Appears in the working notes of the *IJCAI '95 Workshop on Comprehensibility in Machine Learning.*

Extracting Comprehensible Concept Representations from Trained Neural Networks

Mark W. Craven **Jude W. Shavlik**

Computer Sciences Department
University of Wisconsin
1210 West Dayton St.
Madison, Wisconsin 53706
U.S.A.

phone: +1 (608) 263-0475
email: {craven, shavlik}@cs.wisc.edu

Abstract

Although they are applicable to a wide array of problems, and have demonstrated good performance on a number of difficult, real-world tasks, neural networks are not usually applied to problems in which comprehensibility of the acquired concepts is important. The concept representations formed by neural networks are hard to understand because they typically involve distributed, nonlinear relationships encoded by a large number of real-valued parameters. To address this limitation, we have been developing algorithms for extracting "symbolic" concept representations from trained neural networks. We first discuss why it is important to be able to understand the concept representations formed by neural networks. We then briefly describe our approach and discuss a number of issues pertaining to comprehensibility that have arisen in our work. Finally, we discuss choices that we have made in our research to date, and open research issues that we have not yet addressed.

1 Introduction

Neural networks provide an appealing approach to learning because they are applicable to a large class of problems, and because they have demonstrated good generalization performance in a variety of domains (Sejnowski & Rosenberg, 1987; Tesauro & Sejnowski, 1989; Pomerleau, 1991; Smyth & Mellstrom, 1992). A limitation of neural networks, however, is that the concept representations they form are extremely difficult for humans to understand. Because of this limitation, it is commonly believed in the machine-learning community that neural networks are not well suited for domains in which comprehensibility of learned concepts is important. To address this limitation of neural networks, we have been developing methods for extracting from trained networks, comprehensible concept descriptions like those produced by common symbolic learning algorithms (e.g., Michalski, 1983; Quinlan, 1993). The goal of this research is to mitigate the comprehensibility advantage that symbolic methods have over neural learning algorithms.

In this paper, we describe our research in developing methods for extracting symbolic representations from neural networks (Craven & Shavlik, 1994; Craven & Shavlik, 1995), and discuss issues of comprehensibility that have arisen in this work. We first provide motivation for our research by discussing in Section 2 why it is important to be able to understand trained neural networks. In Section 3 we summarize our approach to understanding neural networks, and present some empirical results. Section 4 then presents a discussion of issues related to comprehensibility that we have encountered in our research. Section 5 describes related work, and Section 6 provides conclusions.

2 Neural Networks and Symbolic Representations

One aspect that distinguishes various approaches to concept learning is the way in which learned concepts are represented. A concept representation learned by a neural network is defined by (a) the topology of the network, (b) the activation functions used for the hidden and output units, and (c) the real-valued parameters associated with the network connections (i.e., the weights) and units (e.g., the biases of sigmoidal units). Concept representations learned by neural networks are difficult to comprehend for several reasons. First, typical networks have hundreds or thousands of real-valued parameters. Moreover, in multi-layer networks, these parameters may represent nonlinear, nonmonotonic relationships between the input features and the output categories. Although hidden units in multi-layer networks can be thought of as representing "constructed" features, they are difficult to understand themselves because networks commonly learn *distributed representations* (Hinton, 1986). In a distributed representation, each meaningful concept is encoded by the activations of many hidden units, and each hidden unit plays a part in representing many different concepts. Additionally, hidden units may have graded activations, meaning that they are not completely "on" or completely "off" in response to a given instance.

2.1 Why Use Neural Networks?

There are several reasons why it is sometimes preferable to use neural networks for a concept learning task rather than a symbolic learning method. A major reason is that neural networks offer better generalization performance than common symbolic learning algorithms for some problems (Atlas et al., 1989; Fisher & McKusick, 1989; Weiss & Kapouleas, 1989; Shavlik et al., 1991). When putting learning systems into practice, generalization performance is usually the overriding criterion for selecting one algorithm over another.

Another reason for preferring neural networks over symbolic algorithms is that they can be applied to some problems for which symbolic learning algorithms are poorly suited. For example, the technique of *Q-learning* (Watkins, 1989) requires that the learning system represent concepts as continuous-valued functions, and that these concepts be updated after each training instance. Few symbolic learning algorithms are able to meet both of these requirements. Sequential and temporal prediction tasks provide another type of problem for which neural networks are often more appropriate than symbolic algorithms. *Recurrent networks* (Jordan, 1986; Pineda, 1987), which are often applied to these problems, employ input units that represent the state of the hidden units at the previous time step. Recurrent networks use their own representations of the problem at the previous time step to generate predictions for the current time step. Moreover, such networks are able to propagate credit and blame back to representations that were formed in previous time steps (because their output functions are differentiable with respect to previous representations). No symbolic learning algorithm provides these appealing representational aspects.

2.2 Comprehensibility Is Important

An important criterion by which a machine learning algorithm should be judged is the comprehensibility of the representations it forms. That is, does the algorithm encode the information it learns in such a way that it may be inspected and understood by humans? The importance of this criterion is argued by Michalski (1983) in his *comprehensibility postulate*:

> *The results of computer induction should be symbolic descriptions of given entities, semantically and structurally similar to those a human expert might produce observing the same entities. Components of these descriptions should be comprehensible as single 'chunks' of information, directly interpretable in natural language, and should relate quantitative and qualitative concepts in an integrated fashion.* (pg. 122)

There are a number of reasons why this is an important criterion.

