SYMBOLIC AND NEURAL LEARNING ALGORITHMS: AN EXPERIMENTAL COMPARISON

by

Jude W. Shavlik
Raymond J. Mooney
Geoffrey G. Towell

Computer Sciences Technical Report #857

June 1989

Symbolic and Neural Learning Algorithms: An Experimental Comparison

Jude W. Shavlik

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

shavlik@cs.wisc.edu
(608) 262-7784

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, back-propagation

(Submitted to *Machine Learning*)

Symbolic and Neural Learning Algorithms: An Experimental Comparison

Abstract

Despite the fact that many symbolic and neural network (connectionist) learning algorithms are addressing 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 back-propagation neural learning algorithms have been performed using several large real-world data sets. Back-propagation performs about the same as the other two algorithms in terms of classification correctness on new examples, but takes much longer to train. The effects of the amount of training data, imperfect training examples, and the encoding of the desired outputs are also empirically analyzed. Suggestions for handling imperfect data sets are described and empirically justified. Symbolic and neural approaches work equally well in the presence of noise, while back-propagation does better when examples are incompletely specified. Back-propagation is better able to utilize a distributed output encoding, although ID3 is also able to take advantage of this representation style.

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 [Fisher89a, Mooney89, Weiss89] 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 labelled 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.

Symbolic and neural systems for learning from examples generally require the same input, namely, classified examples represented as feature vectors. In addition, their performance is generally evaluated in the same manner, namely, by testing their ability to correctly classify novel examples. Within symbolic machine learning, numerous algorithms have been developed for learning decision trees [Quinlan86] or logical concept definitions [Michalski83, Mitchell82] from examples, both of which can be used to classify subsequent examples. These algorithms have been tested on problems ranging from soybean disease diagnosis [Michalski80] to classifying chess end games [Quinlan83]. 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 [Hinton86, Rosenblatt62, Rumelhart86]. 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 [Sejnowski87] to evaluating moves in backgammon [Tesauro89].

This article presents the results of several experiments comparing the performance of the ID3 symbolic learning algorithm [Quinlan86] with both the perceptron [Rosenblatt62] and back-propagation [Rumelhart86] neural algorithms. All three systems are tested on four large data sets from previous symbolic and connectionist experiments, and their accuracy on novel examples and learning times are measured. The three algorithms perform about the same in terms of classification correctness on new examples, but back-propagation takes much longer to train. Given its well-known limitations, the performance of the perceptron is surprising.

Besides correctness and learning time, three additional aspects of empirical learning are investigated: the dependence on the amount of training data; the ability to handle imperfect data of various types; and 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 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 back-propagation algorithm is often trained with more complicated output encodings which 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 which neurally-based algorithms can exploit during learning.

The experiments indicate that relative performance of the three learning algorithms is largely independent of the amount of training data and that the two styles of learning perform about the same on imperfect domains. The one exception is that back-propagation better handles domains with missing feature values. The results also suggest that back-propagation may be better able to utilize domain-specific distributed output encodings than are the other two systems, although ID3 is also able to profitably use distributed encodings.

The next section describes the four 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 and some aspects of the versions used. Finally, it describes the representation language used to encode examples.

2.1. Data Sets

Four different data sets are used in the current experiments. Three have been previously used to test different symbolic learning systems and one has been used to test back-propagation.

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 [Michalski80]; however, the exact data is that used in [Reinke84].

The chess data set [Shapiro87] consists of examples of a "king and rook versus king and pawn" end game. There are two categories (*Win* and *Not Win*) and 36 high-level features. There are a total of 591 examples, approximately evenly divided between the categories. Similar chess end game data sets have been previously used to test ID3 [Quinlan83].

The audiology data [Bareiss88] 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 [Bareiss88].

The NETtalk [Sejnowski87] 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 central 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. While inconsistent training data is not a problem for back-propagation, it is a problem for the standard ID3 algorithm. The modification made to ID3 in order to handle noise is described below.

Unfortunately, the full dictionary is too extensive to tractably analyze 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 [Kuchera67]) 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 which are 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 only keeping 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, the ID3, perceptron, and back-propagation algorithms are chosen as representative algorithms. This section briefly describes the systems used and our reason for choosing them as representatives.

2.2.1. ID3

ID3 is chosen 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 [Quinlan83, Wirth88] and is the basis of several commercial rule induction systems. In addition, ID3 has been augmented with techniques for handling noisy data and missing information [Quinlan86]. Finally, in experimental comparisons with other symbolic learning algorithms, ID3 generally performs about as well or better than other systems [O'Rorke82, Rendell89].

ID3 uses 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:

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

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, 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 which determines the splitting attribute acts as a hill-climbing heuristic which tends to minimize the size of the resulting decision tree. The implementation of ID3 used in the experiments is written in Common Lisp, and although some attention is made to efficiency, the code is not carefully optimized.

The standard technique for handling noisy data in ID3 is the *chi-squared* method [Quinlan86]. 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 labelled with the most common category in the partition. However, experiments revealed that this method does not handle noise as well as a simpler method we call *no-chi*. The no-chi method only terminates growth when actual inconsistent training data is encountered, i.e. when all remaining attributes have an information gain of zero but all the examples are not in one category. In this case, a leaf is created and labelled with the most common category present in the examples. Empirical data showing that the no-chi method handles noise better than the chi-squared method is presented in section 3.3.4. Consequently, all experiments presented in this paper use the no-chi version of ID3 unless explicitly stated otherwise.

2.2.2. Perceptron

As is well known, the perceptron learning procedure, one of the first neural network learning algorithms, is incapable of learning concepts that are not linearly separable [Minsky88]. Despite this fact, it performs quite well on several large data sets used to test recent learning systems. Therefore, the perceptron is included 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 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 is linearly separable, this procedure is guaranteed to converge on a correct set of weights after a finite number of input presentations [Minsky88].

The implementation of the perceptron used in the experiments is written in Common Lisp, and although some attention is made to efficiency, the code is not carefully optimized. It 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* [Minsky88] guarantees this will eventually happen if and only if the data is 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 classied 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. Back-Propagation

Over the past several years, a few neural learning algorithms have been developed that are capable of learning concepts that are not linearly separable [Hinton86, Rumelhart86]. Back-propagation (also called the generalized delta rule) [Rumelhart86] is currently the most popular of these procedures and has been tested on several large-scale problems [Sejnowski87, Tesauro89]. Consequently, back-propagation is chosen to represent the new neural learning algorithms.

Like the perceptron, back-propagation 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 which neither directly receive external input nor produce external output). For the method to work, the thresholding function of each unit must be smoothed out 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 "threshold" or bias for unit j. The weights of each unit are changed after the presentation of each pattern p by back-propagating 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

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

where η is a parameter 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 perceptron, back-propagation may get stuck at a local minima and therefore is not guaranteed to converge.

The version of back-propagation used in the experiments is the C code supplied with the third volume of the PDP series [McClelland87]. The learning rate is set to 0.25 and the momentum term is set at 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 number 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

The examples of all four data sets are represented as bit vectors rather than multi-valued feature vectors.[1] 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 set, then all the bits in the set are off, and no special processing for missing features need be done. (The issue of the representation of missing feature values is further discussed in section 3.3.2.)

Binary encodings are a natural representation for neural learning algorithms. The binary encoding was consistently found to slightly improve the classification performance of ID3, and therefore is used for all of the learning algorithms. 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 [Quinlan86]. 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* [Cheng88] 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.

---

[1] The number of input and output bits for each domain is as follows: soybeans - 208 and 17; chess - 73 and 2; audiology - 86 and 24; NETtalk - 189 and 115.

