Small Journal Name, ??, 1–40 (??)
© ?? Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

A Comparison of ID3 and Backpropagation for English Text-to-Speech Mapping

THOMAS G. DIETTERICH tgd@cs.orst.edu
HERMANN HILD hhild@i13d1.ira.uka.de
GHULUM BAKIRI
Department of Computer Science, 303 Dearborn Hall, Oregon State University, Corvallis, OR 97331-3202

Editor: Dennis Kibler

Abstract. The performance of the error backpropagation (BP) and ID3 learning algorithms was compared on the task of mapping English text to phonemes and stresses. Under the distributed output code developed by Sejnowski and Rosenberg, it is shown that BP consistently out-performs ID3 on this task by several percentage points. Three hypotheses explaining this difference were explored: (a) ID3 is overfitting the training data, (b) BP is able to share hidden units across several output units and hence can learn the output units better, and (c) BP captures statistical information that ID3 does not. We conclude that only hypothesis (c) is correct. By augmenting ID3 with a simple statistical learning procedure, the performance of BP can be closely matched. More complex statistical procedures can improve the performance of both BP and ID3 substantially in this domain.

Keywords: ID3, backpropagation, experimental comparisons, text-to-speech

1. Introduction

There is no universal learning algorithm that can take a sample $S = \{\langle \mathbf{x_i}, f(\mathbf{x_i})\rangle\}$ of training examples for an arbitrary unknown function f and produce a good approximation to f (see Dietterich, 1989). Instead, every learning algorithm embodies some assumptions (or "bias") about the nature of the learning problems to which it will be applied. Some algorithms, for example, assume that only a small number of the features describing the data are relevant. Other algorithms assume that every feature makes a small, but independent, contribution to determining the classification. Many algorithms order the hypotheses according to syntactic simplicity in some representation and attempt to find the simplest hypothesis consistent with the training examples.

Unfortunately, for many popular learning algorithms, the assumptions they embody are not entirely known—or, if they are known, they are stated in terms that are difficult to check in any given application domain. For example, Quinlan's (1986) decision-tree algorithm ID3 assumes that the unknown function f can be represented as a small decision tree. However, given a new learning problem, it is difficult to know whether this assumption holds without first running the ID3 algorithm. The result is that we do not have a good understanding of the range of problems for which ID3 is appropriate. Similarly, the backpropagation algorithm (Rumelhart, Hinton, & Williams, 1986) assumes, at a minimum, that the unknown function f can be represented as a multilayer feed-forward network of sigmoid units. Although there have been many successful applications of backpropagation (Touretzky, 1989, 1990), we still lack an understanding of the situations for which it is appropriate.

Furthermore, because clear statements of the assumptions made by ID3 and backpropagation are unavailable, we do not understand the relationship between these two algorithms. Some investigators have even suggested that these algorithms are making very similar assumptions (Lorien Pratt, personal communication).

Hence, we confront two related questions. First, what are the assumptions embodied in ID3 and backpropagation (or equivalently, in what situations should these algorithms be applied)? Second, how are ID3 and backpropagation related?

One can conceive of two different approaches to answering these questions. A theoretical approach could analyze each of these algorithms in an attempt to articulate their assumptions. An experimental approach could test these two algorithms on nontrivial problems and compare their behavior.

In this paper, we take the experimental approach. We apply ID3 and backpropagation to the task of mapping English words into their pronunciations. This task was pioneered by Sejnowski and Rosenberg (1987) in their famous NETtalk system, which employed backpropagation. Rosenberg's doctoral dissertation (1988) included further analysis and experiments in this domain. In our replication of their work, we discover that backpropagation outperforms ID3 on this task. This demonstrates that ID3 and backpropagation do not make identical assumptions.

We then go on to investigate the difference between ID3 and backpropagation. We formulate three hypotheses to explain the difference and conduct experiments to test these hypotheses. These experiments show that ID3, when combined with some simple statistical learning procedures, can nearly match the performance of BP. We also present data showing that the performance of ID3 and backpropagation is very highly correlated over a collection of binary concept learning problems. These data also show that ID3 and BP tend to agree on which of these concepts are easy and which are difficult.

