approach to rule extraction.

For the purposes of this paper, we define the rule-extraction task as follows:

> *Given a trained neural network and the examples used to train it, produce a concise and accurate symbolic description of the network.*

We consider only domains in which the network has discrete output classes and input features that are either Boolean or nominal valued.

[1] In a knowledge-based network (Towell & Shavlik, in press), the topology and initial weights of the network are specified by a domain theory consisting of symbolic inference rules.



extracted rules: $f \longleftarrow a$
$f \longleftarrow b \wedge \neg c$

Figure 1: **A Network and Extracted Rules.** The network has three Boolean inputs and one Boolean output. The rules describe the conditions under which an instance is predicted to be a member of class f (i.e., the output unit is active).

Figure 1 illustrates the task of rule extraction for a simple network. This one-layer network (i.e., a perceptron) has three Boolean input features and one Boolean output feature. Any network, such as this, which has discrete output classes and discrete-valued input features can be exactly described by a set of symbolic *if-then* rules. The rules specify input-feature values that, when satisfied, guarantee a given output state. In the example of Figure 1, the rules describe the conditions under which the output unit has an activation of unity. The output unit's activation, a_i, is calculated as follows:

$$a_i = \begin{cases} 1 & if\ \sum_j w_{ij} a_j > \theta \\ 0 & otherwise \end{cases}$$

where a_j is the activation of input unit j, w_{ij} is the weight from unit j to the output unit i, and θ_i is the threshold of the output unit.

Several research groups have investigated rule-extraction methods which operate by conducting a breadth-first search through a space of possible conjunctive rules (Fu, 1991; Gallant, 1993; Saito & Nakano, 1988; Thrun, 1993). Figure 2 shows such a search space for the network in Figure 1. Each node in the space corresponds to a possible rule, and the arcs indicate specialization relationships (in the downward direction) between nodes. The node at the top of the graph represents the most general rule (i.e. all instances are members of the class f), and the nodes at the bottom level represent specific instances. Unlike an ordinary breadth-first search which continues until a goal node is found, a rule-extraction search continues until all (or many) goal nodes (i.e., all of the maximally-general rules) have been found.

Visiting a node in the search space involves testing the rule that corresponds to the node to see if it accurately describes the network. A rule is tested by considering

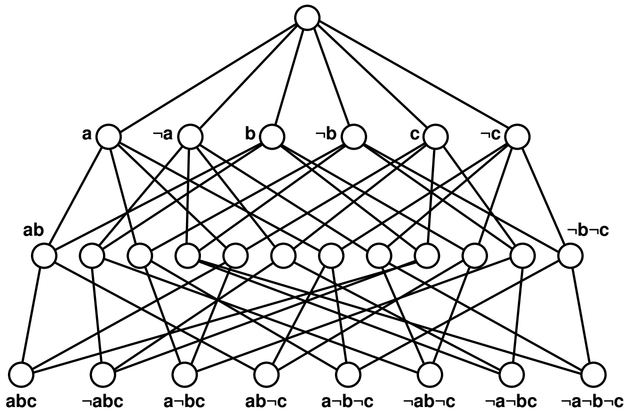


Figure 2: **A Rule Search Space.** Each node in the space corresponds to a possible conjunctive rule for the network in Figure 1. The top node represents the most general rule and the nodes at the bottom represent specific instances. (For clarity, not all nodes are labelled.)

the constraints that it places on the network's input and output units. For example, the rule $f \longleftarrow a$ in Figure 1 specifies that if a has an activation of 1, then the output unit, f, will have an activation of 1, regardless of the activations of b and c. This rule is tested by considering the case where the *undetermined* features, b and c, take on values that least support activating the output unit. In this example, b is minimally supportive when it has an activation of 0 (i.e., it is *false*), and c is minimally supportive when it has an activation of 1 (i.e., it is *true*). The rule is determined to be true because, even when b and c take on these values, f still has an activation of 1.

Testing hypothesized rules for multilayer networks is somewhat problematic since the relationship between a given input and a given output is not necessarily monotonic, but instead, may depend on the values of other input features. In other words, changing the activation of an input may increase the output unit's activation in some cases, and decrease it in other cases. There are two basic approaches to testing rules for multilayer networks, which we discuss below.