Since bit vectors are usable by all three systems, their use also helps standardize the experimental setup. In their related experiments, Fisher and McKusick [Fisher89a] did not use binary input encodings for ID3. The effect this may have had is discussed when the relevant results are reported.

3. EXPERIMENTS AND RESULTS

Four experiments are reported in this section. The first experiment compares learning time and correctness 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, the value of distributed output encodings is investigated.

To perform the experiments, each data set is separated 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. For sets other than NETtalk-full, 2/3*rds* 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 back-propagation uses a different seed random number when determining the initial random network weights.

For NETtalk-full, only a single training set is used. However, for all runs where back-propagation processes the NETtalk-full data, back-propagation is run 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 back-propagation. In tests with the other domains, 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, five randomly chosen initial states are used 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 correctness 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 test set.

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 permutations of the data sets. The actual numbers, their standard deviations, and several other statistics appear in the appendix. The means of the training times for soybeans, chess, audiology and NETtalk-A are 160 seconds (ID3), 373 seconds (perceptron), and 21,000 seconds (back-propagation). For correctness, the means are 81.1% (ID3), 79.4% (perceptron), and 83.6% (back-propagation). The back-propagation 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.

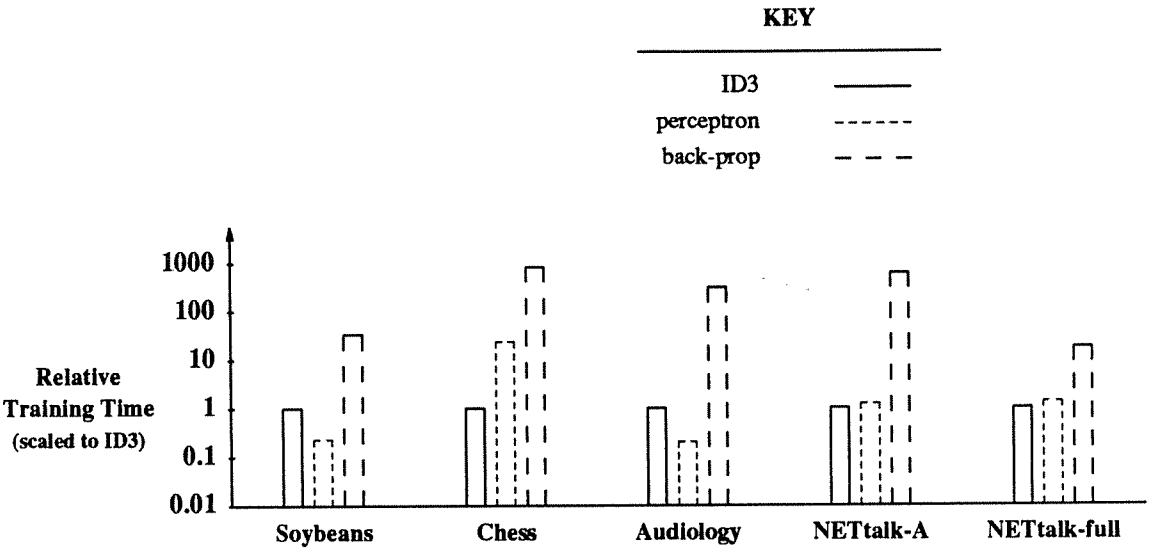

Figure 1. Relative Training Times of the Three Algorithms

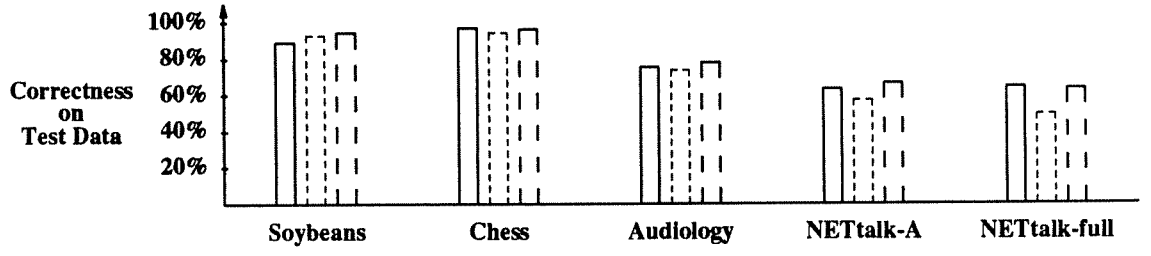

Figure 2. Correctnesses of the Three Algorithms

A relevant statistical test for these problem results is a two-way analysis of variance (a standard technique of statistics). This test is designed to determine the source of variation in the observed correctness over each of the four domains. The two way analysis of variance returns two values: the likelihood that the variation can be explained by the different training sets, and the likelihood that the variation can be explained by differences between training methods.

For NETtalk-A and soybeans (where back-propagation performs best) and chess (where ID3 performs best), it is possible to conclude at the 0.5% level (i.e., with 99.5% confidence) that the variation in observed correctness results from the difference in training methods rather than random error. Results are less conclusive for audiology (where back-propagation out-performs the others). There it is only possible to conclude with 95% confidence that the training method is the source of the variation.

3.1.2. Discussion

In this first set of experiments, all three learning systems are remarkably similar with respect to classifying novel examples. However, ID3 and perceptron train much faster than back-propagation. Despite the obvious differences between decision trees and neural networks, their ability to accurately represent concepts and correctly classify novel instances appears to be quite comparable. Data from other recent experiments supports this general conclusion [Fisher89a, Weiss89].

One possible explanation for the similarity in performance is that most reasonable generalization procedures which 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 [Blumer87] which states, roughly, that any polynomial algorithm that compresses the training data is a polynomial learner in the sense of Valiant [Valiant84]. 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 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 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 and/or slowly adding hidden units as they are needed [Hanson89, Honavar88]. Such systems help ease the burden of having to initially specify an appropriate network for a problem.

A final point of discussion concerns the performance of back-propagation on the NETtalk-full data set. On this data set, back-propagation learns the training set less well than for the other four data sets. In general, throughout all four experiments it was found that the better a system learned the training set, the better it did on the test set (this issue is discussed further in section 3.3.4). Hence, it is possible that with a different number of hidden units or if allowed more than 100 training epochs, back-propagation would have done better on the NETtalk-full test set. Between epochs 75 and 100, the training set performance of back-propagation never improved more than one percentage point. To reach 99% correctness on the training set would probably require several orders of magnitude more training time, which is infeasible.

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

It is possible that some learning methods perform relatively better with sparse training data while others perform relatively better on large training sets. To address this issue, curves that plot correctness as a function of training set size are produced in this experiment. ''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.

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) 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. 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. See [Fisher89b] for a discussion of several ways to apply back-propagation in an incremental setting.