Given that BP is substantially more awkward and time-consuming to apply, these results suggest the following methodology for applying these algorithms to problems similar to the NETtalk task. First, ID3, combined with our statistical learning procedures, should be applied. If its performance is adequate, then there is no need to apply backpropagation. However, if ID3's performance is inadequate, it can still be used to estimate the performance of backpropagation. Then the much more expensive backpropagation procedure can be employed to see if it yields a better classifier.

2. The Task

To conduct our comparisons of ID3 and backpropagation, we have chosen the task of mapping English text into speech. A complete text-to-speech system involves many stages of processing. Ideally, sentences are parsed to identify word senses and parts of speech. Individual words (and their senses) are then mapped into strings of phonemes and stresses. Finally, the phonemes and stresses can be combined by various techniques to generate sound waves. For an excellent review, see Klatt (1987).

A phoneme is an equivalence class of basic sounds. An example is the phoneme /p/. Individual occurrences of a /p/ are slightly different, but they are all considered /p/ sounds. For example, the two p's in "lollypop" are pronounced differently, but they are both members of the equivalence class of phoneme /p/. We use 54 phonemes (see Appendix A.1.).

Stress is the perceived weight given to a syllable in a word. For example, the first syllable of "lollypop" receives the primary stress, the third syllable receives secondary stress, and the middle syllable is unstressed. Stress information is coded by assigning one of six possible stress symbols to each letter. Consonants generally receive one of the symbols "<" or ">", which indicate that the principal vowel in this syllable is to the left or the right (respectively) of the consonant. Vowels are generally marked with a code of 0 (none), 1 (primary), or 2 (secondary) to indicate the degree of stress. Lastly, silent stress ("–") is assigned to blanks.

Let L be the set of 29 symbols comprising the letters `a`–`z`, and the comma, space, and period (in our data sets, comma and period do not appear). Let P be the set of 54 English phonemes and S be the set of 6 stresses employed by Sejnowski and Rosenberg. The task is to learn the mapping f:

$$f : L^* \longrightarrow P^* \times S^*.$$

Specifically, f maps from a word of length k to a string of phonemes of length k and a string of stresses of length k. For example,

```
f("lollypop") = ("lal-ipap", ">1<>0>2<").
```

Notice that letters, phonemes, and stresses have all been aligned so that silent letters are mapped to the silent phoneme /–/.

As defined, f is a very complex discrete mapping with a very large range. If we assume no word contains more than 28 letters (the length of "antidisestablishmentarianism"), this range would contain more than 10^{70} elements. Existing learning algorithms focus primarily on learning Boolean concepts—that is, functions whose range is the set $\{0, 1\}$. Such algorithms cannot be applied directly to learn f.

Fortunately, Sejnowski and Rosenberg (1987) developed a technique for converting this complex learning problem into the task of learning a collection of Boolean concepts. They begin by reformulating f to be a mapping g from a seven-letter window to a single phoneme and a single stress. For example, the word "lollypop" would be converted into 8 separate seven-letter windows:

```
g("___loll") = ("l", ">")
g("__lolly") = ("a", "1")
g("_lollyp") = ("l", "<")
g("lollypo") = ("-", ">")
g("ollypop") = ("i", "0")
g("llypop_") = ("p", ">")
g("lypop__") = ("a", "2")
g("ypop___") = ("p", "<")
```

The function g is applied to each of these 8 windows, and then the results are concatenated to obtain the phoneme and stress strings. This mapping function g now has a range of 324 possible phoneme/stress pairs, which is a substantial improvement.

Finally, Sejnowski and Rosenberg code each possible phoneme/stress pair as a 26-bit string, 21 bits for the phoneme and 5 bits for the stress. Each bit in the code corresponds to some property of the phoneme or stress. This converts g into 26 separate Boolean functions, $h_1, \ldots, h_{26}$. Each function h_i maps from a seven-letter window to the set $\{0, 1\}$. To assign a phoneme and stress to a window, all 26 functions are evaluated to produce a 26-bit string. This string is then mapped to the nearest of the 324 bit strings representing legal phoneme/stress pairs. We used the Hamming distance between two strings to measure distance. (Sejnowski and Rosenberg used the angle between two strings to measure distance, but they report that the Euclidean distance metric gave similar results. In tests with the Euclidean metric, we have obtained results identical to those reported in this paper.)

