University of Wisconsin Computer Sciences Technical Report 955 (August 1990)

Symbolic and Neural Learning Algorithms: An Experimental Comparison (Revised)

Jude W. Shavlik

Computer Sciences Department
University of Wisconsin
1210 West Dayton Street
Madison, WI 53706

shavlik@cs.wisc.edu

Raymond J. Mooney

Department of Computer Sciences
Taylor Hall 2.124
University of Texas
Austin, TX 78712

mooney@cs.utexas.edu

Geoffrey G. Towell

Computer Sciences Department
University of Wisconsin
Madison, WI 53706

towell@cs.wisc.edu

Keywords: empirical learning, connectionism, neural networks, inductive learning, ID3, perceptron, backpropagation

(This paper will appear in *Machine Learning*, Volume 6, 1991.)

Symbolic and Neural Learning Algorithms: An Experimental Comparison (Revised)

Abstract

Despite the fact that many symbolic and neural network (connectionist) learning algorithms address the same problem of learning from classified examples, very little is known regarding their comparative strengths and weaknesses. Experiments comparing the ID3 symbolic learning algorithm with the perceptron and backpropagation neural learning algorithms have been performed using five large, real-world data sets. Overall, backpropagation performs slightly better than the other two algorithms in terms of classification accuracy on new examples, but takes much longer to train. Experimental results suggest that backpropagation can work significantly better on data sets containing numerical data. Also analyzed empirically are the effects of (1) the amount of training data, (2) imperfect training examples, and (3) the encoding of the desired outputs. Backpropagation occasionally outperforms the other two systems when given relatively small amounts of training data. It is slightly more accurate than ID3 when examples are noisy or incompletely specified. Finally, backpropagation more effectively utilizes a "distributed" output encoding.

1. INTRODUCTION

The division between symbolic and neural network approaches to artificial intelligence is particularly evident within machine learning. Both symbolic and artificial neural network (or connectionist) learning algorithms have been developed; however, until recently (Fisher & McKusick, 1989; Mooney, Shavlik, Towell, & Gove, 1989; Weiss & Kapouleas, 1989; Atlas et al., 1990; Dietterich, Hild, & Bakiri, 1990) there has been little direct comparison of these two basic approaches to machine learning. Consequently, despite the fact that symbolic and connectionist learning systems frequently address the same general problem, very little is known regarding their comparative strengths and weaknesses.

The problem most often addressed by both neural network and symbolic learning systems is the inductive acquisition of concepts from examples. This problem can be briefly defined as follows: given descriptions of a set of examples each labeled as belonging to a particular class, determine a procedure for correctly assigning new examples to these classes. In the neural network literature, this problem is frequently referred to as *supervised* or *associative* learning.

For associative learning, symbolic and neural systems generally require the same input; namely, classified examples represented as feature vectors. In addition, the performance of both types of learning systems is usually evaluated by testing their ability to correctly classify novel examples. Within symbolic machine learning, numerous algorithms have been developed to perform associative learning. Systems exist for learning decision trees (e.g., Quinlan, 1986a) and logical concept definitions (Mitchell, 1982; Michalski, 1983) from examples, which can be used to classify subsequent examples. These algorithms have been tested on problems ranging from soybean disease diagnosis (Michalski & Chilausky, 1980) to classifying chess end games (Quinlan, 1983). Within neural networks, several algorithms have been developed for training a network to respond correctly to a set of examples by appropriately modifying its connection weights (e.g., Rosenblatt, 1962; Rumelhart, Hinton, & Williams, 1986; Hinton & Sejnowski, 1986). After training, a network can be used to classify novel examples. Neural learning algorithms have been tested on problems ranging from converting text to speech (Sejnowski & Rosenberg, 1987; Dietterich et al., 1990) to evaluating moves in backgammon (Tesauro & Sejnowski, 1989).

This article presents the results of several experiments comparing the performance of the *ID3* symbolic learning algorithm (Quinlan, 1986a) with both the *perceptron* (Rosenblatt, 1962) and *backpropagation* (Rumelhart et al., 1986) neural algorithms. All three systems are tested on five large data sets from previous symbolic and connectionist experiments, and their learning times and accuracies on novel examples are measured. In two cases, backpropagation's classification accuracy was statistically significantly better than ID3's. In the other three, they perform roughly equivalently. However, in all cases, backpropagation took much longer to train. Given its well-known limitations, the perceptron performed surprisingly well; it usually performed within a few percentage points of the other methods.

Besides accuracy and learning time, this paper investigates three additional aspects of empirical learning: (1) the dependence on the amount of training data; (2) the ability to handle imperfect data of various types; (3) the ability to utilize "distributed" output encodings. These investigations shed further light on the relative merits of the different approaches to inductive learning.

Three types of imperfect data sets are studied. The first type (random noise) occurs when feature values and example categorizations are incorrectly recorded, the second results from missing feature values, while the third occurs when an insufficient number of features are used to describe examples.

The output encoding issue arises because symbolic learning algorithms usually are trained to learn category names. A direct translation of this approach yields a "local" connectionist representation of each category as the output to be learned. That is, each possible category would be represented by a single output unit. However, the backpropagation algorithm is often trained with more complicated output encodings that use information upon which the category naming is based. These "distributed" encodings of the output are more compact than the local encodings and may have aspects that neurally-based algorithms can exploit during learning.

Results of the experiments described in the following sections indicate that the neural methods occasionally perform better than ID3 when given relatively small amounts of training data. The experiments also indicate that backpropagation is slightly more accurate than ID3 when data sets contain imperfections. Finally, the results suggest that backpropagation takes greater advantage of domain-specific distributed output encodings, although ID3 is able to successfully use distributed encodings.

The next section describes the five data sets and their representation, plus the three algorithms and their implementations. Following that, the experiments are described and their results reported and analyzed. A fourth section discusses general issues concerning the relationships between symbolic and neural approaches to inductive learning.

2. GENERAL EXPERIMENTAL SETUP

This section first describes the data sets used to compare the different learning systems. Next, it motivates the choice of algorithms used in the study and briefly describes these algorithms. Finally, it describes the representation language used to encode examples.

2.1. Data Sets

These experiments use five different data sets. Four have been previously used to test different symbolic learning systems and one has been used to test backpropagation.

The *soybean* data set has 17 different soybean diseases described by 50 features, such as weather, time of year, and descriptions of leaves and stems. Each disease has 17 examples, for a total of 289 examples. This domain was popularized by Michalski & Chilausky (1980); however, the exact data is that used in Reinke (1984). It should be noted that several soybean data sets exist. The full set used here should not be confused with the simpler, purely conjunctive, four-disease data used to test clustering systems (Stepp, 1984; Fisher, 1987).

The *chess* data set (Shapiro, 1987) consists of examples of a “king and rook versus king and pawn” end game (KPa7KR). There are two categories (*Win* and *Not Win*) and 36 high-level features. There are a total of 591 examples randomly selected from the original 3196 examples; these examples are approximately evenly divided between the two categories. Similar chess end game data sets have been previously used to test ID3 (Quinlan, 1983).

The *audiology* data (Bareiss, 1989) consist of cases from the Baylor College of Medicine. There are 226 examples of 24 categories of hearing disorders involving 58 features. This data set has a large amount of missing information; an average of only about 11 features have known values for each example (Bareiss, 1989).

The *heart disease* data set contains medical cases from the Cleveland Clinic Foundation (Detrano, unpublished manuscript). It is the only one of the five data sets that contains numerically-valued features; there are eight nominally-valued (i.e., possible values form a finite, unordered set) and six numerically-valued features. There are two categories: healthy and diseased heart. The full data set contains 303 examples, roughly evenly divided between the two categories. However, seven examples were missing feature values, and we did not use these seven in our experiments.

The *NETtalk* (Sejnowski & Rosenberg, 1987) data set involves text-to-speech conversion. It consists of a 20,012 word dictionary in which every letter of each word is associated with a phoneme/stress pair. Training examples in NETtalk are formed by passing one word at a time through a seven-letter window. The phoneme/stress pair of the middle letter in the window constitutes the category of the example. Thus, the number of training examples for each word is equal to the number of letters in that word; the 20,012 word dictionary has more than 143,000 possible training examples.

A seven-letter window is insufficient to uniquely identify the phoneme/stress pair attributable to the central letter of that window. For instance, the sound on the first letter of *asset* and *assess* is different. However, the two words have identical seven-letter windows for the first letter: ---*asse*. As a result, the data can be considered to have low levels of noise.