3.2.2. Discussion

The results indicate that for small amounts of training data back-propagation should be used. With small training sets, the slower speed of back-propagation is much less of a problem and on the four sample domains back-propagation did about the same as or much better than

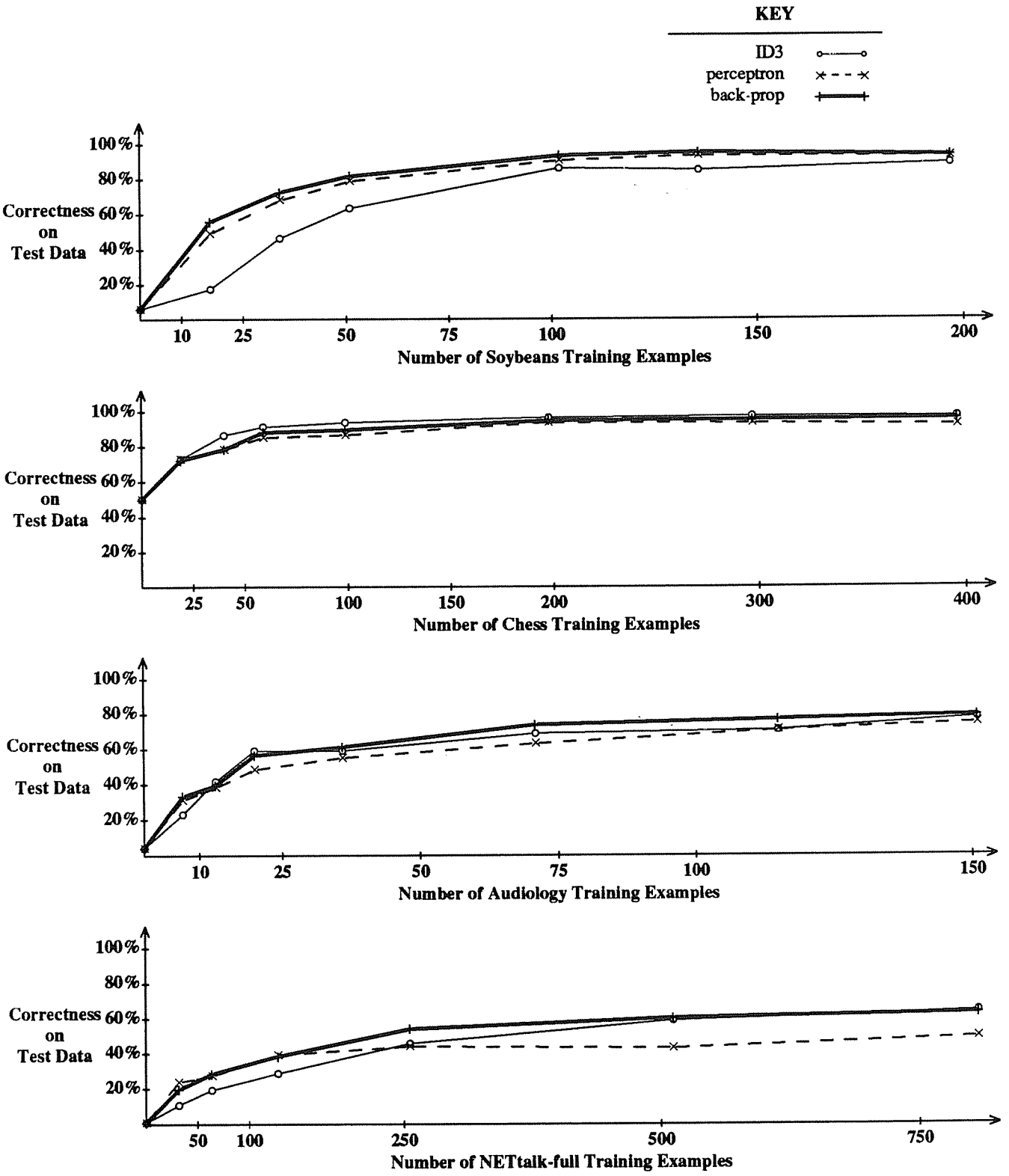

Figure 3. Classification Performance as a Function of Amount of Training Data

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 back-propagation takes so long to run, this technique can have value in practical systems.

Other studies [Fisher89a] suggest that ID3 may be more effective than back-propagation on relatively small training sets, but as the size of the training set increases, back-propagation eventually outperforms ID3. This difference results from two procedural differences. One, when generating learning curves, Fisher and McKusick use back-propagation in an incremental fashion and update the weights only once for each new training example. In the learning curves presented in figure 3, back-propagation is allowed to converge on all of the training examples so far encountered before running it on the test set. Running back-propagation to convergence leads to much better performance on small amounts of training data. Two, Fisher and McKusick do not use binary input encodings for ID3, consequently ID3 and back-propagation are given different encodings of the inputs. We found that using the binary encoding for ID3 can improve correctness by a few percentage points and may account for some of the difference observed by Fisher and McKusick.

3.3. Experiment Three: Effect of Imperfect Data Sets

An important aspect of inductive learning systems is their sensitivity to imperfections in the data sets. Improperly represented examples can occur for several reasons. Mistakes may be made when recording feature values or when judging in which class an example belongs. 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. Following [Quinlan86], 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 also consistent with work on the theoretical basis of learning [Valiant84], in which it is assumed that the distribution of examples is the same during training and test.

Due to processing limitations, in these experiments a reduced version of the NETtalk-full training set is used. Rather than using the 808 word training set, only the first 256 words (1099 examples) are used to train. This data set is called NETtalk256. Except for this data set, all the

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

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. 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.

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, in the test set, 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.

Discussion

ID3 and back-propagation appear to handle random noise about equally well. In one domain (audiology), ID3 significantly out-performed the other two systems, while in another (soybeans) back-propagation was the best. This disagrees with results on the comparative effect of noise reported in [Fisher89a]. Fisher and McKusick report that back-propagation *consistently* out-performs ID3, while the results reported here indicate that ID3 frequently handles noise as well as or better than back-propagation. This difference may be due to the ways examples for ID3 are represented in the two approaches. As previously discussed, we found that converting to a binary representation improved ID3's performance. Also, Fisher and McKusick use the *chi-squared* technique suggested by Quinlan for handling noise. This technique was found to be inferior to closely fitting the training data (see section 3.3.4).