One approach to testing rules for a multilayer network is to treat the network as a collection of perceptrons, and to extract rules for each hidden and output unit separately (Craven & Shavlik, 1993; Fu, 1991; Gallant, 1993; Saito & Nakano, 1988; Towell & Shavlik, 1993). We will refer to this as the *decompositional* approach. In this approach, the rules for each unit are expressed in terms of the units that feed into it. An advantage of the decompositional approach is that it produces "intermediate terms" which may result in simpler descriptions. A disadvantage of this method, however, is that it requires that the hidden units of the network be approximated by threshold units, and thus the extracted rules may not provide an accurate representation of the network.

The other approach to testing rules for multilayer networks is a technique developed by Thrun (1993) called *validity-interval analysis* (VIA). Validity-interval analysis enables the extraction of rules that directly map inputs to outputs for a multilayer network. VIA employs linear programming to determine if a set of constraints on a network's activation values is consistent. To test a rule using VIA, the rule is first negated and then the network's input and output unit activations are constrained according to the negated rule. Given these constraints, validity-interval analysis calculates the activation range for each unit in the network. If an inconsistency is found, the rule is determined to be valid, since the inconsistency indicates that there is no way to satisfy the negation of the rule.

An advantage of validity-interval analysis over the decompositional approach is that it does not require that hidden units be approximated by threshold units. However, it suffers from the drawback that it often will not find maximally-general rules because it makes the (sometimes incorrect) assumption that the activations of the hidden units are independent.

The number and structure of the search spaces used by the decompositional and the VIA approaches also differ. The decompositional approach requires searching a separate rule space for each hidden and output unit in the network. The VIA approach, on the other hand, involves only searching a space for each output class. Because the decompositional approach always deals with a single layer of a network at a time, however, its search spaces may be smaller for some problems. To see this point, consider the network shown in Figure 1. Because the relationship between each input and the output is monotonic, it is not necessary to consider rules that have $\neg a$, $\neg b$, or (non-negated) c in them.

Search-based approaches to rule extraction are adequate for problem domains that do not require exploring large search spaces. However, the number of nodes in a rule space is exponential in the number of input features and consequently, the search combinatorics are overwhelming for many real-world problems. One approach to reducing the complexity of rule searches is to cluster the network's weights into equivalence classes and extract *M*-of-*N* rules (Craven & Shavlik, 1993; Towell & Shavlik, in press). As mentioned in Section 1, however, this approach requires a special training technique, may not enable a sufficiently accurate description of the network to be extracted, and requires that the hidden units be approximated as threshold units. Another approach to reducing search complexity is to limit the depth to which the search space is explored (Fu, 1991; Gallant, 1993; Saito & Nakano, 1988). We have found that in some domains, however, rules are found quite deep in the search space, and hence this method is not effective.

3 RULE EXTRACTION AS LEARNING

In this section, we introduce an approach to extracting conjunctive rules from trained networks that does not employ a top-down search, but instead uses a learning process driven by sampling and queries. Our hypothesis is that this method will typically require less computation than search-based rule-extraction methods to acquire rule sets that model networks to a comparable degree of accuracy.

3.1 AN ALGORITHM

Our approach involves viewing rule extraction as a learning task where the target concept is the function computed by the network and the input features are simply the network's input features. The approach uses two different *oracles* that are able to answer queries about the concept being learned. The EXAMPLES oracle produces, on demand, training examples for the rule-learning algorithm. The SUBSET oracle answers *restricted subset* queries (Angluin, 1988). It takes two arguments: a class label and a conjunctive rule. SUBSET returns *true* if all of the instances that are covered by the rule are members of the given class, and *false* otherwise.