- **Validation**. If the designers and end-users of a learning system are to be confident in the performance of the system, they must understand how it arrives at its decisions. The ability to inspect a learned concept definition is important in many domains, such as medical diagnosis (e.g., Wolberg et al., 1994).

- **Discovery**. Learning systems may also play an important role in the process of scientific discovery. A system may discover salient features and relationships in the 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).

- **Explanation**. In some domains, it is not necessary to have a holistic description of the learning system's concept representation, but it is desirable to be able to explain classifications of individual input patterns (Gallant, 1993). If the concept representation is understandable in such a domain, then explanations of classifications made on a particular cases can be garnered.

- **Improving generalization**. The feature representation used for a concept learning task can have a significant impact on how well an algorithm is able to learn and generalize (Flann & Dietterich, 1986; Craven & Shavlik, 1993c). Learned concept representations that can be understood and analyzed may provide insight into devising better feature representations.

- **Refinement**. Some researchers use concept learning systems to refine approximately-correct domain theories (Pazzani & Kibler, 1992; Ourston & Mooney, 1994; Towell & Shavlik, 1994). In order to complete the theory-refinement process, it is important to be able to express, in a comprehensible manner, the changes that have been imparted to the theory during learning.

In addition to issues of comprehensibility, another motivation for extracting symbolic representations from neural networks is to facilitate flexibility in how learned knowledge is used. Concept representations in neural networks are effectively *procedural representations* rather than *declarative representations*. In a procedural representation, knowledge is inextricably tied to a particular procedure that uses it. A disadvantage of procedural representations is that they limit encoded knowledge to being used by a particular procedure in a particular way. A declarative representation, on the other hand, represents knowledge in a general manner so that it may be used by different procedures in different contexts. For example, one reason why Mitchell et al. (1994) decided not to use neural networks for their calendar-management system is because the "monolithic" representations of neural networks would not allow them to retain only parts of learned concept representations. Because their task involves time-varying target concepts, this was an important consideration. However, given a method for extracting symbolic descriptions of the concepts represented by trained networks, the networks' procedural representations can be converted into declarative ones.

3 An Approach to Extracting Symbolic Representations from Trained Neural Networks

There are several possible approaches to understanding the representations learned by neural networks. Our approach to understanding trained networks is to extract symbolic concept descriptions from them. The task is as follows: given a trained network and the data on which it was trained, produce a concept description that is comprehensible, yet classifies instances in (nearly) the same way as the network.

We have developed a number of algorithms for extracting symbolic descriptions of trained networks. In previous work, we first developed an algorithm for extracting inference rules from *knowledge-based neural networks*[1] (Towell & Shavlik, 1993), and then generalized the this algorithm so that it could be applied to ordinary networks when a special training procedure was used (Craven & Shavlik, 1993a). Recently, we have developed a more general approach to the extraction task (Craven & Shavlik, 1994; Craven & Shavlik, 1995) which does not place any requirements on either the architecture of the network or its training method.

Our new algorithm views the task of extracting a comprehensible concept description from a trained network as an inductive learning problem. In this learning task, the target concept is the function represented by the network, and the concept description produced by our learning algorithm is a decision tree that approximates the network. However, unlike most inductive learning problems, we have available an *oracle* that is able to answer queries during the learning process. Since the target function is simply the concept represented by the network, the oracle uses the network to answer to queries. The advantage of learning with queries, as opposed to ordinary training examples, is that they can be used to garner information precisely where it is needed during the learning process.

We now describe our new algorithm in some detail, and then present experiments that we have conducted to evaluate the algorithm.

3.1 TREPAN: An Algorithm for Extracting Decision Trees

Our algorithm, called TREPAN,[2] is shown in Table 1. TREPAN is similar to conventional decision-tree induction algorithms, such as CART (Breiman et al., 1984), ID3 (Quinlan, 1986), C4.5 (Quinlan, 1993), and ID2-of-3 (Murphy & Pazzani, 1991), which learn directly from a training set. These algorithms build decision trees by recursively partitioning the input space. Each internal node in such a tree represents a partition of some part of the input space, and each leaf represents a predicted class. Our method is substantially different from these conventional algorithms in number of respects, which we detail below.

The Oracle. The role of the oracle is to determine the class (as predicted by the network) of each instance that is presented as a query. Queries to the oracle, however, do not have to be complete instances, but instead can be specified as constraints on the values that the features can take. In the latter case, the oracle generates a complete instance by randomly selecting a value for each feature, while ensuring that the constraints are satisfied. Random values for a given feature are generated using the distribution of values for that feature over the training set. As shown in Table 1, the oracle is used for three different purposes: to determine the class labels for the network's training examples; to determine the class labels for the tree's leaves; and to select the

[1]In a knowledge-based network, the topology and initial weights of the network are specified by a domain theory consisting of symbolic inference rules.

[2]TREPAN stands for TREes PArroting Networks. The name is also metaphorical, however: one connotation of the word trepan is to cut a hole in a skull in order to access the brain.