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

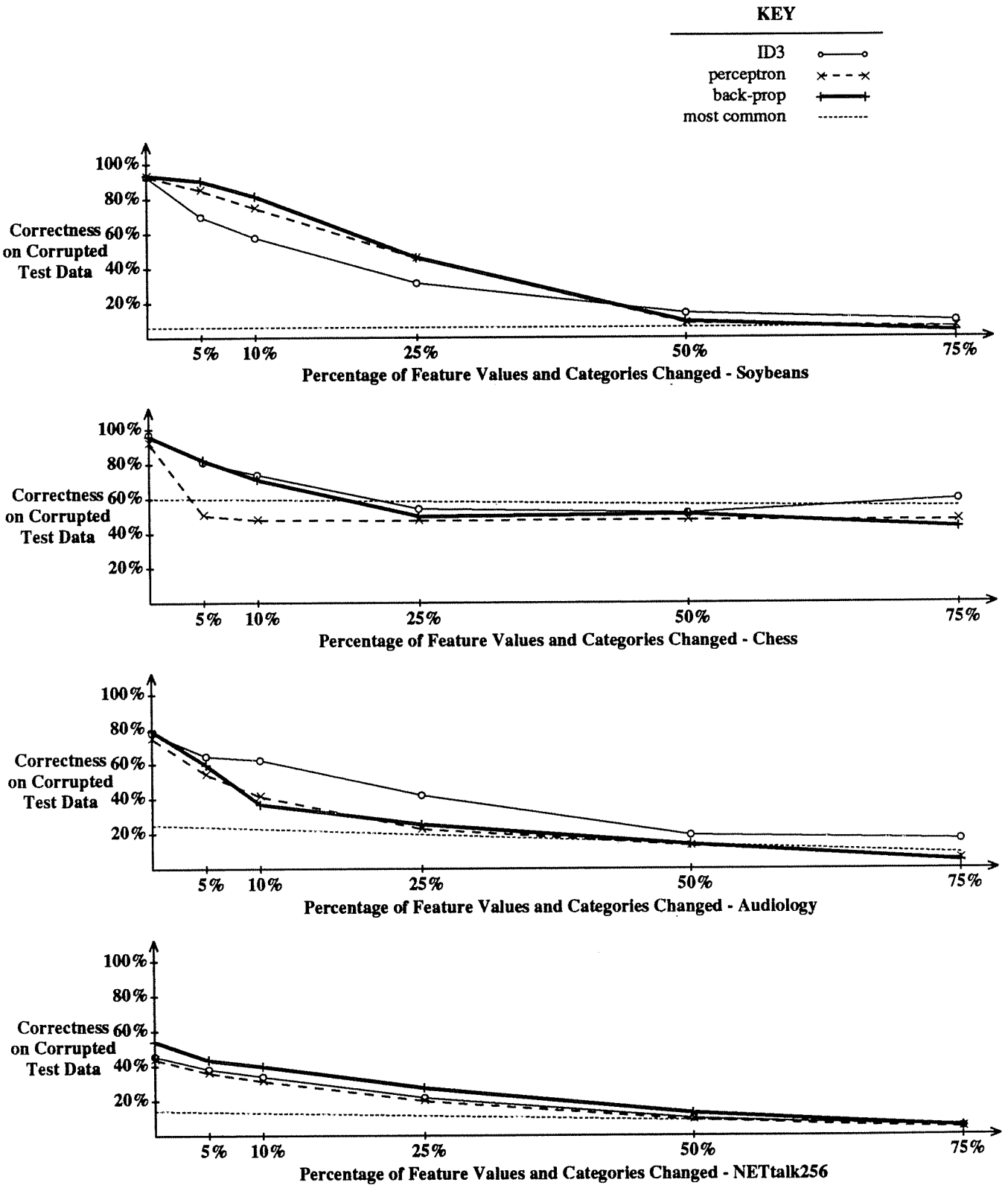

Figure 4. Classification Performance as a Function of Amount of Feature and Category Noise

probability p a given feature value is deleted.

Representation

Learning from incomplete data requires an effective approach to representing and handling missing values. Quinlan has dealt with this issue in ID3 [Quinlan86] and argues that the best approach is to adjust the calculation of information gain for feature A by assigning missing values in proportion to the relative frequency of possible values in the current collection of examples. Duran [Duran88] further investigated this issue. She found the best approach is to replace, before the ID3 algorithm is run, missing values with the most probable value of the feature, given the class of the example. A third approach, given the binary representation used in these experiments, is to set to zero all the bits for features with unknown value.

These three approaches were applied to the soybeans, chess, and audiology data sets. Consistent with Duran's results, the *most probable* approach performs best. This approach is used for the ID3 runs involving missing feature values.

To the best of our knowledge, the issue of representing missing values in neural networks has not been investigated. Three representations of missing features for back-propagation are evaluated. Assume that there are N possible values for some feature. Hence, N inputs are used 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" the single one used to indicate the feature value across all N inputs. (ID3's *most probable* approach is not evaluated, since its use precludes using back-propagation incrementally.)

Performance of these three approaches, averaged over on 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, it is used for back-propagation in the remainder of this experiment.

The three approaches tested with back-propagation are also tested with perceptron. Using all zeroes works best, and this is used for perceptron in the experiment described below. The other two approaches are less successful because they more quickly lead to cycles, indicating the presence of linearly-unseparable concepts and terminating perceptron learning.

(Earlier it is 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 scientist producing the example decided that the feature was *irrelevant*,

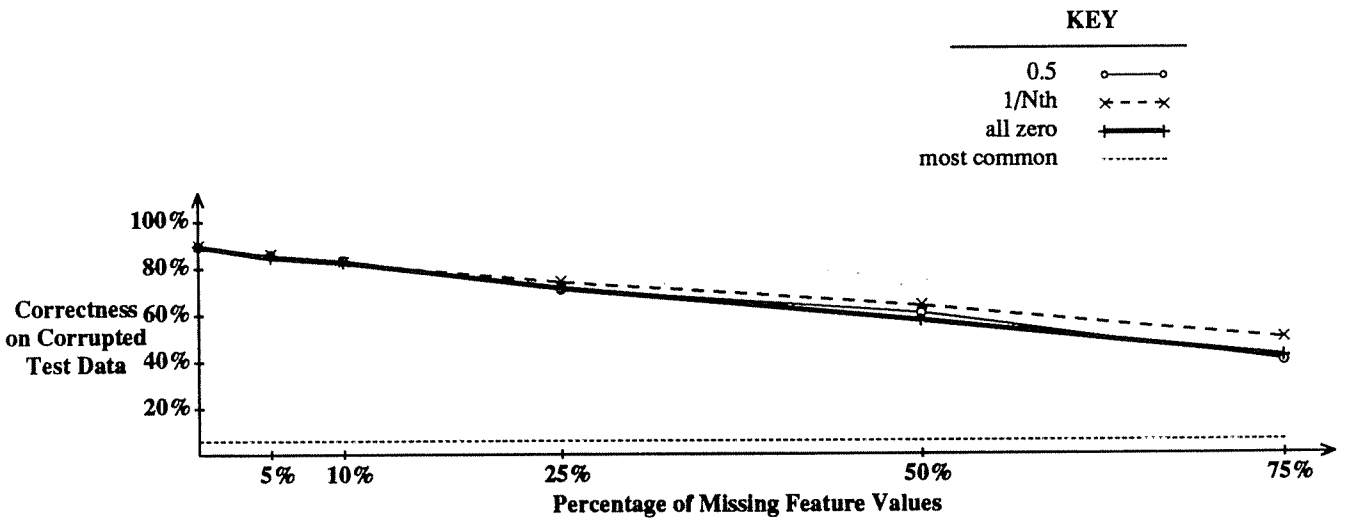

Figure 5. Effect of Representation of Missing Features on Back-Propagation Performance

given the other details of the case [Duran88]. Experiments indicated that it is best to represent this type of absent feature value by all zeroes. To produce the curves below, only the *randomly* discarded feature values are represented using the techniques described above for ID3 and back-propagation.)

Results

The performance of the three learning algorithms as a function of the percentage of missing features values is reported in figure 6. (These plots are often called "ignorance curves.") ID3 and perceptron perform poorly on some of the data sets, while back-propagation performs well across all four domains.

Discussion

The results indicate that back-propagation handles missing feature values better than ID3 and perceptron do. Perceptron does poorly because complicated representations of missing values lead to concepts that are not linearly separable. The reason for back-propagation out-performing ID3 may be that back-propagation naturally supports the representation of partial evidence for a feature's value. Back-propagation makes decisions by summing weighted evidence across many features, while ID3 locally bases its decisions on the value of a single feature. Missing feature values are more naturally matched by the biases inherent in back-propagation.