Unfortunately, the full dictionary is too extensive to analyze tractably in a manner equivalent to the other domains. Instead, a small training set is extracted by looking at the 1000 most common English words (as reported in Kuchera, 1967) and keeping those that appear in the NETtalk dictionary. This training set — called *NETtalk-full* or, simply, *NETtalk* — contains 808 words, which produce 4,259 examples classified into 115 phoneme/stress categories. (In the whole NETtalk corpus, there are 166 categories. The 41 categories not represented in the training set constitute less than 0.5% of the examples in the corpus.) The standard test set for NETtalk-full consists of 1,000 words (7,241 examples) randomly selected from the remainder of the dictionary. For one experiment, the NETtalk-full data set is further pruned by keeping only the examples involving “A” sounds. This produces 444 examples that fall into 18 sound/stress categories; this data set is called *NETtalk-A*.

2.2. Learning Algorithms

In order to experimentally compare neural-net and symbolic learning algorithms, we chose the ID3, perceptron, and backpropagation algorithms as representative algorithms. This section briefly describes the systems used and our reasons for choosing them as representatives.

2.2.1. ID3

We chose ID3 because it is a simple and widely used symbolic algorithm for learning from examples. It has been extensively tested on a number of large data sets (Quinlan, 1983; Wirth & Catlett, 1988) and is the basis of several commercial rule-induction systems. In addition, ID3 has been augmented with techniques for handling numerically-valued features, noisy data, and missing information (Quinlan, 1986a). Finally, in experimental comparisons with other symbolic learning algorithms, ID3 generally performs about as well or better than other systems (O'Rorke, 1982; Rendell, Cho, & Seshu, 1989).

ID3 uses the training data to construct a *decision tree* for determining the category of an example. At each step, a new node is added to the decision tree by partitioning the training examples based on their value along a single, most-informative attribute. The attribute chosen is the one which minimizes the following function:

$$E(A) = -\sum_{i=1}^{V} \frac{S_i}{S} \sum_{j=1}^{N} \frac{k_{ji}}{S_i} \log_2 \frac{k_{ji}}{S_i}$$

where V is the number of values for attribute A, k_{ji} is the number of examples in the jth category with the ith value for attribute A, S is the total number of examples, S_i is the number of examples with the ith value for attribute A, and N is the number of categories. Each resulting partition is processed recursively, unless it contains examples of only a single category, in which case a leaf is created and labeled with this category. The information-gain criterion that determines the "splitting" attribute acts as a hill-climbing heuristic, which tends to minimize the size of the resulting decision tree.

Following Quinlan (1986b), we used *chi-squared pruning* to handle noisy data and prevent overfitting the data. This method terminates the growth of a branch of the decision tree when no remaining attribute improves prediction in a statistically significant manner. In this case, a leaf is created and labeled with the most common category in the partition. Missing feature values are handled using techniques recommended by Quinlan (1989); see Section 3.3.2 for more details. Numerically-valued features are handled by examining all thresholds (e.g. *height* < *5*) that split the data into two partitions (with N examples there are up to N-1 distinct splits for each numerical feature). Each such split is then treated the same in the information-gain calculation as a split on a nominal feature (Quinlan, 1986a).

2.2.2. Perceptron

As is well known, the perceptron learning procedure is incapable of learning concepts that are not linearly separable (Minsky & Papert, 1988). Despite this fact, it performs quite well on several large data sets used to test recent learning systems. Therefore, we included the perceptron, one of the first neural network learning algorithms, in this study as an example of a simple, fast neural learning algorithm.

The perceptron learning procedure is a method for adjusting the weights of a single linear threshold unit so that it produces the correct outputs for a set of training examples. The procedure performs gradient descent (i.e., hill climbing) in weight space in an attempt to minimize the sum of the squares of the errors across all of the training examples. The following specifies how the weights should be changed after each presentation of an input/output pair *p*:

$$\Delta_p w_i = a_{pi} \, (t_p - o_p)$$

where w_i is the weight for input feature *i*, t_p is the target output for pattern *p*, o_p is the current output for pattern *p*, and a_{pi} is the value of input feature *i* for pattern *p*. If the training data are linearly separable, this procedure is guaranteed to converge on a correct set of weights after a finite number of input presentations (Minsky & Papert, 1988).

The implementation of the perceptron we used in our experiments includes cycle detection so that, once a set of weights repeats, the system stops and indicates that the data is not linearly separable. The *perceptron cycling theorem* (Minsky & Papert, 1988) guarantees this will eventually happen if and only if the data are not linearly separable. (However, due to simulation time restrictions, the perceptron is also stopped if it cycles through the training data 100 times on the NETtalk-full data set and 5000 times on the other data sets.) A single perceptron is trained for each possible category to distinguish members of that category from all other categories. A test example is classified by passing it through all the perceptrons and assigning it to the category whose perceptron's output exceeds its threshold by the largest amount.

2.2.3. Backpropagation

Over the past several years, a few neural learning algorithms have been developed that are capable of learning concepts that are not linearly separable (e.g., Rumelhart et al., 1986; Hinton & Sejnowski, 1986; Barnard & Cole, 1989). Backpropagation (also called the *generalized delta rule*, Rumelhart et al., 1986) is a well-known procedure and has been tested on several large-scale problems (e.g., Sejnowski & Rosenberg, 1987; Tesauro & Sejnowski, 1989). Consequently, we chose backpropagation to represent the new neural learning algorithms.

Like the perceptron, backpropagation uses gradient descent in an attempt to minimize the sum of the squares of the errors across all of the training inputs. However, it is capable of doing this for multi-layer networks with hidden units (units that neither directly receive external input nor produce external output). For the method to work, the thresholding function of each unit must be altered so that it is everywhere differentiable. The most common output function is presented below. For each unit *j*, its output o_{pj} for an input/output pair *p* is:

$$o_{pj} = \frac{1}{1 + e^{-(\sum_i w_{ji} \, o_{pi} + \theta_j)}}$$

where w_{ji} is the weight from unit *i* to unit *j* and θ_j is a tunable "threshold" or bias for unit *j*. The weights (and bias) of each unit are changed after the presentation of each pattern *p* by backpropagating error measures layer by layer from the output back to the input according to the following equations:

$$\Delta_p w_{ji} \;=\; \eta\,\delta_{pj}\,o_{pi} + \alpha\,\Delta_{p-1} w_{ji}$$

where

$$\delta_{pj} \;=\; o_{pj}\,(1 - o_{pj})\,(t_{pj} - o_{pj}) \quad \text{if } j \text{ is an output unit}$$

and

$$\delta_{pj} \;=\; o_{pj}\,(1 - o_{pj}) \sum_k \delta_{pk}\,w_{kj} \quad \text{if } j \text{ is a hidden unit.}$$

Parameter η is called the *learning rate*; α is a parameter called the *momentum term* (which reduces fluctuations during hill climbing); t_{pj} is the target output for the output unit j for pattern p; and δ_{pj} measures the error of the output of unit j for pattern p. Unlike the perceptron, backpropagation may get stuck at a local minima and therefore is not guaranteed to converge to 100% correctness on the training data. In addition, it may get trapped in an oscillation; however, the momentum term helps prevent oscillations.

The version of backpropagation used in the experiments is that supplied with the third volume of the PDP series (McClelland & Rumelhart, 1987). We set the learning rate to 0.25 and the momentum term to 0.9, both of which are standard values. Except as described in Section 3.4, networks contain one output bit for each category. Networks contain one hidden layer which is totally connected to the input and output layers. The number of hidden units used is 10% of the total number of input and output units, a value we empirically found to work well. During testing, an example is assigned to the category whose output unit has the highest value. Training terminates when the network correctly classifies at least 99.5% of the training data or when the number of passes through the data (i.e., the number of *epochs*) reaches 5000. Training on the NETtalk-full data is terminated after 100 epochs due to time restrictions.

2.3. Representation of Examples

We represented the examples in the data sets as bit vectors[1] rather than multi-valued feature vectors (with a slight modification for the numerically-valued features in the heart-disease data). Each input bit corresponds to a possible value of a particular feature and is on or off depending on whether an example has that value for that feature. Normally, exactly one bit in the set of bits representing a single feature will be on. However, if the feature value is missing in the original data (which only occurs in the audiology data), then all the bits in the set are off, and no special processing for missing features is required. (The issue of the representation of missing feature values is further discussed in Section 3.3.2.) The numerically-valued features in the heart-disease data set are treated differently. The values for these numeric features are normalized to

---

[1] The numbers of input and output bits for each domain are as follows: soybeans — 208 and 17; chess — 73 and 2; audiology — 86 and 24; heart disease — 26 and 2; NETtalk — 189 and 115.

be real numbers in the range [0,1]; the minimum value for each feature is mapped to zero and the maximum to one.