Table 1: The TREPAN decision-tree extraction algorithm.

```
/* Given a net's training set and feature set, induce a tree that models the net.  */
TREPAN($training\_examples$, $features$)
{
   for each example $E \in training\_examples$    /* obtain the net's labeling of each example */
      class label for $E$ := ORACLE($E$)
   return MAKE_SUBTREE($training\_examples$, $features$, {})
}

MAKE_SUBTREE($examples$, $features$, $constraints$)
{
   check stopping criteria using $examples$ and calls to ORACLE($constraints$)
   if stopping criteria satisfied
      make new leaf, $L$
      determine class label for $L$ using $examples$ and calls to ORACLE($constraints$)
      return $L$
   else
      /* use $features$ to build splits;
         use $examples$ and calls to ORACLE($constraints$) to evaluate them */
      $S$ := best binary split
      using $S$ as a seed, search for best $M$-of-$N$ split, $S'$
      make new node, $N$, for $S'$
      for each outcome, $I$, of $S'$
         $examples_{(I)}$ := members of $examples$ with outcome $I$ on split $S'$
         $constraints_{(I)}$ := $constraints \cup \{S' = I\}$
         $I$th child of $N$ := MAKE_SUBTREE($examples_{(I)}$, $features$, $constraints_{(I)}$)
      return $N$
}
```

splits that create each of the tree's internal nodes. These aspects of the algorithm are discussed in more detail below.

Split Types. The role of internal nodes in a decision tree is to partition the input space in order to increase the separation of members of different classes. In C4.5, each of these splits is based on a single feature. For example, a split on a binary feature results in two subtrees being created as children of the current node – one for each value of the feature. Our algorithm, like Murphy and Pazzani's ID2-of-3 algorithm (1991), forms trees that use M-of-N expressions for its splits.

An M-of-N expression is a Boolean expression that is specified by an integer threshold, m, and a set of n Boolean conditions. An M-of-N expression is satisfied when at least m of its n conditions are satisfied. For example, suppose we have three Boolean features, a, b, and c; the M-of-N expression *2-of-*$\{a,\ \neg b,\ c\}$ is logically equivalent to $(a\ \wedge\ \neg b)\ \vee\ (a\ \wedge\ c)\ \vee\ (\neg b\ \wedge c)$.

Split Selection. Split selection involves deciding how to partition the input space at a given internal node in the tree. A limitation of conventional tree-induction algorithms is that the amount of training data used to select each split decreases with the depth of the tree. Thus splits near the bottom of a tree are often poorly chosen because the decision is based on few training examples. In contrast, because we have an oracle available, TREPAN is able to generate as many training examples as desired to select each split in the tree. Currently, our algorithm chooses a split after considering at least S_{min} examples, where S_{min} is a parameter of the algorithm. Although it is possible to use a statistical test for this decision (Musick et al., 1993), it is not clear that the computation involved in estimating the relevant statistic is worth the effort, since each split-selection step is only a locally optimal decision.

When using the oracle to select a split at a given node, the oracle is given the list of all of the previously selected splits that lie on the path from the root of the tree to that node. These splits serve as constraints on the feature values that any instance generated by the oracle can take. Note that any example must satisfy these constraints in order to reach the current node.

Like the ID2-of-3 algorithm, TREPAN uses a heuristic search process to construct its *M*-of-*N* splits. The search process begins by first selecting the best binary split at the current node; TREPAN uses *information gain* (Quinlan, 1986) to evaluate candidate splits. For two-valued features, a binary split separates examples according to their values for the feature. For features with more than two values, we consider binary splits based on each allowable value of the feature (e.g., *color=red?, color=blue?, ...*). The selected binary split serves as seed for the search process. The search uses information gain as its heuristic evaluation function, and uses the two following operators (Murphy & Pazzani, 1991):

- *m–of–n+1* : Add a new value to the set, and hold the threshold constant. For example, *2-of-*$\{a,\ b\} \Longrightarrow$ *2-of-*$\{a,\ b,\ c\}$.
- *m+1–of–n+1*: Add a new value to the set, and increment the threshold. For example, *2-of-*$\{a,\ b,\ c\} \Longrightarrow$ *3-of-*$\{a,\ b,\ c,\ d\}$.

TREPAN's search for *M*-of-*N* splits differs from ID2-of-3 in two ways. First, whereas ID2-of-3 uses a hill-climbing search, TREPAN uses a beam search. Second, TREPAN constrains *M*-of-*N* splits so that the same feature is not used in two or more disjunctive splits which lie on the same path between the root and a leaf of the tree. Without this restriction, the oracle might have to solve difficult constraint-satisfaction problems in order create instances for nodes on such a path.

Stopping Criteria. TREPAN uses two separate criteria to decide when to stop growing an extracted decision tree. First, a given node becomes a leaf in the tree if, with high probability, the node covers only instances of a single class. In order to make this decision, TREPAN determines the proportion of examples, p_c, that fall into the most common class at a given node, and then calculates a confidence interval around this proportion. The oracle is queried for additional examples until $prob(p_c < 1 - \epsilon) < \delta$, where ϵ and δ are parameters of the algorithm.

TREPAN also accepts a parameter that enables the depth of extracted trees to be limited. In some domains, it may require a very large tree to describe a given network to a high level of fidelity. Since we are interested in extracting comprehensible trees, we use this parameter to limit TREPAN to extracting relatively small trees.