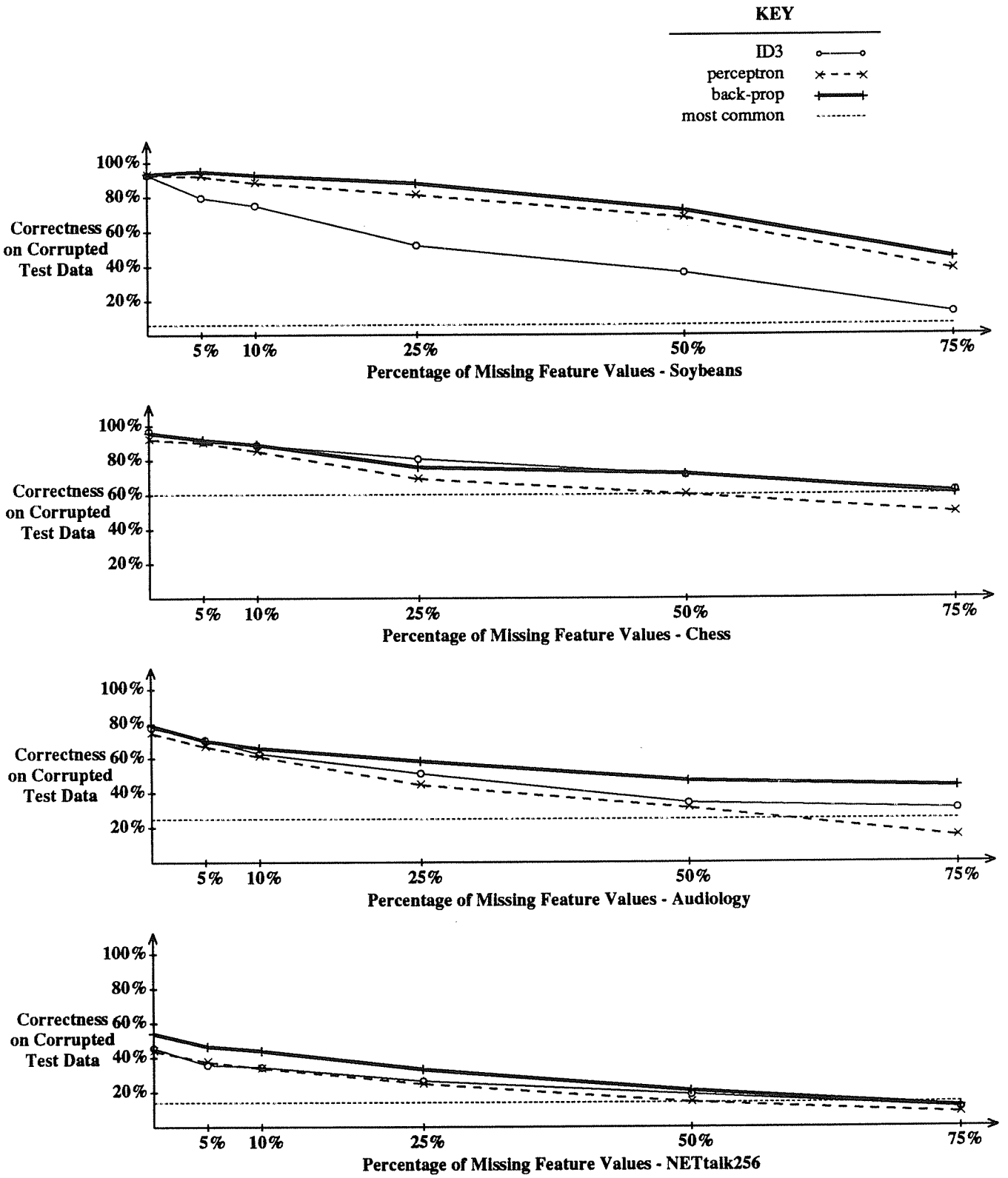

Figure 6. Classification Performance as a Function of Amount of Missing Feature Values

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. Sensitivity of the learning algorithms to the number of features is investigated 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, three different training sets are used and a different random collection of dropped features is chosen 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 back-propagation 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 back-propagation do well as the number of features is reduced. 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. On the NETtalk256 data set, ID3 performs best when it only considers a three-letter window.

3.3.4. ID3 and the Chi-Squared Test

Quinlan suggests a statistical technique called the *chi-squared* test be used to handle noise [Quinlan86]. This statistic measures the relevancy of a feature to category membership. Irrelevant features are not considered for partitioning. Earlier it is stated that this statistic does not fare as well as simply discarding conflicting training examples. The data that supports this conclusion is summarized in this section.

Results

Figure 8 presents the test set correctness results, averaged over all four domains, for the three different types of data imperfections. For each type of imperfection, using the chi-squared

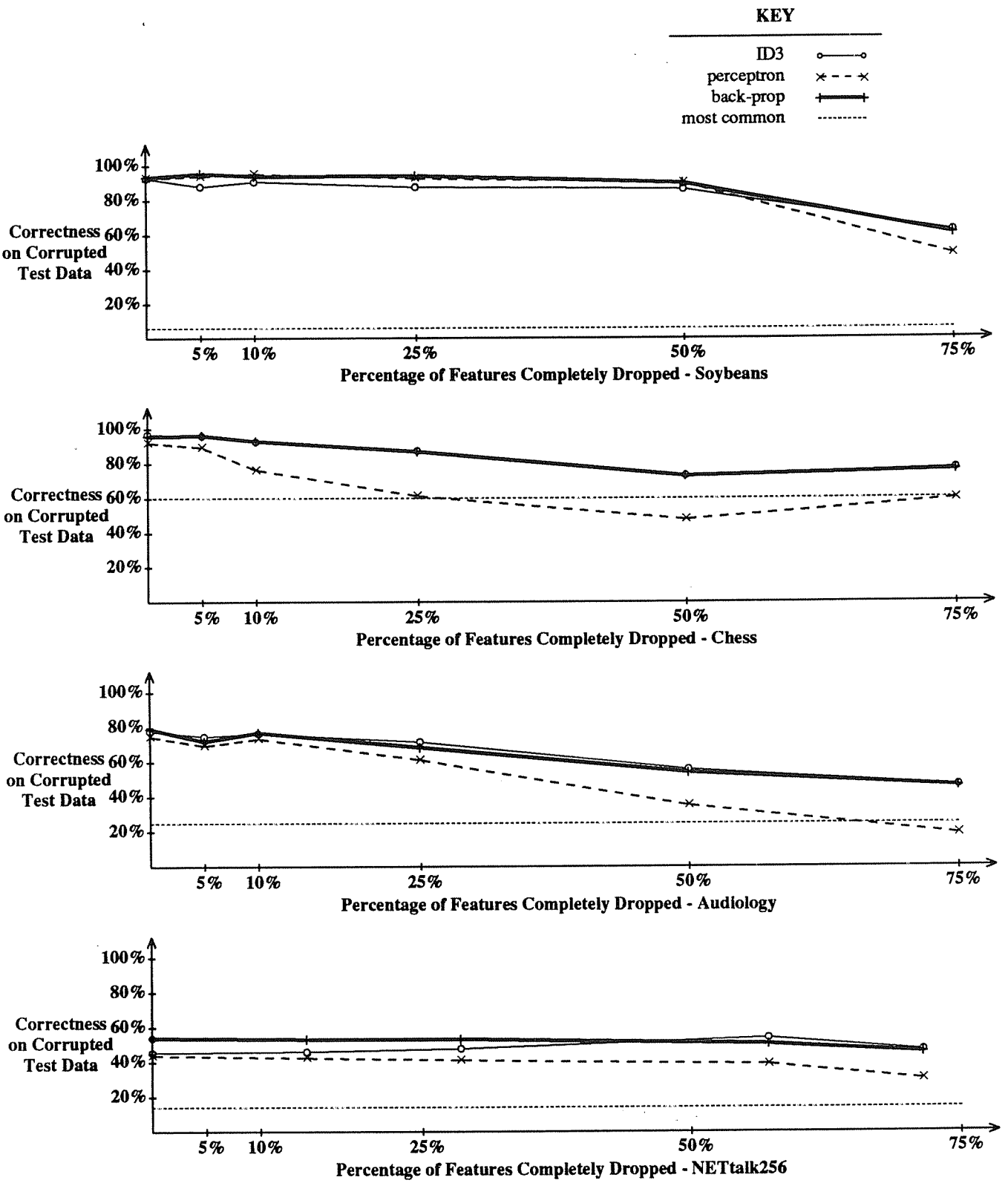

Figure 7. Classification Performance as a Function of Number of Features Totally Dropped