Our algorithm for extracting conjunctive rules from trained neural networks is outlined in Table 1. It is an adaptation of the classical algorithm for PAC-learning monotone DNF expressions (Valiant, 1984). The algorithm maintains a DNF expression[2] for each class. The algorithm repeatedly queries EXAMPLES, and then determines the class of each returned example. If an example is not covered by the current DNF expression for the class then it serves as the basis for a new rule (i.e. a new term in the DNF expression). The new rule is initialized as the conjunction of all of the feature values of the returned example. This rule is then generalized by repeatedly dropping an antecedent (i.e. making one of the features undetermined) and then calling SUBSET to ascertain if the rule still agrees with the network. If SUBSET returns *true*, then the dropped antecedent is left off of the rule (i.e. the feature is left undetermined), otherwise it is put back on the rule.

In some cases, it may not be necessary to extract rules for every output class. For example, in the case of a problem that involves only two classes, say *positive* and *negative*, we can extract rules that describe only the positive class and use the closed-world assumption to identify members of the negative class. In such cases, the EXAMPLES oracle can be instructed to "throw away" examples that belong to the class not being covered.

[2]Note that each term in a DNF expression can be trivially converted into a conjunctive rule.

Table 1: **Conjunctive Rule Extraction Algorithm.**

```
/* initialize rules for each class */
for each class c
    R_c := ∅
repeat
    e := EXAMPLES()
    c := classify(e)
    if e not covered by R_c then
        /* learn a new rule */
        r := conjunctive rule formed from e
        for each antecedent r_i of r
            r' := r but with r_i dropped
            if SUBSET(c, r') = true then r := r'
        R_c := R_c ∨ r
until stopping criterion met
```

We now explain how the two oracles work in practice. Recall that the SUBSET oracle accepts a class label c and a rule r, and returns *true* if all instances covered by r are classified as members of class c by the network. This operation is equivalent to testing a rule in a search-based rule extraction method. Consequently, it can be implemented in the same way: we can use either the decompositional approach where rules are extracted for each hidden and output unit individually, or we can use Thrun's validity-interval analysis technique. The implementation of the SUBSET oracle is different in each case, but the learning algorithm remains fundamentally the same.

Recall that the function of the EXAMPLES oracle is to provide an unlimited source of examples for the learning algorithm shown in Table 1. Initially, the EXAMPLES oracle can return members of the network's training set. At some point, however, EXAMPLES will exhaust the training set. After this has happened, the EXAMPLES oracle finds examples by randomly sampling the instance space.

In some cases, it may be desirable to have the EXAMPLES oracle return only examples that belong to a particular class. This can be done for the decompositional rule-extraction approach using the hill-climbing algorithm shown in Table 2. This algorithm first randomly assigns a value to each feature, and then tests the newly-constructed example to see if it is a member of, say, the positive class. If it is not a positive example, then a random order is imposed on the possible values of all the features, and these values are considered in order. A feature's assigned value is changed to a considered value if doing so increases the total input to the output unit. This approach will not work for multilayer networks when validity-interval analysis is being used since, in this case, there is no guarantee that hill-climbing will find a positive example. In Section 5, however, we discuss a proposed approach to a

Table 2: **An Examples oracle.** This oracle operates on a single-layer network and returns only positive examples.

```
/* create a random example */
for each feature e_i with possible values v_i1,...,v_in
    e_i := randomly-select(v_i1,...,v_in)
calculate the total input s to output unit
if s ≥ θ then return e
impose random order on all feature values
/* consider the values in order */
for each value v_ij
    if changing feature e_i's value to v_ij increases s
        e_i := v_ij
    if s ≥ θ then return e
```

more directed EXAMPLES oracle for the general case.

Note that the algorithm shown in Table 1 employs a stopping criterion to determine when a set of extracted rules provides a sufficiently-good model of a network. There are several reasonable criteria that could be used here. For example, a tuning set might be held aside to estimate the accuracy of the extracted rules. Alternatively, we could use a *patience* criterion (Fahlman & Lebiere, 1989) that causes the procedure to quit after a certain number of iterations have resulted in no new rules. In our experiments we do not define a stopping criterion, but instead, we show for different methods how the accuracy of extracted rule sets changes as a function of the amount of computation.

It is important to note that the fundamental difference between the search-based approach to rule extraction and our learning-based approach is the way in which the space of rules is explored. Both approaches can be used with either the decompositional method or validity-interval-analysis to test hypothesized rules.