Leaf Labeling. Every leaf in a decision tree is given a label that corresponds to the class it predicts for those examples that reach it. In TREPAN, the class label for a leaf is determined by first calculating the proportion of examples, p_c, that fall into the most common class at the leaf. In cases where p_c has not satisfied the first stopping criterion, TREPAN must determine if its confidence in p_c is sufficient to label the leaf with the corresponding class. If necessary, the oracle is queried for additional examples until $\forall_{i \neq c}[\ prob(p_c < p_i) < \delta\ ]$, where δ is a parameter of the algorithm. The *Bonferroni correction* (Rice, 1995) is used for problems that involve more than two classes.

3.2 Empirical Evaluation of the TREPAN Algorithm

In our experiments, we are interested in evaluating our algorithm according to three criteria: (i) the fidelity of the trees to the networks from which they were extracted; (ii) the comprehensibility of the trees; (iii) the accuracy of our extracted decision trees, i.e., how well they generalize.

Table 2: Test-set accuracy.

domain	method			
	networks	C4.5	ID2-of-3	TREPAN
voting	91.5%	89.2%	86.9	92.2%
protein coding	93.7	88.7	91.1	91.7
promoters	90.6	84.4	83.8	86.5

We evaluate TREPAN using three real world domains: Congressional voting (Schlimmer & Fisher, 1986), recognition of protein-coding regions in DNA (Craven & Shavlik, 1993b), and recognition of promoters in DNA (Towell et al., 1990). The voting domain has 15 Boolean features and two output categories. Following Buntine and Niblett (1992), we remove the `physician-fee-freeze` feature to make the problem more difficult. The protein-coding domain has 64 Boolean features and two output classes. The promoter problem has 57 features with four values each, and two output classes. The voting data set has 435 examples, the promoter set has 438 examples, and the protein-coding domain has 20,000 examples. Note that the promoter set we use is larger and more complex than the original data set (Towell et al., 1990). We randomly partition the voting and the promoter data into 10 training and test sets. Because of certain domain-specific characteristics of the data, we use only four training and test sets for the protein-coding task.

We measure accuracy and fidelity on the examples in the test sets. Whereas accuracy is defined as the percentage of test-set examples that are correctly classified, *fidelity* is defined as the percentage of test-set examples on which the classification made by a tree agrees with its neural-network counterpart. Since the comprehensibility of a decision tree is problematic to measure, we measure the syntactic complexity of our extracted trees and take this as being representative of comprehensibility. Specifically, we measure the complexity of a given tree in two ways: (i) the number of internal (i.e., non-leaf) nodes in the tree, and (ii) the number of *symbols* used in the splits of the tree. We count an ordinary, single-feature split as one symbol. We count an M-of-N split as n symbols, since such a split lists n feature values.

The neural networks we use in our experiments have a single layer of hidden units. The number of hidden units used for each network is chosen using cross validation on the network's training set. For each training set, we perform cross-validation runs using 0, 5, 10, 20 and 40 hidden units. The networks are trained using a conjugate-gradient learning method (Kramer & Sangiovanni-Vincentelli, 1989). Training continues until either (i) all of the training-set examples are correctly classified, (ii) a local minimum in the error surface is reached, or (iii) 100 search directions have been tried. A validation set is used to decide at which epoch the weights are saved for a given training run.

Our TREPAN algorithm is applied to each saved network. TREPAN has several parameters which are set as follows for all runs: at least 1000 examples are considered before selecting each split; the beam-width for the M-of-N split search is set to five; the maximum tree depth is also set to five; we set the ϵ and δ parameters, which are used for the stopping-criterion and the leaf-labeling procedures, to 0.05.

As baselines for comparison, we also run Quinlan's C4.5 algorithm (Quinlan, 1993), and Murphy and Pazzani's ID2-of-3 algorithm (1991) on the same testbeds. Recall that ID2-of-3 is similar to C4.5, except that it learns trees that use M-of-N splits. C4.5 avoids overfitting by pruning decision trees after they have been grown. We use cross validation to select the level of pruning used for each training set. The cross-validation runs evaluate unpruned trees and trees pruned with confidence levels ranging from 10% to 90%.

Table 3: Fidelity to networks (measured on test sets).

domain	method		
	C4.5	ID2-of-3	TREPAN
voting	95.2%	92.0%	96.3%
protein coding	89.7	92.5	93.2
promoters	81.8	85.0	88.3

Table 4: Tree complexity.

domain	# internal nodes			# symbols		
	C4.5	ID2-of-3	TREPAN	C4.5	ID2-of-3	TREPAN
voting	20.1	16.3	11.8	20.1	56.3	23.7
protein coding	146.5	5.5	5.0	146.5	58.3	43.3
promoters	11.2	8.6	18.0	11.2	36.9	67.5

Table 2 shows the test-set accuracy results for all four algorithms on each data set. It can be seen that, for every data set, neural networks generalize better than decision trees learned by the C4.5 algorithm. Similarly, although ID2-of-3 generalizes better than C4.5 on most of the problem domains, it does not match the performance of the neural networks on any domain. The decision trees extracted from the networks by our TREPAN algorithm, on the other hand, are more accurate than the C4.5 trees and the ID2-of-3 trees for all domains. Moreover, on the voting task, the trees produced by TREPAN are actually more accurate than the networks from which they were extracted (although the difference is not statistically significant). For all three domains, the differences in accuracy between the neural networks and the two conventional decision-tree algorithms (C4.5 and ID2-of-3) are statistically significant at the 0.05 level using a paired, two-tailed t-test. Clearly, these are domains in which the inductive bias of neural networks is more suitable than the inductive bias of ordinary decision-tree algorithms. We also test the significance of the accuracy differences between TREPAN and the other decision-tree algorithms. For all three domains, these differences are statistically significant. The results in this table indicate that our algorithm is able to extract decision trees which are more accurate than decision trees induced strictly from the training data.