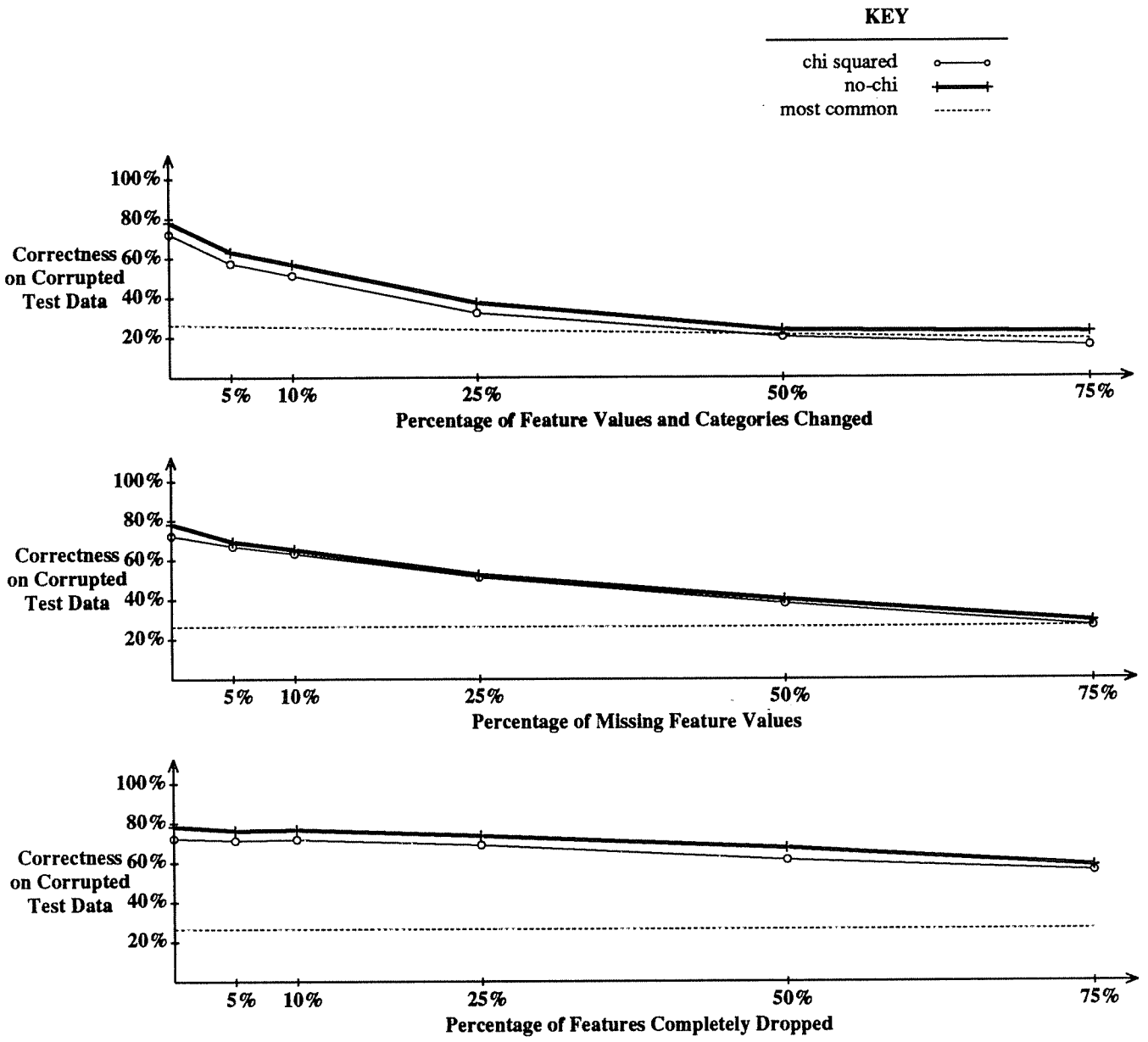

Figure 8. Performance of ID3 for Two Different Noise-Handling Techniques

statistic does worse. On the uncorrupted NETtalk-full data set, which as previously described contains conflicting training examples, the chi-squared version produces a test set correctness of 51.5% while the no-chi version achieves 64.2%. (Except when half or more of the features are totally dropped, the no-chi version of ID3 correctly classifies over 98% of the training examples. The chi-squared version does roughly 20% better classifying the training set than the test set.)

Using the chi-squared statistic requires setting a confidence level for use in making decisions about the irrelevancy of features. Several settings were tested and figure 8 reflects the

results with a 99.5% confidence, which performs best.

Discussion

A number of experiments with symbolic inductive learning systems have observed the phenomenon of *overfitting*, i.e. increasing performance on training data resulting in decreased performance on test data [Clark89, Kononenko84, Michalski86]. Our experiments provided no evidence for overfitting. Every change that increased performance on the training data increased performance on test data as well. The no-chi method for handling noise in ID3 fits the training data more closely than the chi-squared method; however,its accuracy on test data is higher for both random and systematic noise (i.e. the noise present in the original NETtalk data). This is an indication that the chi-squared method has a tendency to underfit the training data and treat worthwhile information as noise. Our data suggests it is better to fit the training data as closely as possible and risk overfitting than to use methods that may underfit.

Perhaps other methods for preventing overfitting, such as *post-pruning* methods which simplify decision trees or rules after they are learned [Kononenko84, Michalski86], are better at avoiding the complementary problem of underfitting. Also, most proponents of pruning emphasize the simplification of concept definitions with little or no loss in accuracy (and possibly a gain). Since neural methods do not form explicit rules, we have not considered the issue of rule complexity.

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, in [Sejnowski87], the sound/stress outputs in NETtalk-full are encoded for back-propagation 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 the final experiment, Sejnowski's methodology is repeated 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 feature-oriented output vector requires more information about the data sets.

3.4.1. Methodology

The back-propagation algorithm easily handles this distributed encoding of the output. For the back-propagation tests involving distributed outputs, 120 hidden units and 30 training epochs are used, following [Sejnowski87]. While back-propagation 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 back-propagation network or by passes through each of the 26 decision trees or perceptrons. Back-propagation produces a vector containing numbers between 0 and 1. ID3 produces a binary vector. Perceptron produces a vector of integers, where each entry is a perceptron's output minus its threshold. The specified category is then determined by finding the phoneme/stress pair encoding which has the smallest angle with the 26 outputs. (This is the "best guess" metric used in [Sejnowski87].)

3.4.2. Results

Table 1 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.

3.4.3. Discussion

The use of a distributed encoding of the output substantially improves the performance of back-propagation. For the distributed encoding, back-propagation's best correctness is 96% on the training set and 72% on the testing set. Using the local encoding - one output bit for each

Table 1
Classification Correctnesses on NETtalk Data Set for Two Output Encoding Methods

	Local Encoding	Distributed Encoding
Back-propagation	63.0%	72.3%
ID3	64.2%	69.3%
Perceptron	49.2%	42.1%

category - results in a best correctness of 88% percent on the training set and 63% on the testing set. ID3 is also able to take advantage of the distributed encoding, but not by as much as back-propagation.

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, however, a cost of using the distributed encoding with back-propagation. The range of correctness varies widely when the training data is imperfect. Using the NETtalk256 data corrupted with 5% noise, back-propagation's correctness ranges from 1% to 31% when only the seed random number is varied. The problem worsened as the noise level increased. On all other data sets, and on NETtalk using the local output encoding, there is a very small range 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.