3.2 EVALUATING THE ALGORITHM

In order to evaluate our approach, we empirically compare it to a search-based method similar to those described in Section 2. Our hypothesis is that our learning approach requires less computation than the search-based method to get rule sets that model networks to a comparable degree of *fidelity*. We measure fidelity by comparing the classification performance of a rule set to the network from which it was extracted; the fidelity of a rule set is the fraction of examples on which it agrees with the network. We make this comparison using examples that the EXAMPLES oracle is not allowed to access (i.e. test sets), so that we can see how well our extracted rule sets (i.e. our learned concept descriptions) generalize.

Although our method is applicable to multilayer networks, we use perceptrons in our experiments. Our interest is in measuring the relative efficiency of exploring a rule space using our approach versus a search-based approach; single-layer networks provide an adequate milieu for this experiment.

We evaluate the two approaches using the *promoter* domain (Towell et al., 1990). Promoters are short sequences in DNA that occur before genes and play a critical role during gene transcription. Each feature in this domain represents a position in a DNA sequence; thus each feature takes one of the values in $\{A, G, C, T\}$. In our experiments, we do not use all 57 of the available features, but instead we use only a subset of 8 contiguous features.[3] This reduced subset is small enough that we can, in a reasonable amount of time, run the search-based method until it has found all of the rules in its search space. The promoter dataset we use comprises 468 examples, half of which are positive examples (promoters).

We train 10 perceptrons, using a different training set (consisting of 90% of the total examples) for each. The examples left out of each training set are used to measure the fidelity of extracted rule sets. The perceptrons have 32 (8 features $\times$ 4 values) input units and one output unit. We then apply both our learning-based extraction method and the conventional search-based method to the trained perceptrons. We extract rules that describe the conditions under which the output unit is active, and hence predict when the given sequence is a *promoter*.

The search method that we use conducts a breadth-first search through a space of possible conjunctive rules. Unlike the graph shown in Figure 2, we organize the rule space as a tree so that each node in the search space can not be visited more than once. In order to ensure that the extracted rules are maximally general, whenever a valid rule is found, it is compared to the rules found at previous levels and retained only if it is not more specific than any of them. Additionally, we employ an optimization that enables some branches of the tree to be pruned: whenever a rule is tested and rejected during the search, we also determine if there is any sequence of antecedents that could be added to the rule to make it valid. If there is no such sequence of antecedents, then we need not explore any descendents of this node. The computation involved in this determination is essentially the same as that involved in testing a rule: the difference is that instead of assuming that undetermined features will take on values that contribute minimally to activating an output unit, we assume that they will take on values that contribute maximally.

The computation required by the search algorithm can be quantified by counting two operations:

[3] The features we use are nucleotides from the so-called *-35 region*; it is these features that are thought to be among the most important in defining the concept of a *promoter*.

- the number of nodes tested during the search,
- subsumption-check comparisons to previously extracted rules.

Similarly, the computation required by our learning-based algorithm can be quantified by counting three operations:

- calls to the EXAMPLES oracle,
- calls to the SUBSET oracle,
- subsumption-check comparisons to the individual terms of R_c, the DNF expression for class c.

Except for calls to the EXAMPLES oracle, the computational complexity of each of these operations is $O(f)$,[4] where f is the number of input features. The computational complexity of calling the EXAMPLES oracle is $O(f)$ when the oracle is used to produce examples of any class. We use the EXAMPLES oracle to produce only positive examples, however, and thus its time complexity is $O(v \log v)$, where v is the total number of values for all input features. In our experiments, we measure the amount of computation performed by counting operations in terms of $O(f)$ units. Thus visiting a node in the search space or making a subsumption check counts as 1 unit. Calling the EXAMPLES oracle counts as $\frac{v \log v}{f}$ units (in our domain, $\frac{v \log v}{f} = 20$). Since these operations indicate the number of times that the non-constant-time parts of the algorithm are executed, their total count provides a reasonable basis for comparing our learning algorithm against the search method.