Binary encodings are a natural representation for neural learning algorithms. We found that binary encoding slightly improved classification performance of ID3, and therefore we used it for all of the learning algorithms. Since bit vectors are usable by all three systems, their use helps standardize the experimental setup.

ID3's improved performance with the binary encoding may be due to several factors. First, since every feature has only two values, the binary encoding eliminates the gain criterion's undesirable preference for many-valued features (Quinlan, 1986a). Second, it allows for more general branching decisions such as *red* vs. *non-red* instead of requiring nodes to branch on all values of a feature. This may help overcome the *irrelevant values problem* (Cheng, Fayyad, Irani, & Qian, 1988) and result in more general and therefore better decision trees. However, since the binary-encoded examples have more features, ID3's run time is increased somewhat under this approach.

3. EXPERIMENTS AND RESULTS

We report four experiments in this section. The first experiment compares the learning times and accuracies of the three algorithms. Relative performance as a function of the amount of training data is studied in the second experiment. The third experiment investigates the performance of the learning algorithms in the presence of three types of imperfect data. In the final experiment, we investigate the value of distributed output encodings.

To perform the experiments, we separated each data set into a collection of training and testing sets. After each system processes a training set, its performance is measured on the corresponding test set. To reduce statistical fluctuations, except for NETtalk-full, results are averaged over several different training and testing sets. In all of our experiments, two-thirds of the examples in each category are randomly placed in the training set, and the others are placed in the corresponding test set. Again to reduce statistical fluctuations, each run of backpropagation uses a different seed random number when determining the initial random network weights.

For NETtalk-full, we only used a single training set. However, for all runs where backpropagation processes the NETtalk-full data, we ran backpropagation five times, with a different seed random number for each run. The network that performs the best on the training set is used to classify the corresponding test set.

The initial random choice of weights can effect the performance of backpropagation. In our tests with the domains other than NETtalk, local minima problems were not found to occur. Rather, there often are long plateaus where correctness remains constant before again rising.

Since only a small number of training epochs are possible with the NETtalk data, we used five randomly chosen initial states in order to minimize the effect of starting out in a bad state.

3.1. Experiment One: Learning Time and Correctness

Two major issues in inductive learning are the time spent learning and the classification accuracy on novel examples. Except for NETtalk-full, each data set is randomly permuted and divided into training and test sets ten times for this experiment. The training sets are processed until the learning algorithms terminate and then correctness is measured on the corresponding test sets.

3.1.1. Results

Figures 1 and 2 contain the experimental results. The first figure reports the training times of the three systems (normalized to the time taken by ID3). Correctness on the test data is reported in Figure 2. The numbers reported are the means over the ten partitionings of the data sets; the actual numbers, their standard deviations, and several other statistics appear in the appendix. The geometric means of the training times for soybeans, chess, audiology, heart disease, and NETtalk-A are 98 seconds (ID3), 130 seconds (perceptron), and 15,100 seconds (backpropagation). For correctness, the geometric means are 78.5% (ID3), 73.8% (perceptron), and 82.2% (backpropagation). The backpropagation time for NETtalk-full is for *one* run, although the test set correctness is that of the network (out of five) that performed best on the training set.

To isolate the source of variation in the correctness results, a two-way analysis of variance (ANOVA) is used. For this data, ANOVA compares (a) the variance attributable to different learning methods and (b) the variance attributable to different training sets to (c) the variance that cannot be attributed to either of these sources. This analysis indicates that the learning algorithm chosen is a significant cause of the variations in correctness for every data set. Confidence in this conclusion ranges from a low of 95% for audiology to greater than 99.5% for the other data sets. Difference among the ten training sets is a significant source of variation for soybeans (with greater than 99% confidence) and audiology (with 95% confidence). For the other three data sets, this test neither confirms nor rejects the hypothesis that differences in training sets contribute significantly to the variation in the data.

In addition, we used a statistical *t*-test for paired differences to compare all possible pairs of systems. On the heart data, it can be concluded with 99% confidence that (a) backpropagation is superior to both ID3 and perceptron and (b) that ID3 is superior to perceptron. The *t*-test results do not support such a total ordering on the other data sets. On the soybean data, it can be concluded with 99.9% confidence that both backpropagation and perceptron are superior to ID3.

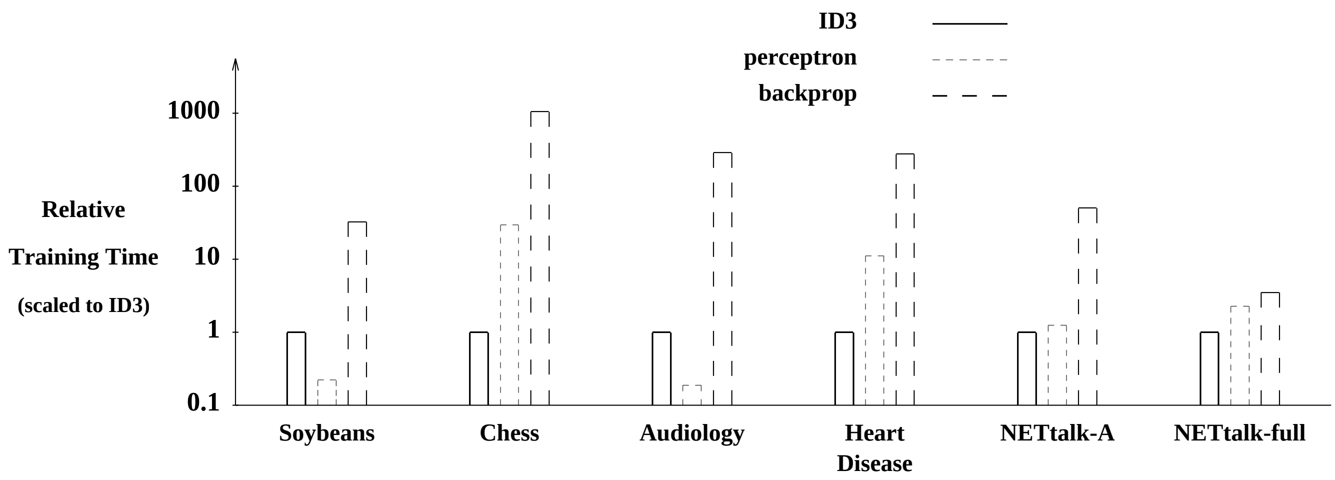

Figure 1. Relative training times of the three algorithms.

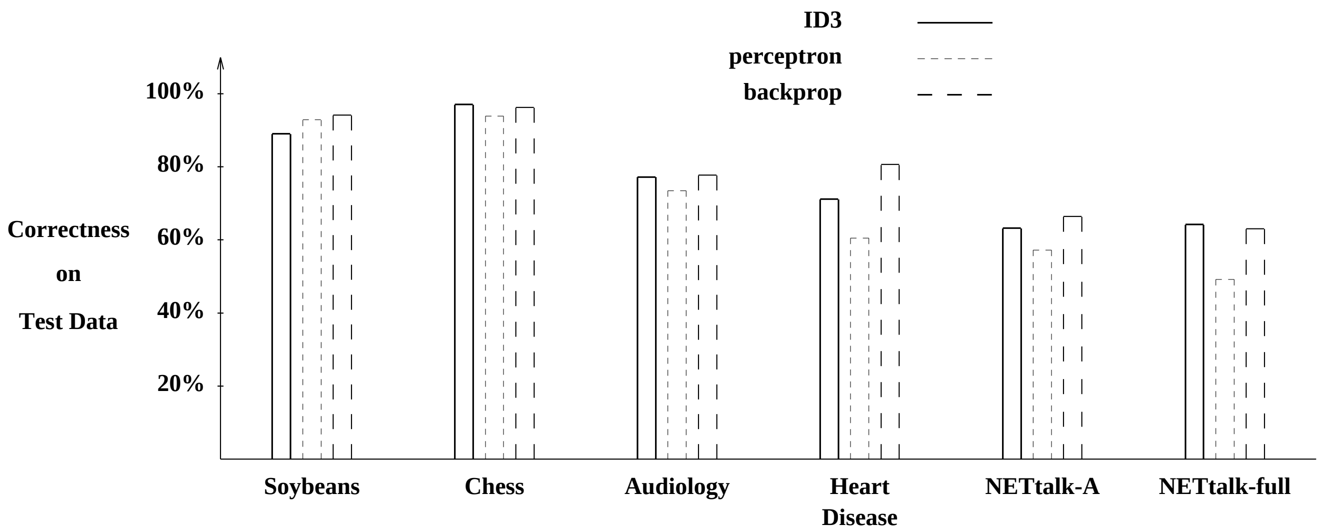

Figure 2. Accuracies of the three algorithms.