With this reformulation, it is now possible to apply Boolean concept learning methods to learn the h_i. However, the individual h_i must be learned extremely well in order to obtain good performance at the level of entire words. This is because errors aggregate. For example, if each h_i is learned so well that it is 99% correct and if the errors among the h_i are independent, then the 26-bit string will be correct only 77% of the time. Because the average word has about 7 letters, whole words will be correct only 16% of the time.

So far, we have only discussed the representation of the *outputs* of the mapping to be learned. The *inputs* are represented in a straightforward fashion, using the approach recommended by Sejnowski and Rosenberg (1987). Each seven-letter window is represented as the concatenation of seven 29-bit strings. Each 29-bit string represents a letter (one bit for each letter, period, comma, and blank), and hence, only one bit is set to 1 in each 29-bit string. This produces a string of 203 bits for each window. These 203 bits provide the input features for the learning algorithms.

3. The Algorithms

3.1. ID3

ID3 is a simple decision-tree learning algorithm developed by Ross Quinlan (1983, 1986b). It constructs a decision tree recursively, starting at the root. At each node, it selects, as the feature to be tested at that node, the feature a_i whose mutual information with the output classification is greatest (this is sometimes called the information gain criterion). The training examples are then partitioned into those examples where $a_i = 0$ and those where $a_i = 1$. The algorithm is then invoked recursively on these two subsets of training examples. The algorithm halts when all examples at a node fall in the same class. At this point, a leaf node is created and labelled with the class in question. The basic operation of ID3 is quite similar to the CART algorithm developed by Breiman, Friedman, Olshen, and Stone (1984) and to the tree-growing method developed by Lucassen and Mercer (1984). The algorithm has been extended to handle features with more than 2 values and features with continuous values as well.

In our implementation of ID3, we did not employ windowing (Quinlan, 1983), CHI-square forward pruning (Quinlan, 1986a), or any kind of reverse pruning (Quinlan, 1987). We did apply one simple kind of forward pruning to handle inconsistencies in the training data: If all remaining features have zero information gain, then growth of the tree was terminated in a leaf and the class having more training examples was chosen as the label for that leaf (in case of a tie, the leaf is assigned to class 0).

To apply ID3 to this task, the algorithm must be executed 26 times—once for each mapping h_i. Each of these executions produces a separate decision tree.

3.2. Backpropagation

The error backpropagation method (Rumelhart, Hinton, & Williams, 1986) is widely applied to train artificial neural networks. However, in its standard form, the algorithm requires substantial assistance from the user. Specifically, the user must specify the transfer function of each artificial neuron (unit), the network architecture (number of layers and their interconnections), the number of hidden units in each layer, the learning rate, the momentum term, the initial weight values, and the target thresholds.[1] Furthermore, the user must decide when to terminate training. To make the comparison between ID3 and backpropagation fair, it is necessarily to transform BP from a user-assisted method into an algorithm that involves no user assistance.

We have developed such a transformation. We call the resulting algorithm BPCV (BackPropagation with Cross-Validation). To define BPCV, we fix some of the user-specified properties and set the remaining parameters via cross-validation using the methods introduced by Lang, Waibel, and Hinton (1990) as explained below.

In BPCV, there is only one hidden layer, and it is fully connected to the input layer and to the output layer. Every unit in the hidden and output layers is implemented by taking the dot product of a vector of weights $\mathbf{w}$ with a vector of incoming activations $\mathbf{x}$, adding a bias θ, and applying the logistic function

$$y = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + \theta)}},$$

which is a continuous, differentiable approximation to the linear threshold function used in perceptrons. Several parameters are given fixed values: the learning rate is always 0.25, the momentum term is 0.9, and target thresholds are not used. The criterion to be minimized is the sum squared error (SSE). These are basically the same parameters (except for the target thresholds) that were used by Sejnowski and Rosenberg. We have conducted some cross-validation and found that performance was insensitive to these parameter choices.

The remaining parameters—number of hidden units, random starting weights, and stopping total sum squared error (TSSE)—are set by the following cross-validation procedure. Given a set of examples S, we subdivide S into three sets: a training set (S_{tr}), a cross-validation set (S_{cv}), and a test set (S_{test}). Then we execute backpropagation several times on the training set S_{tr} while varying the number of hidden units and the random starting weights. After each pass through the training data, we test the performance of the network on S_{cv}. The goal of this search of parameter space is to find those parameters that give peak performance on the cross-validation set. These parameters can then be used to train backpropagation on the union $S_{tr} \cup S_{cv}$, and a good estimate of generalization performance can be obtained by testing with S_{test}.