Figure 3 shows the test-set fidelity of extracted rule sets for the two approaches, averaged over the ten perceptrons. This figure is analogous to a generalization curve for a conventional learning system. The x-axis indicates the number of operations (EXAMPLES *calls* + SUBSET *calls* + *comparisons* for the learning method, *nodes visited* + *comparisons* for the search-based method) for the two approaches; note that the scale used for this axis is logarithmic. The y-axis denotes the averaged fidelity of the rule sets measured on the test sets. Both methods initially have a fidelity measure of about 50% since their empty rule sets never predict the *promoter* class. While the search-based approach requires a fair amount of time before it finds any rules, our learning method immediately begins extracting rules, and thus its fidelity quickly improves. In fact, for most values on the y-axis, the search-based method requires orders of magnitude more operations to achieve the same fidelity. Our learning-based method explores many fewer nodes than the search-based method; as its extracted rule sets grow, most its effort is expended comparing training examples to

[4]The computational complexity of SUBSET calls may be more than $O(f)$ when validity-interval analysis is used with multilayer networks.

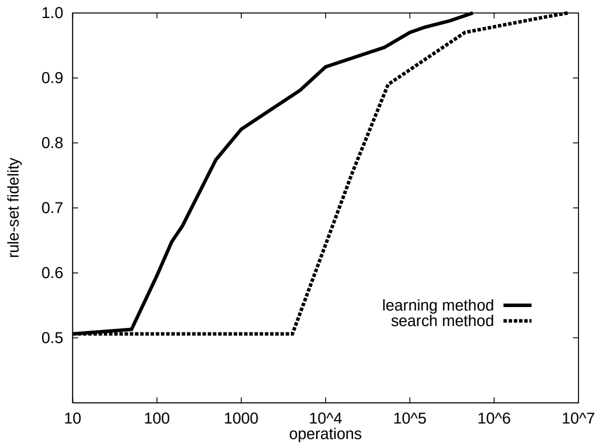


Figure 3: **Conjunctive rule-set fidelity.** This figure shows the averaged fidelity of extracted conjunctive rule sets after a given number of operations. The x-axis, which has a logarithmic scale, shows the number of operations (as described in the text). The y-axis reports the fidelity (measured on test sets) of the extracted rules.

extracted rules to see if the examples are covered. The fidelity of both approaches increases monotonically because all acquired rules are sound with respect to the network. We have observed that, for a given level of fidelity, the sizes of the rule sets extracted by the two approaches are comparable.

To understand why our learning approach outperforms the search-based extraction method in this problem domain, it is helpful to consider the space of rules that is being explored. Whereas the search method explores this space from the top down, our learning approach explores the space from the bottom up. The effectiveness of the search approach is determined by the depth to which it needs to explore the space in order to find rules. Since the computational complexity of this search grows exponentially in depth, it is expensive for the search method to find rules that have more than a few antecedents. Our learning method, on the other hand, is able to efficiently find rules with many antecedents. The positive training examples serve to identify terminal nodes of the search space that need to be covered by rules. Using these terminal nodes as initial rules, our learning method uses a polynomial-time algorithm to traverse upward in the search space until maximally-general rules are found.

A disadvantage of our approach is that, because it stochastically samples the instance space, it may take a very long time to find all of the maximally-general rules – especially rules that cover only a few instances. The search-based approach, on the other hand, methodically explores the rule space, and thus may be more effective for networks that have only a small number of features, or result in rules that have few antecedents.

4 EXTRACTING *M*-of-*N* RULES

In this section we generalize the algorithm presented in Section 3 to extract *M*-of-*N* rules from trained networks. *M*-of-*N* rules are better suited to describing neural networks than are conjunctive rules (Towell & Shavlik, 1993) because they more closely match the inductive bias of units in a neural network. Hence, fewer *M*-of-*N* rules than conjunctive rules are usually required to describe a network.

The *M*-of-*N* rules that our algorithm extracts consist of conjunctions of one or more *M*-of-*N* *terms*. An *M*-of-*N* term is satisfied when at least M of its N antecedents are satisfied. For example, the term *2-of-*$\{a,\ b,\ c\}$ is logically equivalent to $(a \wedge b) \vee (a \wedge c) \vee (b \wedge c)$. An *M*-of-*N* rule is satisfied when all of its terms are satisfied.