On the chess, audiology, and NETtalk-A data, it can be concluded with at least 95% confidence that both backpropagation and ID3 are superior to perceptron. All other pair-wise differences are not statistically significant.

3.1.2. Discussion

In this experiment, for the most part all three learning systems are similar with respect to accurately classifying novel examples. Accuracy shortcomings occur for perceptron on the heart disease and NETtalk data sets, which are not even close to being linearly separable. Compared to backpropagation, ID3 performs poorly on the heart-disease data set, which contains many numerically-valued features. Otherwise all of the systems are within five percentage points of each other. However, ID3 and perceptron train much faster than backpropagation. Despite the obvious differences between decision trees and neural networks, their ability to accurately represent concepts and correctly classify novel instances is often quite comparable. Data from other recent experiments supports this conclusion.

Fisher and McKusick (1989) compared ID3 and backpropagation on two natural domains. In both cases they found backpropagation was about three percentage points more accurate. Weiss and Kapouleas (1989) compared backpropagation to CART (Breiman, Friedman, Olshen, & Stone, 1984), an algorithm similar to ID3 that also produces decision trees. They studied four data sets; CART outperformed backpropagation on two data sets and on the other two, backpropagation worked better. They also considered their PVM algorithm (Weiss, Galen, & Tadepalli, 1987), which produces classification rules. PVM beat backpropagation on three out of the four domains, but never by more than a few percentage points. Atlas et al. (1990) also compared CART to backpropagation on three data sets. In all three cases, backpropagation was more accurate. However, only once was the difference statistically significant. As noted previously, we found that ID3 can improve slightly when examples are converted to a binary representation. As best we can tell, these other studies did not use a binary representation with their decision-tree builders; hence, the differences they found may well be reduced by doing so. Finally, all of these studies mention the slowness of backpropagation.

One possible explanation for the similarity in performance is that most reasonable generalization procedures that correctly classify a particular set of training instances (possibly allowing for some noise) are about equally likely to classify a novel instance correctly. This is consistent with recent work in computational learning theory (Blumer, Ehrenfeucht, Haussler, & Warmuth, 1987) which states, roughly, that any polynomial algorithm that compresses the training data is a polynomial learner in the sense of Valiant (1984). That is, any algorithm guaranteed to compress the information present in the training set is guaranteed to learn a probably approximately correct (PAC) concept and therefore, on average, will perform well on novel test data.

Another possible explanation is that the inductive biases inherent in connectionist and symbolic representations and algorithms reflect implicit biases in real categories equally well. For example, all three systems share some form of an “Occam’s Razor” bias and, at least to some degree, prefer simpler hypotheses. However, for standard neural network approaches, the complexity of the hypothesis is constrained by the user who must initially select an appropriate network for the problem. Unlike most symbolic learning systems which explicitly search for a simple hypothesis, neural systems simply search for a correct hypothesis which fits into a user-specified network. Although both generally use hill-climbing search to guide learning, neural systems hill climb in “correctness space” while symbolic systems also hill climb in “simplicity space.” There are recent interesting developments involving neural learning algorithms that explicitly try to learn simple networks, for example by eliminating unnecessary hidden units or slowly adding hidden units as they are needed (e.g., Hanson & Pratt, 1988; Honavar & Uhr, 1988; Ash, 1989; Le Cun, Denker, & Solla, 1990). Such systems help ease the burden of having to initially specify an appropriate network for a problem.

On the heart-disease data set, backpropagation performed substantially more accurately than ID3. As previously mentioned, this is the only data set in our study that contains numerically-valued features. Hence, our results suggest that backpropagation works better than ID3 when the training examples containing numeric values. This conclusion is corroborated by the study of Atlas et al. (1990). Two of their three data sets contain numeric features, including the one where the accuracy superiority of backpropagation over CART is statistically significant. However, Weiss and Kapouleas’ (1989) study also involved numeric features. Each of their four data sets contains numeric values, but twice the decision tree-building CART algorithm produced a more accurate result than did backpropagation. Thus, more detailed study is required to understand the relative merits of the symbolic and connectionist approaches to learning in the presence of numeric features. Quinlan (1987a) proposed a method for learning “soft thresholds” for decisions involving numerically-valued features. We briefly investigated the use of soft thresholds in ID3 and found no statistically significant differences on the heart-disease data.

Another recent study found a substantial accuracy difference between backpropagation and ID3. Towell, Shavlik, and Noordewier (1990) investigate learning how to recognize an biologically-interesting class of DNA sequences called *promoters*. In their experiment, which did not involve numeric features, backpropagation’s test set accuracy was 92%, while ID3’s accuracy was only 82%. As their DNA data set contained only 106 examples, one explanation of the accuracy difference is that backpropagation outperforms ID3 on “small” data sets; we investigate this issue of training set size in our next experiment. A second, and more compelling, explanation is that the DNA task involves learning a concept of the form *X of these N predicates must be true*. Fisher and McKusick (1989) report that backpropagation performs better than ID3 on problems containing *X of N* functions. Some aspects of the promoter task fit this format. For example, there are several potential sites where hydrogen bonds can form between the DNA and a protein; if enough of these bonds form, promoter activity can occur.

A final point of discussion concerns the performance of backpropagation on the NETtalk-full data set. On this data set, backpropagation learns the training set less well than on the other four data sets. It is possible that with a different number of hidden units or if allowed more than 100 training epochs, backpropagation would have done better on the NETtalk-full test set. However, between epochs 75 and 100, the training set performance of backpropagation never improved more than one percentage point; hence, it may require a prohibitive amount of training to substantially improve backpropagation's performance on NETtalk-full.

3.2. Experiment Two: Effect of the Number of Training Examples

It is possible that some learning methods perform relatively better with a small amount of training data while others perform relatively better on large training sets. To address this issue, this experiment presents graphs that plot correctness as a function of training set size. These "learning curves" are generated by performing batch training on the first *N* examples in each training set as *N* is gradually increased. Hence, subsequent training sets of increasing size subsume one another. Learning is non-incremental; for example, there is no carryover from the learning on 50 training examples to the learning on 100 training examples. Although learning in this experiment is not incremental, the results provide an upper bound on the performance, in terms of correctness, that can be expected from incremental algorithms. Fisher, McKusick, Mooney, Shavlik, and Towell (1989) provide a discussion of several ways to apply backpropagation in an incremental setting.

3.2.1. Results

Figure 3 presents correctness as a function of the amount of training data. Each data point represents the average of three (rather than ten, due to simulation time restrictions) randomly chosen training sets, except the NETtalk curve represents training on only a single data set. The correctness expected from random guessing is plotted when there are zero training examples.

On the soybean and NETtalk data, both backpropagation and the perceptron perform better than ID3 when given only a small amount of training data. Backpropagation consistently outperforms the other two systems on the heart-disease data set. Finally, there is little difference among the three systems on the chess and audiology data.

3.2.2. Discussion

The results indicate that for small amounts of training data backpropagation is perhaps a better choice. With small training sets, the slower speed of backpropagation is much less of a problem, and on the five sample domains studied backpropagation did about the same as or better than ID3. The results also suggest that if ID3 does relatively well on a small sample, then the faster ID3 should be used on the full training set. Since backpropagation takes so long to

Figure 3. Classification performance as a function of amount of training data.

run, this technique can have value in practical systems.

Perceptron's accuracy on the heart-disease data declines slightly as the number of examples increases. This can be attributed to the fact that perceptron terminates once it decides that the data set is not linearly separable. Perceptron's performance on both the heart-disease and NETtalk data could be improved by having it, once it detects an inseparable data set, search for the hyperplane that best separates the training examples.