Table 3 shows the fidelity measurements on each data set. Recall that fidelity is defined as the percentage of test-set examples for which a given tree and its neural-network counterpart make the same classification. As expected, the TREPAN trees exhibit a higher level of network fidelity than do either the C4.5 or the ID2-of-3 trees. We measure the statistical significance of fidelity differences using a paired, two-tailed t-test at a significance level of 0.05. Except for the difference between TREPAN and C4.5 on the voting task, all differences between TREPAN and the other two algorithms are statistically significant.

Table 4 shows tree-complexity measurements for C4.5, ID2-of-3, and TREPAN on all three data sets. Except for on the promoter domain, the trees learned by TREPAN have fewer internal nodes than the trees produced by C4.5 and ID2-of-3. Although the trees generated by TREPAN on the promoter problem are more complex than C4.5 and ID2-of-3 trees, we believe that this difference is not so large as to make the TREPAN trees significantly more difficult to understand. Based on these results, we conjecture that the trees extracted by TREPAN are as comprehensible as the trees learned by conventional decision-tree algorithms.

4 Comprehensibility Issues in Extracting Symbolic Representations from Trained Networks

In this section, we address some of the issues related to comprehensibility that have arisen in our research. We first discuss a number of directions that we have taken in our approach, and the reasoning behind the decisions we have made. We then present a number of open issues that we plan to consider in future work.

Some of the major issues pertaining to comprehensibility that we have considered in our work to date are the following:

- **Concept descriptions and explanations:** Much research on the topic of comprehensibility in AI has focused on generating *explanations* of a system's behavior (Shortliffe, 1976; Swartout, 1983; Paris, 1987; Moore & Swartout, 1989). The explanation task in this body of work has involved both describing why a given instance is classified the way that it is, and describing the defining criteria for a particular concept. In contrast to providing explanations of individual instances, our research has concentrated on providing complete (or nearly complete) descriptions of learned concept representations; an extracted decision tree describes *all* of the sets of sufficient conditions for each class. Our TREPAN extraction algorithm, however, can also be used for the task of describing the learned concept over limited regions of a classifier's instance space. As one anonymous referee pointed out, this limited extraction task may be interesting because the instance space may be so large and heterogeneous that it is not possible to produce a comprehensible description of a network's entire concept representation. The ability to extract descriptions of limited parts of learned concepts is also useful when the users of a classifier have specific questions about the learned concept that they want answered.

- **Symbolic descriptions:** To date, our research has focused on extracting concept descriptions like those produced by common symbolic learning algorithms (Michalski, 1983; Quinlan, 1993). We have used these kind of representation languages, instead of more numerically oriented representations – such as probabilistic rules or linear discriminant functions – because we believe that symbolic representations are usually more comprehensible. As discussed below, however, we are reconsidering this issue. Incidentally, our research to date has involved problem domains that have only discrete-valued attributes and discrete output categories. We are currently extending our TREPAN algorithm to handle problems with real-valued attributes.

- ***M*-of-*N* expressions:** A central aspect of our extracted representations is the use of *M*-of-*N* expressions. We contend that *M*-of-*N* expressions can greatly aid comprehensibility by reducing the syntactic complexity of extracted concept representations and by highlighting symmetric relationships that are present in them. We have found empirically that, by using *M*-of-*N* expressions in our extracted representations, we are able to produce concept descriptions which are much more concise than extracted representations that do not allow them (Craven & Shavlik, 1993a; Towell & Shavlik, 1993). As evidence that *M*-of-*N* rules are readily understandable, we note that in the field of medicine, they are common representations for diagnostic decision criteria (Spackman, 1988).

- **Syntactic complexity:** A major premise of our work is that syntactic complexity is a good indicator of comprehensibility. For a given representation language, we contend that, other things being equal, simpler descriptions are better than complex descriptions. Not surprisingly, the psychological literature supports the notion that humans prefer simple concepts (Neisser & Weene, 1962; Pinker, 1979; Medin et al., 1987). We have used measures such as rule and antecedent counts to quantify the complexity of rule sets, and node and "symbol" counts to quantify the complexity of decision trees. Complexity is a thorny issue, however, since it does not have an objective arbiter. Even comparing representations expressed in the same language is problematic because it requires specifying the relative costs of different components of the language. For example, is a description with lots of short rules (few antecedents) more or less complex than a description with few long rules?

There are still many issues that we plan to further explore in our research. Among them are the following:

- **Trading off comprehensibility and fidelity:** Our algorithm may produce concept descriptions that have a high level of fidelity to the networks from which they were extracted, yet are too complex to be understood. In such situations, we would like to have a mechanism that enables fidelity to be traded-off for comprehensibility. Hence, an area for future investigation is to design an algorithm that is able to transform extracted concept descriptions into simpler, though perhaps less faithful and accurate, representations.