4. GENERAL DISCUSSION

This study's experiments investigate the relative performance of symbolic and neural approaches to learning. The first experiment shows that back-propagation performs about the same as the other two algorithms in terms of classification correctness on new examples, but takes much longer to train. The result of the second experiment is that the relative performance of the three learning algorithms is independent of the amount of training data. The third experiment, which investigates the handling of imperfect data sets, indicates that back-propagation handles missing feature values better than ID3 and perceptron do. ID3 and back-propagation perform about the same in the presence of random noise and with reduced numbers of features. In the fourth experiment, back-propagation is better able to utilize a distributed output encoding, although ID3 is also able to take advantage of this representation style. Several additional aspects of the experiments are discussed 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 [Smolensky88]. The experiments reported in the previous section demonstrate that this distinction can be misleading ([Fodor88, Langley89] also discuss this issue). ''Symbolic'' systems like ID3 and ''subsymbolic'' systems like back-propagation 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 like back-propagation seem to be able to take somewhat 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) [Muroga79] 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 iff at least X of the N inputs are 1). Not surprisingly, experiments with artificial data show that back-propagation performs better than ID3 on problems which fit its natural bias, like *X of N* functions [Fisher89a]. 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

One surprising result of these experiments is how well the simple perceptron algorithm performs. The perceptron was largely abandoned as a general learning mechanism about twenty years ago because of its inherent limitations, such as its inability to learn concepts that are not linearly-separable [Minsky88]. Nevertheless, it performs quite well in these experiments. Except on NETtalk 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 possible 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 [Quinlan83]. The actual explanation is probably a combination of these two important factors.

Regardless of the reason, data for many "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* [Minsky88] 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 back-propagation can be tried. The *perceptron tree error correction procedure* [Utgoff88b] 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 Back-Propagation

Although back-propagation 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 four data sets, in the first experiment back-propagation takes about 500 times as long to train as ID3. (Testing in back-propagation takes about 10 times longer than the other two systems.) These factors would probably increase if one optimized the ID3 and perceptron code, which are not coded efficiently compared to the C version of back-propagation.

One obvious way back-propagation can be made faster is to exploit its intrinsic parallelism. The networks used in these experiments contained an average of about 175 units. Consequently, assuming one processor per unit and perfect speedup, the training time for back-propagation 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 learning time of each learning algorithm as the size of the problem increases is an important theoretical and practical issue. Learning time for ID3 grows linearly with the number of examples and, in the worst case, quadratically with the number of features [Schlimmer86, Utgoff88a]. However, 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. Recently there have been speculations that neural networks scale badly [Minsky88]. Recent theoretical work indicates that back-propagation learning may be NP-complete [Blum88]. There is empirical evidence that on average back-propagation's time to reach asymptotic performance grows quadratically with the number of features [Hinton89]. However, as evidenced on the NETtalk-full data, back-propagation can still produce impressive results even when training terminates before the training set is approximately fully learned.

4.5. Incremental Learning

One issue not investigated is incremental learning without requiring the complete storage of all previous examples. Incremental versions of ID3 have been proposed [Schlimmer86, Utgoff88a] and back-propagation 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 [Fisher89b].

4.6. Other Issues

There are several other differences between the three learning algorithms and between symbolic and connectionist approaches in general. For one, given a collection of input/output training pairs, ID3 and perceptron can be directly run. On the other hand, in order to run back-propagation, a network architecture must be chosen, currently much more of an art than a science. Not only must the number of hidden units be chosen, but the number of hidden layers must also be specified. In addition, the learning rate and the momentum term must be chosen. 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, back-propagation may fail to converge efficiently. However, it should also be mentioned that many symbolic learning systems have parameters which must be appropriately

set to insure good performance [Rendell89].

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 [Shapiro87]. 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 [McClelland86]. 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. 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 back-propagation) are compared on four real-world data sets. These data sets have been used in previous experiments. Three in symbolic learning research (soybeans, chess, and audiology) 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 back-propagation, both during learning and during classification of novel examples. In all of the sample domains (except for perceptron on the NETtalk data), the probability of correctly classifying new examples is about the same for the three systems.
- Additional experiments, in which the sample data sets are carefully corrupted, suggest that with noisy data sets the two learning approaches are approximately equally effective. The standard ID3 technique for handling noisy data is shown to be inferior to a simpler algorithm in which conflicting training examples are discarded. The commonly-held belief that "overfitting" the training data is a serious problem is contradicted by the experiments. The better a learning algorithm did on the training data, the better it did on the testing data.
- When examples are incompletely specified, back-propagation performs better than the other two approaches. A technique for representing missing feature values in neural networks is proposed and shown effective.

- One claimed advantage of neural approaches to learning is that they are able to profitably use distributed encodings. Empirical studies of the NETtalk data set suggest that back-propagation is able to take advantage of domain-specific distributed output encodings better than the two other systems, although ID3 is also able to utilize distributed output encodings.

This study provides the beginnings of a better understanding of the relative strengths and weaknesses of the symbolic and neural approaches to machine learning.

Acknowledgements

The authors would 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; Rob Holte and Peter Clark for Alen Shapiro's chess data; and Terry Sejnowski for the NETtalk data. Tom Dietterich suggested the noise-handling technique used in ID3. 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 [Litzkow88], 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 and the University of Texas at Austin's Department of Computer Sciences and Artificial Intelligence Laboratory.

Appendix

Table 2 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.

Table 2. Results of Experiment 1 (Means and Standard Deviations)

Domain / System	Training		Testing	Correctness (%)	
	Time (sec)	Epochs	Time (sec)	Training Set	Test Set
Soybeans					
ID3	155.0 ± 7.8		0.1 ± 0.0	100.0 ± 0.0	89.0 ± 2.0
perceptron	35.8 ± 5.2	11.9 ± 0.7	0.8 ± 0.1	100.0 ± 0.0	92.9 ± 2.1
back-prop	5,260.0 ± 7,390.0	157.8 ± 175.3	7.9 ± 0.3	99.9 ± 0.2	94.1 ± 2.5
Chess					
ID3	40.4 ± 1.9		0.1 ± 0.0	100.0 ± 0.0	97.0 ± 1.6
perceptron	970.0 ± 554.0	3,460.0 ± 1,910.0	0.1 ± 0.0	97.6 ± 2.4	93.9 ± 2.2
back-prop	34,700.0 ± 15,000.0	4,120.0 ± 1,770.0	1.8 ± 0.0	99.3 ± 0.4	96.3 ± 1.0
Audiology					
ID3	61.8 ± 1.8		0.1 ± 0.0	100.0 ± 0.0	75.5 ± 4.4
perceptron	12.5 ± 0.5	9.6 ± 0.5	0.3 ± 0.0	100.0 ± 0.0	73.5 ± 3.9
back-prop	19,000.0 ± 13,100.0	2,880.0 ± 1,980.0	1.8 ± 0.1	99.8 ± 0.3	77.7 ± 3.8
NETtalk-A					
ID3	382.0 ± 15.4		0.1 ± 0.0	98.3 ± 0.4	63.1 ± 3.0
perceptron	472.0 ± 30.4	108.0 ± 28.4	1.2 ± 0.0	88.9 ± 1.8	57.2 ± 2.0
back-prop	234,000.0 ± 27,600.0	5,000.0 ± 0.0	10.1 ± 1.5	96.7 ± 0.9	66.4 ± 2.4
NETtalk-Full					
ID3	9,220		712	98.5	64.2
perceptron	12,300	38.4	5,640	67.7	49.2
back-prop	168,000	100.0	24,300	88.5	63.0

References

[Bareiss88] E. R. Bareiss, "PROTOS: A Unified Approach to Concept Representation, Classification and Learning," Ph.D. Thesis, Department of Computer Science, University of Texas, Austin, TX, 1988. (Available as Technical Report AI88-83.)

[Blum88] A. Blum and R. L. Rivest, "Training a 3-Node Neural Network is NP-Complete," *Proceedings of the 1988 Workshop on Computational Learning Theory*, Cambridge, MA, August 1988, pp. 9-18.

[Blumer87] A. Blumer, A. Ehrenfeucht, D. Haussler and M. K. Warmuth, "Occam's Razor," *Information Processing Letters 24*, (1987), pp. 377-380.

[Cheng88] J. Cheng, U. M. Fayyad, K. B. Irani and Z. Qian, "Improved Decision Trees: A Generalized Version of ID3," *Proceedings of the Fifth International Conference on Machine Learning*, Ann Arbor, MI, June 1988, pp. 100-106.