ID3's tendency to perform worse on small training sets may be related to the *problem of small disjuncts* (Holte, Acker, & Porter, 1989). Holte et al. have shown that leaves in a decision tree (or disjuncts in a DNF rule) that cover only a small number of training examples tend to have much higher error rates than those that cover a large number of examples. They present evidence that this higher error rate is due to the *maximum generality bias* used by most symbolic induction systems. The basic problem is that the simplest conjunction that covers a small set of examples is frequently an overgeneralization. Given a small training set, ID3 will build trees composed primarily of small disjuncts with high error rates. Unlike most symbolic systems, backpropagation and the perceptron do not build explicit disjuncts and do not have a maximum generality bias; therefore, they may be less susceptible to the problem of small disjuncts. Also, note that data with a large number of categories will have fewer examples per category and therefore generate smaller disjuncts. This would explain why ID3 does better on the chess data, which has only two classes. It also explains why ID3 performs as well as backpropagation on the audiology data; although there are 24 audiology categories, about three-fourths of the examples fall into one of five classes.

Fisher and McKusick (1989) suggest that ID3 may be more effective than backpropagation on relatively small training sets, but as the size of the training set increases, backpropagation eventually outperforms ID3. This discrepancy results from a procedural difference. When generating learning curves, Fisher and McKusick use backpropagation in an incremental fashion and update the weights only once for each new training example. In the learning curves presented in Figure 3, backpropagation is allowed to converge on all of the training examples so far encountered before it is run on the test set. Running backpropagation to convergence leads to much better performance on small amounts of training data.

3.3. Experiment Three: Effect of Imperfect Data Sets

An important aspect of inductive learning systems is their sensitivity to imperfections in the data. Improperly represented examples can occur for several reasons. Mistakes may be made when recording feature values or when judging the class of an example. Some feature values may be missing or an insufficient collection of features may be used to describe examples.

Experiments described in this section investigate various types of imperfections and report their effect on the three learning algorithms. Noisy, incomplete, and reduced data sets are produced and the performance of the three learning algorithms compared. Classification performance after learning on corrupted data is measured using test sets that have the same types and rates of imperfections as the training data.[2] This is consistent with work on the theoretical basis of learning (Valiant, 1984), which assumes that the distribution of examples is the same during training and testing.

Due to processing limitations, in these experiments we used a reduced version of the NETtalk-full training set. Rather than using the 808-word training set, only the first 256 words (1099 examples) are used in training. Except for this data set, called NETtalk256, all the curves in this experiment are the result of averaging performance on three random training/test divisions.

3.3.1. Random Noise

The first type of imperfection investigated will be called *random noise.* This is the noise produced when feature values and example classifications are mistakenly recorded. With probability p, a given feature value is randomly changed to another legal value for that feature. With the same probability, the classification of the example is randomly changed to a different category. We replaced heart disease's numeric features by randomly choosing uniformly from the interval [0,1].

Note that noise is introduced at the feature level. That is, noise is added to data sets before they are converted to the binary representation. For instance, at a 75% noise level in the NETtalk data, 75% of the feature values are different but less than 6% of the bits actually change (each feature comprises 27 bits, all but one of which are zero, so to change a feature value requires that only two bits change their values).

Results

Figure 4 presents the effect of the amount of random noise on classification performance. (The same relative performance is obtained when there is only feature noise, i.e., when no classification errors are introduced.) The thin dotted lines on each of these graphs represents the frequency of the most common item in the training set. Thus, this line represents the level of performance obtainable by merely guessing the most common category in the training set.

---

[2] There is one exception to this. In one experiment's training set (Section 3.3.1), the classifications of examples are randomly corrupted. The classifications in the corresponding test sets, against which the accuracies of the learning algorithms are measured, are *not* corrupted.

While in one domain (audiology) ID3 outperforms the other two systems, in two others (soybeans and chess) backpropagation is the best. ID3 and backpropagation degrade at roughly the same rate in the heart-disease and NETtalk domains.

Discussion

Overall, backpropagation appears to handle random noise slightly better than ID3 does. An explanation for this difference between ID3 and backpropagation is that backpropagation makes its decisions by simultaneously weighting all the input features, while ID3 sequentially considers the values of input features as it traverses the decision trees is learns. Hence, a single noisy feature value early in an ID3 decision tree could significantly impact classification. Our results mildly disagree with Fisher and McKusick's (1989) study of the comparative effect of noise. They report that backpropagation *consistently* outperforms ID3, while our results indicate that sometimes ID3 can handle noise as well as or better than backpropagation.

Recently, several alternatives to chi-squared pruning have been proposed for handling noisy data in ID3 (Quinlan, 1987b; Mingers 1989). These are generally referred to as *post-pruning* methods because they prune the tree after it is built rather than terminating growth during construction. These newer methods of handling noise in ID3 may be more competitive with backpropagation. However, backpropagation has the advantage that it handles noise naturally without requiring such special-purpose procedures.

3.3.2. Missing Feature Values

A second type of data set imperfection occurs when feature values are missing. The presence of incompletely described examples complicates learning, as the information necessary to distinguish two examples may be absent. To build data sets with missing feature values, with probability p a given feature value is deleted.

Methods for Handling Missing Values

Learning from incomplete data requires an effective method for handling missing feature values. Several techniques have been developed for dealing with unknown attribute values in ID3. Quinlan (1989) presents a thorough set of experiments comparing the various approaches. Our version of ID3 uses a method that is arguably the best according to the results of these experiments. When evaluating a potential splitting feature, the system adjusts its information gain by distributing examples with unknown values across the possible values according to their frequency (Quinlan, 1986b). When considering partitioning the training set along a particular attribute if a training example has an unknown value for this attribute, a fraction of the example is assigned to each subset based on the frequency of its corresponding value (Kononenko, Bratko, & Roskar, 1984). When classifying a new case with an unknown value for the attribute

Figure 4. Classification performance as a function of amount of feature and category noise.

being tested, all branches are explored and the results are combined to reflect the relative probabilities of the different outcomes (Quinlan, 1986b). Our own experimental results confirm that this approach is superior to other methods such as replacing unknown values with the most common value or discarding examples with unknown values during partitioning.

However, while this method leads to higher accuracies on novel examples, it has one drawback; processing time grows exponentially as a function of the number of missing feature values. One approach to reducing ID3's run time is to prevent splitting on features that create a partition containing less than a total of one example (Quinlan, personal communication). This frequently (although not always) decreases run time a substantial amount but also tends to reduce accuracy by a few percentage points. Even with this addition, run time grows exponentially in the number of missing values and becomes intractable for large data sets with large amounts of missing data. Unfortunately, for this reason this technique could not be applied to the NETtalk256 data set. Hence, to apply ID3 to the NETtalk data, we replaced unknown values with the most common value for an attribute, given the example's classification. In our experiments we found that on the other data sets this method lead to only a small degradation in test set accuracy.

To the best of our knowledge, the issue of representing missing values in neural networks has not been investigated previously. We evaluated three representations of missing features for backpropagation. Assume that there are *N* possible values for some feature; hence, we used *N* inputs to represent the value of this feature. If the value is missing, these inputs could all be zero, all be 0.5, or all be *1/Nth*. The first representation reflects the fact that none of the possible values are known to be present. The second uses the intermediate input value of 0.5 to represent unspecified inputs, while the third reflects the *a priori* probability that a given input is a one. In a sense, the third approach "spreads" across all *N* inputs the single "one" used to indicate the feature value. (This third approach could be extended by distributing the single "one" according to the distribution of the feature's values in the training set; however, we did not investigate this approach.)

Performance of these three approaches, averaged over the soybeans, chess, and audiology data sets, is reported in Figure 5. The reason that *1/Nth* works best may be that the other two techniques provide too little or too much input "activity" at features whose value is missing. Since *1/Nth* works best, we used it for backpropagation in the remainder of this experiment. For heart disease's numeric features, we replaced missing features with the value 0.5.

We also tested perceptron with the same three approaches and the *1/Nth* approach works best. Therefore, this method is also used with the perceptron.

The file for this figure has been lost.

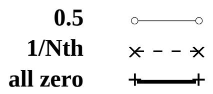

Figure 5. Effect of representation of missing features on backpropagation performance.

Earlier we mentioned that the audiology data set contains many missing feature values. However, these features are not missing because they were "lost." Rather, "missing" in this domain means that the expert producing the example decided that the feature was *irrelevant*, given the other details of the case (Duran, 1988). Our experiments indicated that it is best to represent this type of absent feature value by all zeroes. To produce the curves below, we represented only the *randomly* discarded feature values using the techniques described above.

Results

Figure 6 contains the performance of the three learning algorithms as a function of the percentage of missing features values. ID3 and perceptron perform poorly on some of the data sets, while backpropagation performs well across all four domains. Once again, ID3 performs substantially worse than the other methods on the soybean data. Perceptron performs poorly on the chess data. Surprisingly, perceptron outperforms the other two systems on the audiology data.