- **Numeric representations:** As mentioned previously, our research has focused on producing concept descriptions like those formed by common symbolic learning algorithms such as AQ (Michalski, 1983) and C4.5 (Quinlan, 1993). However, we also plan to investigate the utility of concept representations that use numeric weights to represent the strength of relationships. Specifically, we plan to consider the representation of simple perceptrons that have attributes and combinations of attributes as inputs. Although it may seem pointless to translate a neural network into another, albeit simpler network, we do not believe that this is true. Recall that in Section 2, we stated that neural networks are difficult to understand because they usually have a large number of parameters, and because the relationships between input features and output categories may be nonlinear and nonmonotonic. We contend that representations, such as simple perceptrons, in which there are relatively few real-valued parameters and in which each parameter describes a simple (i.e. linear) relationship, are comprehensible. As evidence for this position, consider that linear discriminant functions are commonly used to express decision criteria in medicine (Spackman, 1988). They are also widely used in molecular biology to describe sequence characteristics of interest (Stormo, 1987).

  In order to produce a perceptron that is able to accurately describe a network that has learned high-order relationships, it is necessary to have some of the perceptron's inputs represent combinations of attributes (conjunctions of attributes, for example). We are currently investigating the applicability of algorithms developed in the computational learning theory community to this task (Blum et al., 1994). The potential benefit of using simple perceptrons to describe trained multi-layer networks is that they may, in some cases, be more comprehensible than their equivalent decision trees. We view this as a question to be decided by domain experts (see below).

- **Expert evaluation of extracted representations:** To date, we have evaluated the comprehensibility of our extracted representations by measuring their syntactic complexity. It is not clear how well syntactic complexity serves as a proxy for actual comprehensibility, and thus we would like to have humans evaluate our extracted representations in some domains. We do not have the resources to conduct the large-scale psychological experiment that would be needed to objectively measure the comprehensibility of our representations. However, we do plan to work with experts in some of our problem domains (e.g., branch prediction in compilers, Calder et al., 1995) in order to assess how comprehensible they find the representations produced by our algorithms.

- **Dialogue vs. one-shot processes:** Some work has cast the explanation problem not as a "one-shot" process, but instead as a dialogue between system and user. An open research issue in our work is how we might enable the user to request more, or different, information in response to a concept description produced by our method. For example, we may augment our system so that it can produce more abstract descriptions when requested to do so by the user.

5 Related Research

A number of researchers have employed *scientific visualization* techniques to assist in understanding neural networks. Scientific visualization uses graphical attributes such as color, size, and spatial organization to depict systems with large numbers of numerical parameters. Visualization techniques have been used to illustrate both the learning process and the prediction process in neural networks. Specifically, the following aspects of neural networks have been described using visualization methods: the weights and magnitudes of a network's connections (Hinton, 1986; Wejchert & Tesauro, 1989; Craven & Shavlik, 1992); the decision boundaries formed by units in a network (Lang & Witbrock, 1988; Munro, 1991; Pratt et al., 1991); unit activations and the forward propagation of activation signals through the network (Craven & Shavlik, 1992); the backward propagation of error signals during learning (Craven & Shavlik, 1992); and the trajectory of units in weight space during learning (Wejchert & Tesauro, 1989). Although visualization methods can provide insight into the learning and prediction behavior of a network, they are not a substitute for extraction methods. None of these visualization methods is able to provide a complete description of a concept learned by a neural network, and several of the methods (e.g., decision-boundary visualization) only work well for very small networks.

A number of statistical techniques have been used to characterize the activity of the hidden units in neural networks. These methods include: hierarchical clustering of hidden-unit activations (Sejnowski & Rosenberg, 1987; Elman, 1989; Hanson & Burr, 1990), *contribution analysis* (Sanger, 1989), and *weight pattern analysis* (Gorman & Sejnowski, 1988). Like visualization techniques, these hidden-unit analysis methods are limited in that they do not provide complete descriptions of the concepts learned by networks.

Numerous research groups have developed methods for extracting finite-state automata (FSA) from recurrent neural networks (Cleeremans et al., 1989; Pollack, 1991; Giles et al., 1992; Watrous & Kuhn, 1992; Zeng et al., 1993; Das & Mozer, 1994). Because recurrent networks have links from their hidden or output units to their input units, they are able to maintain state information from one input instance to the next. This ability to maintain state information allows recurrent networks to learn simple grammatical concepts. Recurrent networks have been trained both to predict the next element in a sentence, and to distinguish positive and negative instances of a target language. FSA-extraction algorithms operate by associating patterns of hidden-unit activity with states in a FSA, and then recording the state transitions that occur for various inputs. Although these algorithms have been able to extract accurate finite-state automata from networks trained on simple languages, they have not yet shown their usefulness for real-world applications.