Table 3 outlines the algorithm we use to extract *M*-of-*N* rules from trained networks. In the same manner as the algorithm presented in Table 1, the first step is to learn a conjunctive rule using the instance supplied by the EXAMPLES oracle. The algorithm then makes this conjunction into a trivial *M*-of-*N* rule for which M is set to N. The next step is to generalize this rule through the application of two operators:

- *add-value*: generalizes an *M*-of-*N* term by adding to it a feature value that is not already present in the set of antecedents.
- *new-term*: takes an existing *M*-of-*N* term and splits it into two terms of the form *L*-of-*L* and $(M - L)$-of-$(N - L)$. For example, *3-of-*$\{a,\ b,\ c\} \Longrightarrow$ *2-of-*$\{a,\ b\} \wedge$ *1-of-*$\{c\}$.

The *add-value* operator may either add another possible value for a feature already in the antecedent set (i.e., create an *internal disjunction*, Michalski, 1983), or add a feature that is not yet represented.

The *new-term* operation, by itself, is not able to generalize a term; therefore this operator is always used in conjunction with the *add-value* operator. Additionally, *new-term* may have the undesired effect of specializing a rule unless it is applied only to terms for which M is equal to the number of features represented in the set of antecedents (i.e., the only disjunctions represented by the term are internal disjunctions). Therefore, we allow the *new-term* operator to be applied only when it meets this condition. Another restriction that we place on this operator is that it cannot arbitrarily partition an *M*-of-*N* term, because there are $2^{N-1} - 1$ possibilities for a term with N antecedents. Since we are extracting rules from perceptrons in our experiments, we are able to order the antecedents according to the magnitude of their weights, and require that newly-created terms maintain the order of the antecedents. This constraint reduces the number of possible partitions to $N - 1$.

Table 3: ***M*-of-*N* Rule Extraction Algorithm.**

```
for each class c
    R_c := ∅
repeat
    e := EXAMPLES()
    c := classify(e)
    if e not covered by R_c then
        learn conjunctive rule, r, as in Table 1
        trivially convert r to a M-of-N rule
            where M = N
        do
            r' := result of applying add-value or
                  new-term to r
            if SUBSET(c, r') then r := r'
        while ∃ additional operator applications
        R := R ∨ r
until stopping criterion met
```

The *add-value* and *new-term* operators are used to successively generalize a given *M*-of-*N* rule until further generalizations result in a rule that is not consistent with the network. On each iteration of the loop, one of the possible operator applications is selected and applied to the given rule. The SUBSET oracle is used to determine if a generalized rule is consistent with the network. The loop continues until all operator applications have been tried and the rule cannot be further generalized. The algorithm is quite naive in how it selects operator applications; an area for future work is to investigate heuristics for guiding this selection.

As with conjunctive queries, the SUBSET oracle can be implemented in one of two ways, depending upon whether the decompositional approach or validity-interval analysis is being used. With the decompositional approach, in addition to calculating the minimum contribution of undetermined features, the SUBSET oracle must determine which M antecedents minimally satisfy a term. That is, since the oracle wants to determine if a given unit will be active whenever the terms of the rule are satisfied, it must consider the case where each term is satisfied by those antecedents that contribute the least to activating the unit. A SUBSET oracle for *M*-of-*N* rules can also be implemented for the validity-interval-analysis approach; Thrun (1993) has described how *M*-of-*N* constraints can be incorporated into the linear programs that test hypothesized rules.

To evaluate the *M*-of-*N* version of our algorithm, we extract rules from the perceptrons used in the experiment in Section 3. We count the number of "operations" as before, except that we adjust our accounting for calls to SUBSET to reflect the fact that queries about *M*-of-*N* rules are more expensive to handle than queries about conjunctive rules. Whereas the complexity of answering a query about a conjunctive rule