The advantage of cross-validation training is that no information from the test set is employed during training, and hence, the observed error rate on the test set is a fair estimate of the true error rate of the learned network. This contrasts with the common, but unsound practice of adjusting parameters to optimize performance on the test set.

One advantage of BPCV on the NETtalk task is that, unlike ID3, it is only necessary to apply BPCV once, because all 26 output bits can be learned simultaneously. Indeed, the 26 outputs all share the same set of hidden units, which may allow the outputs to be learned more accurately. However, while ID3 is a batch algorithm that processes the entire training set at once, BP is an incremental algorithm that makes repeated passes over the data. Each complete pass is called an "epoch." During an epoch, the training examples are inspected one-at-a-time, and the weights of the network are adjusted to reduce the squared error of the outputs. We used the implementation provided with McClelland and Rumelhart (1988).

Because the outputs from BP are floating point numbers between 0 and 1, we had to adapt the Hamming distance measure when mapping to the nearest legal phoneme/stress pair. We used the following distance measure: $d(\mathbf{x}, \mathbf{y}) = \sum_i |x_i - y_i|$. This reduces to the Hamming distance when $\mathbf{x}$ and $\mathbf{y}$ are Boolean vectors.

Table 1. Optimal network size via cross-validation

Number of Hidden Units	Letters (% Correct)	TSSE	Number of Epochs
40	67.0	2289	28
60	67.7	939	46
80	68.3	1062	25
100	69.3	1041	19
120	68.7	1480	12
140	70.0	541	27
160	70.7	445	37
180	69.3	477	28

3.3. The Data Set

Sejnowski and Rosenberg provided us with a dictionary of 20,003 words and their corresponding phoneme and stress strings. From this dictionary we drew at random (and without replacement) a training set of 800 words, a cross-validation set of 200 words, and a test set of 1000 words.

4. Results

4.1. Cross-validation Training

Before presenting the results of our study, we first discuss the results of the cross-validation procedure for BPCV. We performed a series of runs that systematically varied the number of hidden units (40, 60, 80, 100, 120, 140, 160, and 180) and the random starting weights (four sets of random weights were generated for each network). Performance on the cross-validation set was evaluated after each complete pass through the training data (epoch). The networks were trained for 30 epochs (except for a few cases, where training was continued to 60 epochs to ensure that the peak performance had been found). Table 1 shows the peak performance (percent of letters correctly pronounced) for each network size and the total sum squared error (on S_{tr}) that gave the peak performance. These TSSE numbers (appropriately adjusted for the number of training examples) can then be used to decide when to terminate training on the entire training set ($S_{tr} \cup S_{cv}$). Based on these runs, the best network size is 160 hidden units.

Having completed cross-validation training, we then proceeded to merge the training set and cross-validation set to form a 1000-word training set. During cross-validation training, we stored a snapshot of the weight values after the first complete epoch for each random network that was generated. Hence, to perform training on the entire training set, we used the best stored 160-hidden unit snapshot as a starting point.[2] The original training set S_{tr} contained 5,807 seven-letter windows, while

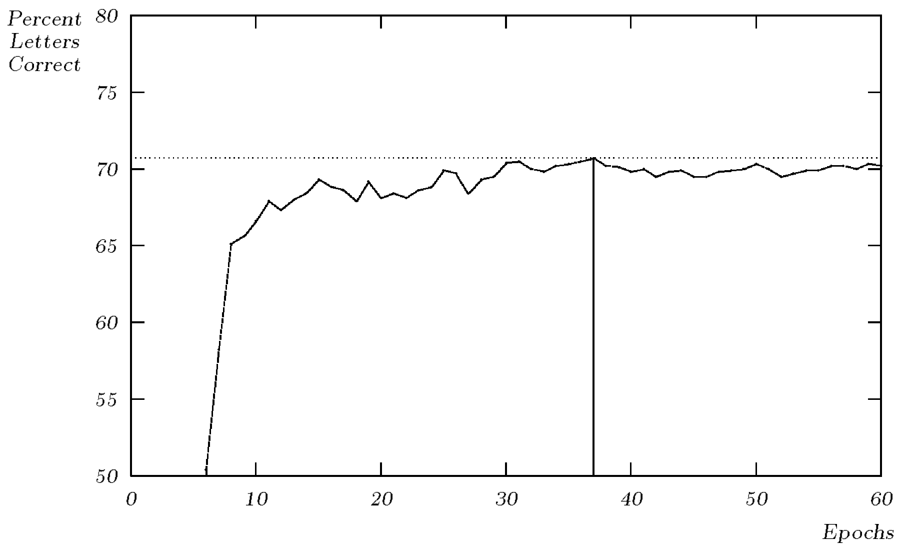

Figure 1. Training curve for the best 160-hidden unit network. Vertical bar indicates point of maximum performance.