Other research groups have also investigated the problem of extracting symbolic representations from neural networks. Many of these approaches require special network architectures, training methods, or both (Hayashi, 1990; McMillan et al., 1992; Sethi et al., 1993; Tan, 1994; Tchoumatchenko & Ganascia, 1994; Alexander & Mozer, 1995). Our algorithm, in contrast, simply uses the network as a black box to answer queries during the extraction process. There are several existing algorithms which do not require special network architectures or training procedures (Saito & Nakano, 1988; Fu, 1991; Gallant, 1993). These algorithms, however, do not extract M-of-N rules, but instead extract only conjunctive rules. In previous work (Towell & Shavlik, 1993; Craven & Shavlik, 1994), we have shown that this type of algorithm produces rule-sets which typically are far to complex to be comprehensible. Additionally, these algorithms assume that each hidden unit in a network can be accurately approximated by a threshold unit. Thrun (1995) has developed a general method for rule extraction, and has described how his algorithm can be used to *verify* that an M-of-N rule is consistent with a network, but he has not proposed a method for searching for M-of-N rule sets. A strength of our algorithm, in contrast, is its scalability. We have demonstrated that our algorithm is able to produce succinct decision-tree descriptions of networks with large input spaces.

6 Conclusions

A widely held belief in the machine-learning community is that neural networks are not suitable for problems in which it is important to be able to comprehend induced concepts. We have been working to develop methods that elicit comprehensible concept descriptions from trained neural networks. We have discussed a number of issues related to comprehensibility that have arisen in our research. We have also presented a number of open issues that we plan to consider in future work. It is our hope that this paper will serve to provoke discussion on two fronts: the relationship between neural networks and comprehensibility in machine learning, and general issues of comprehensibility in machine learning that transcend any particular learning method (e.g, how can we measure comprehensibility).

References

Alexander, J. A. & Mozer, M. C. (1995). Template-based algorithms for connectionist rule extraction. In Tesauro, G., Touretzky, D., & Leen, T., editors, *Advances in Neural Information Processing Systems (volume 7)*. MIT Press, Cambridge, MA.

Atlas, L., Cole, R., Connor, J., El-Sharkawi, M., Marks II, R. J., Muthusamy, Y., & Barnard, E. (1989). Performance comparisons between backpropagation networks and classification trees on three real-world applications. In Touretzky, D., editor, *Advances in Neural Information Processing Systems (volume 2)*, (pp. 622–629), San Mateo, CA. Morgan Kaufmann.

Blum, A., Furst, M., Jackson, J., Kearns, M., Mansour, Y., & Rudich, S. (1994). Weakly learning DNF and characterizing statistical query learning using fourier analysis. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, (pp. 253–262), Montreal, Canada. ACM Press.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA.

Buntine, W. & Niblett, T. (1992). A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8:75–86.

Calder, B., Grunwald, D., Lindsay, D., Martin, J., Mozer, M., & Zorn, B. (1995). Corpus-based static branch prediction. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, (pp. 18–21), La Jolla, CA.

Cleeremans, A., Servan-Schreiber, D., & McClelland, J. (1989). Finite state automata and simple recurrent networks. *Neural Computation*, 1:372–381.

Craven, M. W. & Shavlik, J. W. (1992). Visualizing learning and computation in artificial neural networks. *International Journal on Artificial Intelligence Tools*, 1(2):399–425.

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

Craven, M. W. & Shavlik, J. W. (1993b). Learning to predict reading frames in E. coli DNA sequences. In *Proceedings of the 26th Hawaii International Conference on System Sciences*, (pp. 773–782), Wailea, HI. IEEE Press.

Craven, M. W. & Shavlik, J. W. (1993c). Learning to represent codons: A challenge problem for constructive induction. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, (pp. 1319–1324), Chambery, France. Morgan Kaufmann.

Craven, M. W. & Shavlik, J. W. (1994). Using sampling and queries to extract rules from trained neural networks. In *Proceedings of the Eleventh International Conference on Machine Learning*, (pp. 37–45), New Brunswick, NJ. Morgan Kaufmann.

Craven, M. W. & Shavlik, J. W. (1995). Extracting tree-structured representations of trained networks. Submitted for publication.

Das, S. & Mozer, M. (1994). A unified gradient-descent/clustering architecture for finite state machine induction. In Cowan, J., Tesauro, G., & Alspector, J., editors, *Advances in Neural Information Processing Systems (volume 6)*. Morgan Kaufmann, San Mateo, CA.

Elman, J. L. (1989). Representation and structure in connectionist models. Technical Report 8903, Center for Research in Language, University of California, San Diego.

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

Flann, N. S. & Dietterich, T. G. (1986). Selecting appropriate representations for learning from examples. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, (pp. 460–466), Philadelphia, PA. Morgan Kaufmann.

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

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

Giles, C. L., Miller, C. B., Chen, D., Chen, H. H., Sun, G. Z., & Lee, Y. C. (1992). Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4:393–405.

Gorman, R. P. & Sejnowski, T. J. (1988). Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1:75–89.

Hanson, S. J. & Burr, D. J. (1990). What connectionist models learn: Learning and representation in connectionist networks. *Behavioral and Brain Sciences*, 13:471–518.

Hayashi, Y. (1990). A neural expert system with automated extraction of fuzzy if-then rules. In *Advances in Neural Information Processing Systems (volume 3)*, (pp. 578–584), San Mateo, CA. Morgan Kaufmann.

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.

Jordan, M. (1986). Serial order: A parallel distributed processing approach. Technical Report 8604, University of California, Institute for Cognitive Science, San Diego.

Kramer, A. H. & Sangiovanni-Vincentelli, A. (1989). Efficient parallel learning algorithms for neural networks. In Touretzky, D., editor, *Advances in Neural Information Processing Systems (volume 1)*. Morgan Kaufmann, San Mateo, CA.