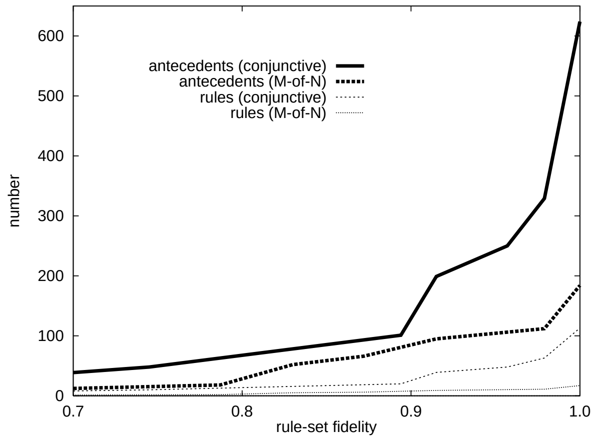


Figure 4: **Rule-set sizes.** This figure shows the average number of rules and antecedents in the rule sets extracted by our conjunctive and M-of-N learning algorithms. The x-axis reports the fidelity of the extracted rule sets and the y-axis indicates rule counts and the total number of antecedents summed over all the rules in a rule set.

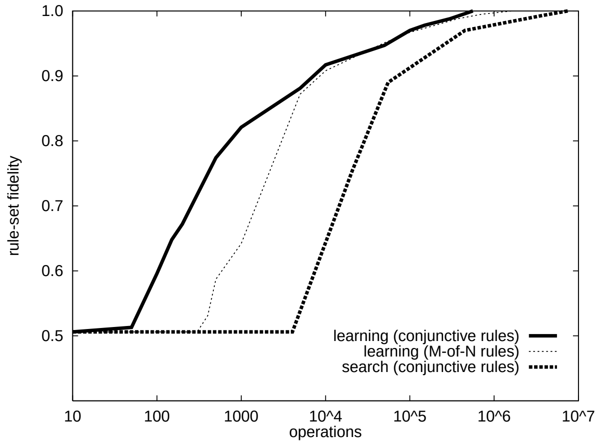


Figure 5: ***M*-of-*N* rule-set fidelity.** This figure shows the averaged fidelity of M-of-N rule sets after a given number of operations. The x-axis, which has a logarithmic scale, shows the number of operations (as described in the text). The y-axis reports the fidelity of the extracted rule sets. Also shown are the fidelity curves for conjunctive rules for both the search method and our learning approach.

is $O(f)$, where f is the number of features, the complexity of answering a query about an M-of-N term is $O(f + M \log f)$, since we have to find the M smallest-weighted antecedents (Knuth, 1981). We therefore calculate the cost of a SUBSET query in this experiment by summing $1 + \frac{M \log f}{f}$ for every term in an M-of-N rule (recall that the basic unit of our accounting is an $O(f)$ operation).

The primary advantage of extracting M-of-N rules is that the rules are typically much more concise than conjunctive rules. Figure 4 reports the average number of rules and antecedents extracted for both conjunctive and M-of-N rule sets. This figure indicates that a much smaller set of M-of-N rules is able to describe a network at a given level of fidelity.

Figure 5 shows the fidelity of extracted M-of-N rule sets, averaged over the ten perceptrons. The curves for conjunctive rules from Figure 3 are also shown. As can be seen in the figure, our algorithm for extracting M-of-N rules is initially not as efficient as the algorithm for conjunctive rules, but asymptotically, their performance is about the same. The M-of-N algorithm expends more effort in generalizing its training examples, but it requires fewer operations than the conjunctive-rule algorithm to test if an example is already covered, since it typically has fewer rules to test against.

5 FUTURE WORK

There are three areas in which we plan to further develop our approach: implementing a more directed EXAMPLES oracle, finding better intermediate terms in extracted rules, and handling real-valued features.

The EXAMPLES oracle presented in this paper selects training examples by uniformly sampling the instance space. This method is inefficient in that the EXAMPLES oracle does not direct its attention to the regions of the instance space that have not yet been covered by rules. We plan to develop and evaluate an EXAMPLES oracle that uses the current set of extracted rules to influence sampling. Our proposed approach would use previously-extracted rules to formulate a propositional satisfiability (SAT) problem. Every solution to such a SAT problem is an example that is not covered by any of the extracted rules. Although the general SAT problem is NP-hard, we hypothesize that a greedy local-search algorithm, such as GSAT (Selman et al., 1992), will be able to efficiently find solutions in cases where there are many uncovered examples. Only when there are very few uncovered examples should it be difficult to find one; an EXAMPLES oracle that randomly generates training examples, however, is likely to be ineffective in these situations as well.