Discussion

The results indicate that backpropagation handles missing feature values better than ID3 and perceptron. Perceptron frequently does poorly because complicated representations of missing values lead to concepts that are not linearly separable. The reason for backpropagation outperforming ID3 may be that backpropagation naturally supports the representation of partial evidence for a feature's value. As discussed in the analysis of performance on noisy data,

Figure 6. Classification performance as a function of amount of missing feature values.

backpropagation makes decisions by summing weighted evidence across many features, while ID3 locally bases its decisions on the value of a single feature. It appears missing feature values are most naturally matched by the inductive biases inherent in backpropagation.

An interesting question is why ID3 consistently does poorly on the soybean data. Possibly its high number of input features (more than that of any other domain), large number of categories, and even distribution of examples across categories interacts poorly with ID3's information gain measurement. In each training set there are only 11 examples for each of 17 categories.

3.3.3. Completely Dropped Features

A third type of imperfection arises when an insufficiently rich vocabulary is used to represent examples. Features that would simplify classification may not have been included when the examples were produced. We investigate sensitivity of the learning algorithms to the number of feature by randomly dropping a percentage of the features; if a feature is dropped, it is dropped from *all* examples in the training set and in the corresponding test set. Except for NETtalk256, we used three different training sets and chose a different random collection of dropped features for each set. For NETtalk256, which has a clear ordering on its features, letters are dropped from both ends of the example window until a sufficient number of features are discarded. (When an odd number of features are discarded, two training sets are produced. In one case the extra letter is dropped from the front and in the other it is dropped from the back. The results are then averaged.)

Results

Figure 7 presents the results of the experiment where some fraction of the features are completely dropped. ID3 and backpropagation degrade at roughly the same rate as the number of features is reduced, while for large numbers of dropped features perceptron degrades more drastically.

Discussion

Both ID3 and backpropagation do surprisingly well as the number of features is reduced. Perceptron's poor performance is due to the training sets becoming linearly inseparable. One interesting aspect of Figure 7 is the apparent redundancies in several of the domains. Randomly dropping half the features only slightly impairs performance. Also interesting are the occasions where performance *improves* when a small number of features are dropped, illustrating that extra features can degrade inductive learning algorithms. For example, on the NETtalk256 data set, ID3 performs best when it only considers a three-letter window.

Figure 7. Classification performance as a function of number of features totally dropped.

3.4. Experiment Four: Effect of Output Encoding of NETtalk Examples

Often in neural network systems, concept representations are distributed over a collection of units. For example, Sejnowski and Rosenberg (1987) encode the phoneme/stress outputs in NETtalk-full as a sequence of 26 bits (as opposed to one bit for each of the 115 categories). The first 21 bits are a distributed encoding of the 51 phonemes contained in the dictionary. Each bit is a unary feature describing one of: position of the tongue in the mouth, phoneme type, vowel height, or punctuation. The remaining five bits form a local encoding of the five types of stresses used in the dictionary.

In our final experiment, we repeat Sejnowski's methodology with the NETtalk-full data set to investigate the merits of distributed output encodings and to ascertain which learning algorithms can best utilize this type of output representation. This experiment compares classification performance after learning using the localized method of output encoding to the distributed encoding. The output encoding experiment only involves the NETtalk data set, as this is the only data set whose producers rendered a distributed output encoding. Hence, note that using a distributed output vector requires more information about the data sets.

3.4.1. Methodology

The backpropagation algorithm easily handles this distributed encoding of the output. For the backpropagation tests involving distributed outputs, we used 120 hidden units, 26 output units, and 30 training epochs, following Sejnowski and Rosenberg (1987). While backpropagation handles the distributed output representation naturally, to implement this representation in ID3 requires building a separate decision tree for each of the 26 output bits. Similarly, to implement this representation in perceptron, one perceptron is learned for each of the 26 output bits.

To categorize a test example, a string of 26 outputs is formed by either a single pass through a backpropagation network or by passes through each of the 26 decision trees or perceptrons. Backpropagation produces a vector containing numbers between 0 and 1. Perceptron produces a vector of numbers, where each entry is a perceptron's output minus its threshold. As we discuss in the next section, ID3 produces either a binary vector or a vector of numbers between 0 and 1. The specified category is then determined by finding the phoneme/stress pair encoding that has the smallest angle with the 26 outputs. (This is the "best guess" metric used by Sejnowski and Rosenberg, 1987.)

3.4.2. Results

Figure 8 compares the classification performance of the learning systems on the NETtalk data using the the local output encoding and the distributed output encoding. All training for this table is done using the 808-word set. Classification accuracy is measured using the 1000-word test set described previously.

In this experiment, we initially used the chi-squared version of ID3 and got poor performance (82% correctness on the training set and 49% correctness on the test set). We next investigated the hypothesis that producing a binary vector before making a best guess lead to a loss of useful information; knowing the amount of evidence for each output value may be advantageous. To answer this question, we applied Buntine's (1989) method for giving a probabilistic interpretation of the outputs. Using, as input to the "best guess" routine, the 26 probabilities produced by this method results in a training set correctness of 89% and a test set correctness of 56%. Next, we turned off ID3's chi-squared pruning and fitted the data as closely as possible, getting 97% correctness on the training set and 62% on the test set. Using Buntine's classification method with these trees (i.e., those built without using chi-squared pruning) produced the best results. With this configuration, we got a training set accuracy of 98% and a test set accuracy of 65%; Figure 8's statistics refer to this final method.

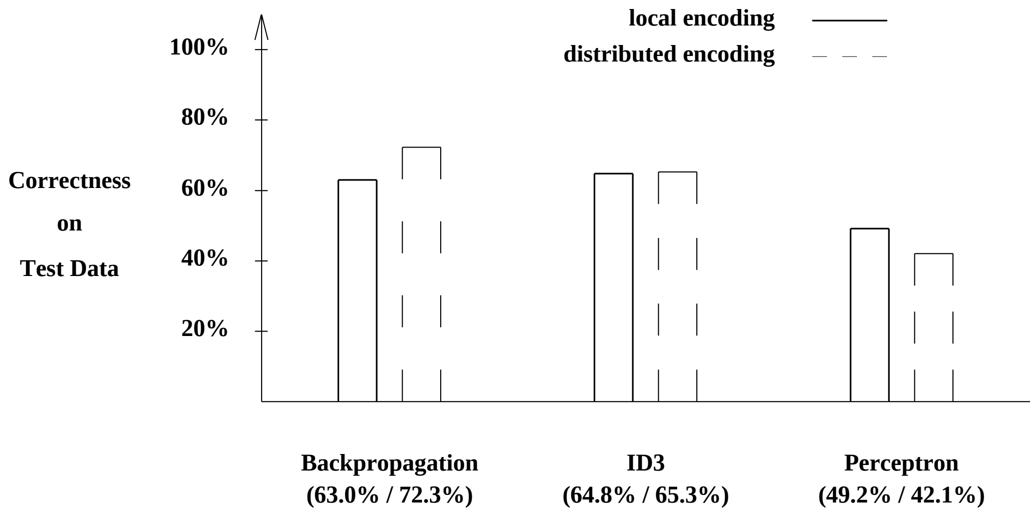

Figure 8. Classification correctness on the NETtalk data set for two output encoding methods.

3.4.3. Discussion

The use of a distributed encoding of the output substantially improves the performance of backpropagation. For the distributed encoding, backpropagation's correctness is 96% on the training set and 72% on the testing set. Using the local encoding — one output bit for each category — results in a backpropgation correctness of 88% percent on the training set and 63% on the testing set. ID3 is also able to successfully use the distributed encoding, but its performance with the two encoding styles are roughly equivalent (at about a 65% test-set accuracy). With the distributed encoding, backpropagation performs substantially better than ID3 on the NETtalk task. Dietterich et al. (1990) report similar results for their comparison of ID3 and backpropagation on the NETtalk data.

Perceptron has no similar improvement in performance and, instead, degrades. This occurs because the 115 individual concepts are easier to learn than the 26 concepts. With the local encoding, perceptron converged on 60 out of the 115 bits, while on the distributed encoding it converged on only four.