Lang, K. J. & Witbrock, M. J. (1988). Learning to tell two spirals apart. In *Proceedings of the 1988 Connectionist Models Summer School*, (pp. 52–59). Morgan Kaufmann.

McMillan, C., Mozer, M. C., & Smolensky, P. (1992). Rule induction through integrated symbolic and subsymbolic processing. In Moody, J., Hanson, S., & Lippmann, R., editors, *Advances in Neural Information Processing Systems (volume 4)*. Morgan Kaufmann, San Mateo, CA.

Medin, D. L., Wattenmaker, W. D., & Michalski, R. S. (1987). Constraints and preferences in inductive learning: An experimental study of human and machine performance. *Cognitive Science*, 11:299–339.

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

Mitchell, T., Caruana, R., Freitag, D., McDermott, J., & Zabowski, D. (1994). Experience with a learning personal assistant. *Communications of the ACM*, 37(7):80–91.

Moore, J. D. & Swartout, W. R. (1989). A reactive approach to explanation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, (pp. 1504–1510), Detroit, MI.

Munro, P. (1991). Visualizations of 2-D hidden unit space. Technical Report LIS035/IS91003, School of Library and Information Science, University of Pittsburgh, Pittsburgh, PA.

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

Musick, R., Catlett, J., & Russell, S. (1993). Decision theoretic subsampling for induction on large databases. In *Proceedings of the Tenth International Conference on Machine Learning*, (pp. 212–219), Amherst, MA. Morgan Kaufmann.

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

Ourston, D. & Mooney, R. (1994). Theory refinement combining analytical and empirical methods. *Artificial Intelligence*, 66(2):273–309.

Paris, C. (1987). Combining discourse strategies to generate descriptions to users along a naive/expert spectrum. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, (pp. 626–632), Milan, Italy.

Pazzani, M. & Kibler, D. (1992). The utility of knowledge in inductive learning. *Machine Learning*, 9(1):57–94.

Pineda, F. J. (1987). Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59:2229–2232.

Pinker, S. (1979). Formal models of language learning. *Cognition*, 7:217–283.

Pollack, J. (1991). The induction of dynamical recognizers. *Machine Learning*, 7:227–252.

Pomerleau, D. A. (1991). Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3:88–97.

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

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.

Rice, J. A. (1995). *Mathematical Statistics and Data Analysis*. Wadsworth, Belmont, CA.

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.

Sanger, D. (1989). Contribution analysis: A technique for assigning responsibilities to hidden units in connectionist networks. *Connection Science*, 1(2):115–138.

Schlimmer, J. C. & Fisher, D. (1986). A case study of incremental concept induction. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, (pp. 496–501), Philadelphia, PA.

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

Sethi, I. K., Yoo, J. H., & Brickman, C. M. (1993). Extraction of diagnostic rules using neural networks. In *Proceedings of the Sixth IEEE Symposium on Computer-Based Medical Systems*, (pp. 217–222), Ann Arbor, MI. IEEE Press.

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

Shortliffe, E. (1976). *Computer-based medical consultations: MYCIN*. American Elsevier, New York, NY.

Smyth, P. & Mellstrom, J. (1992). Fault diagnosis of antenna pointing systems using hybrid neural network and signal processing models. In Moody, J., Hanson, S., & Lippmann, R., editors, *Advances in Neural Information Processing Systems (volume 4)*, (pp. 667–674), San Mateo, CA. Morgan Kaufmann.

Spackman, K. A. (1988). Learning categorical decision criteria. In *Proceedings of the Fifth International Conference on Machine Learning*, (pp. 36–46), Ann Arbor, MI.

Stormo, G. (1987). Identifying coding sequences. In Bishop, M. J. & Rawlings, C. J., editors, *Nucleic Acid and Protein Sequence Analysis: A Practical Approach.* IRL Press, Oxford, England.

Swartout, W. (1983). XPLAIN: A system for creating and explaining expert consulting programs. *Artificial Intelligence*, 21(3):285–325.

Tan, A.-H. (1994). Rule learning and extraction with self-organizing neural networks. In *Proceedings of the 1993 Connectionist Models Summer School.* Lawrence Erlbaum Associates, Hillsdale, NJ.

Tchoumatchenko, I. & Ganascia, J.-G. (1994). A Bayesian framework to integrate symbolic and neural learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, (pp. 302–308), New Brunswick, NJ. Morgan Kaufmann.

Tesauro, G. & Sejnowski, T. J. (1989). A parallel network that learns to play backgammon. *Artificial Intelligence*, 39(3):357–390.

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

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

Towell, G. & Shavlik, J. (1994). Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1-2):119–165.

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.

Watkins, C. (1989). *Learning from delayed rewards.* PhD thesis, King's College, Cambridge.

Watrous, R. L. & Kuhn, G. M. (1992). Induction of finite state languages using second-order neural networks. In Moody, J., Hanson, S., & Lippmann, R., editors, *Advances in Neural Information Processing Systems (volume 4).* Morgan Kaufmann, San Mateo, CA.

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

Wejchert, J. & Tesauro, G. (1989). Neural network visualization. In Touretzky, D., editor, *Advances in Neural Information Processing Systems (volume 2)*, (pp. 465–472), San Mateo, CA. Morgan Kaufmann.

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

Zeng, Z., Goodman, R. M., & Smyth, P. (1993). Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, 5(6):976–990.