[Clark89] P. Clark and T. Niblett, "The CN2 Induction Algorithm," *Machine Learning 3*, 4 (1989), pp. 261-284.

[Duran88] R. T. Duran, "Concept Learning with Incomplete Data Sets," Technical Report AI88-82, Department of Computer Science, University of Texas, Austin, TX, August 1988.

[Fisher89a] D. H. Fisher and K. B. McKusick, "An Empirical Comparison of ID3 and Back-propagation," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989.

[Fisher89b] D. Fisher, K. McKusick, R. J. Mooney, J. W. Shavlik and G. G. Towell, "Processing Issues in Comparisons of Symbolic and Connectionist Learning Systems," *Proceedings of the Machine Learning Workshop*, Ithaca, NY, July 1989.

[Fodor88] J. A. Fodor and Z. W. Pylyshyn, "Connectionism and Cognitive Architecture: A Critical Analysis," in *Connections and Symbols*, S. Pinker and J. Mehler (ed.), MIT Press, Cambridge, MA, 1988, pp. 3-71.

[Hanson89] S. J. Hanson and L. Y. Pratt, "Comparing Biases for Minimal Network Construction with Back-Propagation," in *Advances in Neural Information Processing Systems, Volume 1*, D. S. Touretsky (ed.), Morgan-Kaufmann, Los Altos, CA, 1989, pp. 177-185.

[Hinton86] G. E. Hinton and T. J. Sejnowski, "Learning and Relearning in Boltzmann Machines," in *Parallel Distributed Processing, Vol. I*, D. E. Rumelhart and J. L. McClelland (ed.), MIT Press, Cambridge, MA, 1986, pp. 282-317.

[Hinton89] G. Hinton, "Connectionist Learning Procedures," *Artificial Intelligence*, (in press) .

[Honavar88] V. Honavar and L. Uhr, "A Network of Neuron-Like Units that Learns to Perceive by Generation as Well as Reweighting of its Links," in *Proceedings of the 1988 Connectionist Models Summer School*, G. E. Hinton, T. J. Sejnowski and D. S. Touretzky (ed.), Morgan Kaufmann, San Mateo, CA, 1988.

[Kononenko84] I. Kononenko, I. Bratko and E. Roskar, "Experiments in Automatic Learning of Medical Diagnostic Rules," Technical Report, Jozef Stefan Institute, Ljubljana, Yugoslavia, 1984.

[Kuchera67] H. Kuchera and W. N. Francis, *Computational Analysis of Modern-Day American English*, Brown University Press, Providence, RI, 1967.

[Langley89] P. Langley, "Editorial: Toward a Unified Science of Machine Learning," *Machine Learning 3*, 4 (1989), pp. 253-259.

[Litzkow88] M. Litzkow, M. Livny and M. W. Mutka, "Condor - A Hunter of Idle Workstations," *Eighth International Conference on Distributed Computing Systems*, June 1988.

[McClelland86] J. L. McClelland, "Resource Requirments of Standar and Programmable nets," in *Parallel Distributed Processing, Vol. I*, D. E. Rumelhart and J. L. McClelland (ed.), MIT Press, Cambridge, MA, 1986, pp. 460-487.

[McClelland87] J. L. McClelland and D. E. Rumelhart, *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*, MIT Press, Cambridge, MA, 1987.

[Michalski80] R. S. Michalski and R. L. Chilausky, "Learning by Being Told and Learning from Examples: an Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis," *Policy Analysis and Information Systems 4*, 2 (June 1980), pp. 125-160.

[Michalski83] R. S. Michalski, "A Theory and Methodology of Inductive Learning," *Artificial Intelligence 20*, 2 (1983), pp. 111-161.

[Michalski86] R. S. Michalski, I. Mozetic, J. Hong and N. Lavrac, "The Multi-Purpose Incremental Learning System AQ15 and its Testing Application in Three Medical Domains," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 1041-1047.

[Minsky88] M. L. Minsky and S. Papert, *Perceptrons: Expanded Edition*, MIT Press, Cambridge, MA, 1988. (Original edition published in 1969.)

[Mitchell82] T. M. Mitchell, "Generalization as Search," *Artificial Intelligence 18*, 2 (1982), pp. 203-226.

[Mooney89] R. J. Mooney, J. W. Shavlik, G. G. Towell and A. Gove, "An Experimental Comparison of Symbolic and Connectionist Learning Algorithms," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989.

[Muroga79] S. Muroga, *Logic Design and Switching Theory*, John Wiley & Sons, New York, N.Y., 1979.

[O'Rorke82] P. O'Rorke, "A Comparative Study of Inductive Learning Systems AQ11 and ID-3 Using a Chess Endgame Test Problem," Technical Report: UIUCDCS-F-82-899, Department of Computer Science,

University of Illinois, Urbana, IL, September 1982.

[Quinlan83] J. R. Quinlan, "Learning Efficient Classification Procedures and their Application to Chess End Games," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, T. M. Mitchell (ed.), Tioga Publishing Company, Palo Alto, CA, 1983.

[Quinlan86] J. R. Quinlan, "Induction of Decision Trees," *Machine Learning 1*, 1 (1986), pp. 81-106.

[Reinke84] R. Reinke, *Knowledge Acquisition and Refinement Tools for the ADVISE Meta-Expert System*, M.S. Thesis, University of Illinois at Urbana-Champaign, 1984.

[Rendell89] L. A. Rendell, H. H. Cho and R. Seshu, "Improving the Design of Similarity-Based Rule-Learning Systems," *International Journal of Expert Systems 2*, 1 (1989), pp. 97-133.

[Rosenblatt62] F. Rosenblatt, *Principles of Neurodynamics*, Spartan, New York, 1962.

[Rumelhart86] D. E. Rumelhart, G. E. Hinton and J. R. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing, Vol. I*, D. E. Rumelhart and J. L. McClelland (ed.), MIT Press, Cambridge, MA, 1986, pp. 318-362.

[Schlimmer86] J. C. Schlimmer and D. Fisher, "A Case Study of Incremental Concept Induction," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 496-501.

[Sejnowski87] T. J. Sejnowski and C. R. Rosenberg, "Parallel Networks that Learn to Pronounce English Text," *Complex Systems 1*, (1987), pp. 145-168.

[Shapiro87] A. Shapiro, *Structured Induction in Expert Systems*, Addison Wesley, Reading, MA, 1987.

[Smolensky88] P. Smolensky, "On the Proper Treatment of Connectionism," *Behavioral and Brain Sciences 11*, (1988), pp. 1-23.

[Tesauro89] G. Tesauro and T. J. Sejnowski, "A Parallel Network that Leans to Play Backgammon," *Artificial Intelligence 39*, 3 (1989), .

[Utgoff88a] P. E. Utgoff, "ID5: An Incremental ID3," *Proceedings of the Fifth International Conference on Machine Learning*, Ann Arbor, MI, June 1988, pp. 107-120.

[Utgoff88b] P. E. Utgoff, "Perceptron Trees: A Case Study in Hybrid Concept Representations," *Proceedings of the National Conference on Artificial Intelligence*, St. Paul, MN, August 1988, pp. 601-606.

[Valiant84] L. G. Valiant, "A Theory of the Learnable," *Communications of the Association for Computing Machinery* 27, 11 (November 1984), pp. 1134-1142.

[Weiss89] S. M. Weiss and I. Kapouleas, "An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989.

[Wirth88] J. Wirth and J. Catlett, "Experiments on the Costs and Benefits of Windowing in ID3," *Proceedings of the Fifth International Conference on Machine Learning*, Ann Arbor, MI, June 1988, pp. 87-99.