There is a cost of using the distributed encoding with backpropagation. The range of correctness varies widely when the training data is imperfect. Using the NETtalk256 data corrupted with 10% noise, backpropagation's correctness on three runs ranged from 27% to 44% when we only varied the seed random number. With 25% noise, the accuracy on three runs ranged from 0.1% to 19%. The problem became worse as we increased the noise level. On all other data sets, and on NETtalk using the local output encoding, there was only a little variation in classification performance between different randomly-initialized networks. The use of the distributed encoding seems to introduce problems with local minima that are absent with the local encoding.

Dietterich et al. (1990) performed an extensive series of tests in an attempt to isolate the reasons for the superiority of backpropagation over ID3 under the distributed output encoding. They considered three hypotheses that could, individually or jointly, account for the differences between ID3 and backpropagation. Their experiments indicate that neither of the following hypotheses account for the observed differences between ID3 and backpropagation: (a) ID3 overfits the training data; (b) backpropagation's ability to share intermediate conclusions in its hidden units is a factor. They concluded that the primary difference was that backpropagation's numerical parameters (e.g., its continuous output values) allow it to capture statistical information missed by ID3. They propose a "block decoding" scheme for ID3 that substantially reduces the accuracy difference between ID3 and backpropagation.

4. GENERAL DISCUSSION

This study investigates the relative performance of symbolic and neural approaches to learning. The first experiment shows that backpropagation performs slightly better than the other two algorithms in terms of classification correctness on novel examples, but takes much longer to train. Results on the heart-disease data set suggest that backpropagation may perform substantially better than ID3 when there are numerically-valued features. The result of the second experiment is that the neural algorithms tend to perform more accurately than ID3 when given relatively small amounts of training data. The third experiment, which investigates the handling of imperfect data sets, indicates that backpropagation handles noisy and missing feature values slightly better than ID3 and perceptron do. ID3 and backpropagation perform about the same in the presence of reduced numbers of features. In the fourth experiment, backpropagation is better able to utilize a distributed output encoding, although ID3 is also able to take advantage of this representation style. We discuss several additional aspects of the experiments in this section.

4.1. The Symbolic/Subsymbolic Distinction

A supposed difference between traditional symbolic and neural-net learning systems is that the latter operate at a distinct “subsymbolic” level (Smolensky, 1988). The experiments reported in the previous section demonstrate that this distinction can be misleading (Fodor and Pylyshyn (1988) and Langley (1989) also discuss this issue). “Symbolic” systems like ID3 and “subsymbolic” systems like backpropagation are essentially solving the same learning problem and each can be applied to problems typically used to test the other. They both take a set of input vectors and produce a classification function. Both accept input features of any grain size; the ability to use “microfeatures” is not unique to neural nets. As demonstrated in Section 3.4, both can also use distributed representations; however, current neural-net methods such as backpropagation seem to be able to take greater advantage of distributed representations than current symbolic systems like ID3.

It is also important to note that, assuming the initial inputs and final outputs are finite-valued, different concept representations such as decision trees, disjunctive normal form (DNF) expressions, and multi-layer neural nets are all equivalent in terms of representational power. Each of these formalisms can represent all 2^{2^N} functions on N binary inputs. This is a well-known property of DNF expressions (i.e. two-layer $AND-OR$ networks; Muroga, 1979) and any DNF expression can be easily converted into an equivalent decision tree. Since the weights of a linear threshold unit can be set to model *AND*, *OR*, and *NOT* gates, assuming a sufficient number of hidden units are available, any DNF expression can also be converted into an equivalent neural net. Consequently, every neural net implicitly represents some logical rule (i.e. one of the possible 2^{2^N} functions); decision trees and DNF formulae are simply more explicit representations of such rules.

The difference between symbolic and connectionist systems lies in their inherent inductive biases, which determine which of the many logical rules consistent with a set of data is actually chosen as the concept definition. Most symbolic learning systems prefer syntactically simple rules, while many functions that are easily learned by neural nets correspond to relatively complex DNF formulae or decision trees (e.g. *X of N* functions which output 1 if and only if at least X of the N inputs are 1). Not surprisingly, experiments with artificial data show that backpropagation performs better than ID3 on problems which fit its natural bias, like *X of N* functions (Fisher and McKusick, 1989). However, the current experiments indicate that the inductive biases of symbolic and neural net systems are equally suitable for many “real world” problems.

4.2. Perceptron Performance and Problem Difficulty

One surprising result of these experiments is how well the simple perceptron algorithm performs. The perceptron was largely abandoned as a general learning mechanism over twenty years ago because of its inherent limitations, such as its inability to learn concepts that are not linearly separable (Minsky & Papert, 1988). Nevertheless, it performs quite well in these experiments. Except on the heart-disease and NETtalk data sets, and in the presence of imperfect training data, the accuracy of the perceptron is hardly distinguishable from the more complicated learning algorithms. In addition, it is very efficient; perceptron’s training time is comparable to ID3’s.

These results indicate the presence of a large amount of regularity in the training sets chosen as representative of data previously used to test symbolic learning systems. The categories present in both the soybean and audiology data are linearly separable for all ten randomly chosen training sets. The two categories in the chess data are linearly separable for four of the ten training sets and almost linearly separable on the rest (average correctness on the training sets is 97.5%). Despite the fact that the data sets are large and represent real categories, their regularity makes them relatively “easy” for even simple learning algorithms like the perceptron.

One possible explanation for the regularity in the data is that it is reflecting regularity in the real-world categories. In other words, members of a real category naturally have a great deal in common and are relatively easy to distinguish from examples of other categories. Another more likely explanation is that the features present in the data have been very carefully engineered to reflect important differences in the categories. For example, formulating features for chess end games which were appropriate for learning required considerable human effort (Quinlan, 1983).

More difficult problems need to be studied in order to determine the true power of current inductive learning systems. The NETtalk data set is one of the most difficult ones studied so far and other researchers in machine learning are beginning to use it as a benchmark (e.g., Dietterich et al., 1990).

Regardless of the reason, data for a number of "real" problems seems to consist of linearly separable categories. Since the perceptron is a simple and efficient learning algorithm in this case, using a perceptron as an initial test system is probably a good idea. If a set of categories are not linearly separable, the *perceptron cycling theorem* (Minsky & Papert, 1988) guarantees that the algorithm will eventually repeat the same set of weights and can be terminated. In this case, a more complicated algorithm such as ID3 or backpropagation can be tried. The *perceptron tree error correction procedure* (Utgoff, 1988) is an example of such a hybrid approach. This algorithm first tries the perceptron learning procedure, and if it fails splits the data into subsets using ID3's information-theoretic measure and applies itself recursively to each subset.

4.3. Slowness of Backpropagation

Although backpropagation performs about as well or better than the other systems at classifying novel examples, it consistently takes a lot more time to train. Averaged across all five data sets, in the first experiment backpropagation takes about 150 times as long to train as ID3. (Testing in backpropagation takes about 10 times longer than the other two systems.) These factors would probably increase if we optimized our Common Lisp versions of ID3 and perceptron, which are not coded efficiently compared to the C version of backpropagation.

Researchers are developing faster methods for neural network training (e.g., Fahlman, 1988; Barnard & Cole, 1989). For example, Fahlman's *quickprop* algorithm ran almost an order of magnitude faster than backpropagation on one sample task. Unfortunately, Fahlman does not report how well his algorithm performs relative to backpropagation in terms of accuracy on novel examples.

Another obvious way backpropagation can be made faster is to exploit its intrinsic parallelism. The networks used in these experiments contained an average of about 150 units. Consequently, assuming one processor per unit and perfect speedup, the training time for backpropagation could possibly be made competitive with ID3's. However, ID3 is a recursive divide-and-conquer algorithm and therefore also has a great deal of intrinsic parallelism. In addition, in perceptron each output bit is learned independently, one simple source of parallelism for this method. Comparing the training time for parallel implementations of all three algorithms would be the only fair way to address this issue.

4.4. Scalability

The computational complexity of a learning algorithm is an important theoretical and practical issue. This issue is generally more important than parallelism since a fixed number of processors can at most decrease run time by a constant factor while increasing the size of a problem can increase run time an arbitrary amount depending on the computational complexity of the algorithm.

Learning time for ID3 grows linearly with the number of examples and, in the worst case, quadratically with the number of features (Schlimmer & Fisher, 1986; Utgoff, 1989). However, our learning times for the “dropped features” experiment (Section 3.4.3) support the hypothesis that on average ID3’s learning time grows linearly with the number of features. Therefore, ID3 seems to scale quite well.