A second area of planned research is to investigate algorithms that search for useful *intermediate terms* during the rule-extraction process. Intermediate terms can aid rule-set comprehensibility by improving the concision of extracted descriptions, and by better illustrating the structure of learned concepts. The decompositional approach to rule extraction introduces intermediate terms corresponding to each of the hidden units in the network. This approach, however, involves the assumptions that each hidden unit in the network behaves like a Boolean variable, and the individual hidden units of the network correspond to concepts that are meaningful in the context of the domain. Neural networks, however, often learn *distributed rep-*

resentations (Hinton, 1986) in which each concept is encoded by the activations of many hidden units, and each hidden unit plays a part in representing many different concepts. Given such representations, the "right" intermediate terms might represent patterns of activity across groups of hidden units, instead of individual hidden units themselves.

Finally, in order to make our approach applicable to more problem domains, we plan to extend it to handle real-valued features. Other researchers have already investigated this issue to some extent (Gallant, 1993; Thrun, 1993); we plan to incorporate their techniques into our algorithm.

6 CONCLUSIONS

We have described a novel approach that casts the problem of extracting symbolic rules from trained neural networks as a learning task. Our approach exploits both training examples and queries to learn concept descriptions that accurately describe trained neural networks. Our experiments demonstrate that this approach can be more efficient than conventional search-based techniques. We have also described how our approach can be generalized to extract *M*-of-*N* rules. We believe that our approach provides a promising advance toward the goal of readily interpreting trained neural networks.

Acknowledgements

We wish to thank Rich Maclin for providing helpful comments on an earlier version of this paper, and Nick Street and Sebastian Thrun for interesting discussions regarding this work. This research was partially supported by DOE Grant DE-FG02-91ER61129, ONR Grant N00014-93-1-0998, and NSF Grant IRI-9002413.

References

Angluin, D. (1988). Queries and concept learning. *Machine Learning*, 2:319–342.

Craven, M. W. & Shavlik, J. W. (1993). Learning symbolic rules using artificial neural networks. In *Proceedings of the Tenth International Conference on Machine Learning*, (pp. 73–80), Amherst, MA. Morgan Kaufmann.

Fahlman, S. & Lebiere, C. (1989). The cascade-correlation learning architecture. In Touretzky, D., editor, *Advances in Neural Information Processing Systems (volume 2)*. Morgan Kaufmann, San Mateo, CA.

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. AAAI/MIT Press.

Gallant, S. I. (1993). *Neural Network Learning and Expert Systems*. MIT Press, Cambridge, MA.

Hinton, G. E. (1986). Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, (pp. 1–12), Amherst, MA. Erlbaum.

Hunter, L. & Klein, T. (1993). Finding relevant biomolecular features. In *Proceedings of the First International Conference on Intelligent Systems for Molecular Biology*, (pp. 190–197), Bethesda, MD. AAAI Press.

Knuth, D. E. (1981). *The Art of Computer Programming (volume 3)*. Addison-Wesley, second edition.

Lapedes, A., Barnes, C., Burks, C., Farber, R., & Sirotkin, K. (1989). Application of neural networks and other machine learning algorithms to DNA sequence analysis. In Bell, G. I. & Marr, T. G., editors, *Computers and DNA (volume VII)*. Addison-Wesley.

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

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

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

Selman, B., Levesque, H., & Mitchell, D. (1992). A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, (pp. 440–446), San Jose, CA. AAAI/MIT Press.

Thrun, S. B. (1993). Extracting provably correct rules from artificial neural networks. Technical Report IAI-TR-93-5, Institut für Informatik III, Universität Bonn.

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

Towell, G. & Shavlik, J. (in press). Knowledge-based artificial neural networks. *Artificial Intelligence*.

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

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, 27:1134–1142.

Wolberg, W. H., Street, W. N., & Mangasarian, O. L. (in press). Machine learning techniques to diagnose breast cancer from image-processed nuclear features of fine needle aspirates. *Cancer Letters*.