There have been speculations that neural networks scale poorly (Minsky & Papert, 1988). Recent theoretical work shows that neural network learning can be NP-complete (Blum & Rivest, 1988; Judd, 1988). There is empirical evidence that, on average, backpropagation’s time to reach asymptotic performance grows as the cube of the number of weights (Hinton, 1989). However, as evidenced on the NETtalk-full data, backpropagation can still produce impressive results even when training is terminated before the training set is fully learned.

4.5. Incremental Learning

One issue we did not investigate is learning incrementally. Incremental versions of ID3 have been proposed (Schlimmer & Fisher, 1986; Utgoff, 1989) and backpropagation can be performed by processing each new example some number of times before discarding it. Comparison of various incremental approaches is another area for future research (Fisher et al., 1989).

4.6. Other Issues

There are several other differences between the three learning algorithms and between symbolic and connectionist approaches in general. For example, if one has a collection of input/output training pairs, one can directly run ID3 and perceptron. On the other hand, in order to run backpropagation, a network architecture must be chosen, currently much more of an art than a science. Not only must one choose the number of hidden units, but the number of hidden layers must also be specified. In addition, one must choose settings for the learning rate and the momentum term. Performance may depend greatly on the initial randomly-selected weights and several runs with different initial weights may be necessary to get a good final result. Finally, a criterion for stopping training must be chosen. If parameters are inappropriately set or if initial weights are unfavorable, backpropagation may fail to converge efficiently. However, it should also be noted that many symbolic learning systems have parameters which one must appropriately set to insure good performance (Rendell et al., 1989).

Another issue is the human interpretability of the acquired rules. Symbolic learning can produce interpretable rules while networks of weights are harder to interpret. However, large decision trees can also be very difficult to interpret (Shapiro, 1987). If the learned classifiers are implemented using unreliable components, sensitivity to component failure can be a concern. Neural models, which sum partial evidence, are robust in the presence of component failure (McClelland, 1986). Sensitivity to component failure is seldom addressed in symbolic systems. Finally, connectionist models are intended to be neurally-plausible models, while symbolic models are not. Hence, connectionist models may shed more light on neurophysiology.

5. FUTURE RESEARCH ISSUES

Our experiments suggest a number of issues for future research. We have already mentioned several interesting research topics: comparing parallel implementations; comparing incremental learning approaches; understanding why ID3 does poorly on the soybean data; further investigating the hypothesis that backpropagation better addresses the problem of *small disjuncts* (Holte et al., 1989). The remainder of this section describes additional topics involving empirical, theoretical, and psychological comparison of symbolic and connectionist learning.

An obvious extension is comparison of additional algorithms and domains. Besides other connectionist and rule-based algorithms, instance-based, genetic, and probabilistic algorithms could be included. Such experiments would hopefully give additional insight into which algorithms are suitable for different types of problems.

Also of interest would be detailed experiments that isolate the algorithmic differences between backpropagation and ID3 that cause the observed performance differences. Each inductive learning algorithm contains some sort of inductive bias, and it is of interest how well these biases work on real-world learning tasks. As mentioned in Section 3.4.3, Dietterich et al. (1990) performed a comparative analysis on the NETtalk data, represented in the distributed encoding. Similar analysis of numerical data, noisy data, data with missing values, and small data sets could lead to additional insights and the development of better learning algorithms for these types of data. The relative performance of ID3 and backpropagation in the presence of numerical data is of particular interest, since the results of our experiments and those of Atlas et al. (1990) and Weiss and Kapouleas (1989) do not provide a clear answer to this question.

The effect of parameter settings on the relative performance of algorithms such as backpropagation is another area for research. Manually tuning parameters to fit a particular set of data can give an algorithm an unfair advantage. Consequently, we did not carefully tune backpropagation's parameters to each data set, but rather adopted a uniform parameter-setting policy after some initial experimentation. Improved policies for setting parameters could result in better performance.

Theoretical analysis of the relative performance of different approaches to concept learning is also warranted. A particularly relevant result would provide a formal characterization of classes of concepts that different types of algorithms are probabilistically guaranteed to learn with high accuracy from few examples. Most theoretical work has focussed on which classes are learnable or unlearnable in polynomial time, rather than on detailed analysis of the relative sample complexity of practical algorithms.

Finally, this paper focuses on evaluating algorithms based on training time and predictive accuracy. Psychological plausibility is another interesting evaluation metric. For example, Pazzani and Dyer (1987) found that backpropagation does not adequately model human data on learning simple conjunctive and disjunctive concepts. Further investigation is needed on the relative ability of symbolic and neural learning algorithms to model various aspects of human learning.

6. CONCLUSION

A current controversy is the relative merits of symbolic and neural network approaches to artificial intelligence. Although, symbolic and connectionist learning systems often address the same task of inductively acquiring concepts from classified examples, their comparative performance has not been adequately investigated. In this article, the performance of a symbolic learning system (ID3) and two connectionist learning systems (perceptron and backpropagation) are compared on five real-world data sets. These data sets have been used in previous experiments. Four in symbolic learning research (soybeans, chess, audiology, and heart-disease) and one in neural network research (NETtalk).

There are several contributions of the reported experiments:

- Experimental results indicate that ID3 and perceptron run significantly faster than does backpropagation, both during learning and during classification of novel examples. Except for ID3 and perceptron on the heart-disease data and perceptron on the NETtalk data, the probability of correctly classifying new examples is about the same for the three systems.
- Backpropagation's performance on the heart-disease data set suggests it outperforms ID3 on data sets that contain numerically-valued features. This conclusion is supported by the experiments of Atlas et al. (1990) and contradicted by the results of Weiss and Kapouleas (1989).
- Empirically evidence indicates that, when given "small" amounts of training data, backpropagation occasionally learns more accurate classifiers than ID3 does. Also,

backpropagation's slowness is less of a concern on small data sets.

- Additional experiments, in which the sample data sets are carefully corrupted, suggest that with noisy data sets backpropagation slightly outperforms ID3. However, in one domain (audiology) ID3 was more robust to noise.
- When examples are incompletely specified, backpropagation also performs slightly better than the other two approaches. A technique for representing missing feature values in neural networks is proposed and shown effective.
- ID3 and backpropagation appear to be equally sensitive to reductions in the number of features used to express examples. Surprisingly, often *half* the features could be randomly dropped without substantially impacting classification accuracy.
- One claimed advantage of neural approaches to learning is that they are able to profitably use distributed encodings. Empirical study of the NETtalk data set suggests that backpropagation is able to take advantage of domain-specific distributed output encodings better than the two other systems, although ID3 is able to successfully use distributed output encodings.

In conclusion, this study provides the beginnings of a better understanding of the relative strengths and weaknesses of the symbolic and neural approaches to machine learning. We hope it provides results against which future learning algorithms can be evaluated.

Acknowledgments

The comments and suggestions of editor Ross Quinlan and an anonymous reviewer are gratefully acknowledged. The authors would also like to thank the following people for supplying data sets: Bob Stepp and Bob Reinke for the soybean data; Ray Bareiss, Bruce Porter, and Craig Wier for the audiology data which was collected with the help of Professor James Jerger of the Baylor College of Medicine; Dr. Robert Detrano of the Cleveland Clinic Foundation for the heart disease data set (which was obtained from the University of California — Irvine's repository of machine learning data bases, managed by David Aha); Rob Holte and Peter Clark for Alen Shapiro's chess data; and Terry Sejnowski for the NETtalk data. Alan Gove ran many of the preliminary experiments and assisted in converting the data sets into a common format. Elizabeth Towell assisted with the analysis of variance. Rita Duran, Wan Yik Lee, and Richard Maclin contributed to the implementations. The Condor system (Litzkow, Livny, & Mutka, 1988), which runs Unix jobs on idle workstations, provided many of the large number of cycles needed for these experiments. The equivalent of over two Sun-4/110 CPU *years* was provided by Condor.

This research was partially supported by the University of Wisconsin Graduate School, the University of Texas at Austin's Department of Computer Sciences and Artificial Intelligence Laboratory, Office of Naval Research Grant N00014-90-J-1941 to Shavlik, and NASA-Ames Grant NCC-2-629 to Mooney.

This paper's results extend and supercede those previously reported at the Eleventh International Conference on Artificial Intelligence and in University of Wisconsin Technical Report #857.

Appendix

Table 1 contains the arithmetic means and standard deviations of the results produced on the ten data sets for each domain (Section 3.1). For perceptron, the number of epochs refers to the mean number of cycles taken per category. NETtalk-full only involves one training set and, hence, no standard deviations are reported. Statistically significantly different test set accuracies are reported; all other pair-wise comparisons between test set accuracies are not statistically significant.