the full training set $S_{tr} \cup S_{cv}$ contains 7,229 seven-letter windows. Hence, the target TSSE for the full training set was 554.

We were surprised by the figures shown in Table 1, since we expected that a reasonably small network (e.g., 80 hidden units) would give a good fit to the data. However, the table clearly shows that generalization steadily improves as the quality of the fit to the training data improves. Furthermore, Figure 1 shows that as training of a network continues past the point of peak performance, performance does not decline appreciably.

Previous work by Sejnowski and Rosenberg (1986) and Rosenberg (1988) has used networks with 40, 80, and 120 hidden units. However, to our knowledge, no one has previously conducted a systematic study of the relationship between network size and performance on the NETtalk task. Similar results showing that larger networks can give improved performance have been published by Martin and Pittman (1990).

4.2. Performance Comparison

Table 2 shows percent correct (over the 1000-word test set) for words, letters, phonemes, and stresses. A letter is considered correct if both the phoneme and the stress were correctly predicted (after mapping to the nearest legal phoneme and

Table 2. Percent correct over 1000-word test set

	Level of Aggregation (% correct)				
Method	Word	Letter	Phoneme	Stress	Bit (mean)
ID3	9.6	65.6	78.7	77.2	96.1
BPCV	13.6**	70.6***	80.8***	81.3***	96.7*

Difference in the cell significant at $p < .05^*$, $.005^{**}$, $.001^{***}$

		Backpropagation		
		Correct	Incorrect	
ID3	Correct	4239 (58.5%)	512 (7.1%)	Disagree: 1385 (19.2%)
	Incorrect	873 (12.1%)	1618 (22.3%)	Agree: 5857 (80.9%)

stress). A word is correct if all of its letters are correct. Virtually every difference in the table at the word, letter, phoneme, and stress levels is statistically significant (using a one-tailed test for the difference of two proportions based on the normal approximation to the binomial distribution). Hence, we conclude that there is a substantial difference in performance between ID3 and BPCV on this task.

It should be noted that although the test set contains 1000 disjoint words, some of the seven-letter windows in the test set also appear in the training set. Specifically, 946 (13.1%) of the windows in the test set appear in the 1000-word training set. These represent 578 distinct windows. Hence, the performance at the letter, phoneme, and stress levels are all artificially high if one is concerned about the ability of the learning methods to handle unseen cases correctly. However, if one is interested in the probability that a letter (or phoneme, or stress) in an unseen word will be correctly classified, then these numbers provide the right measure.

To take a closer look at the performance difference, we can study exactly how each of the 7,242 seven-letter windows in the test set are handled by each of the algorithms. Table 2 categorizes each of these windows according to whether it was correctly classified by both algorithms, by only one of the algorithms, or by neither one.

The table shows that the windows correctly learned by BPCV do not form a superset of those learned by ID3. Instead, the two algorithms share 4,239 correct windows, and then each algorithm correctly classifies several windows that the other algorithm gets wrong. The overall result is that BPCV classifies 361 more windows

Table 3. Average percent correct (1000-word test set) over five trials.

	Level of Aggregation (% correct)				
Method	Word	Letter	Phoneme	Stress	Bit (mean)
ID3	10.2	65.2	79.1	76.5	96.1
BP	15.1****	71.3****	81.3****	81.7****	96.7****

Difference in the cell significant at $p < .0001$****

correctly than does ID3. This shows that the two algorithms, while they overlap substantially, have learned fairly different text-to-speech mappings.

The information in this table can be summarized as a correlation coefficient. Specifically, let X_{ID3} (X_{BPCV}) be a random variable that is 1 if and only if ID3 (BPCV, respectively) makes a correct prediction at the letter level. In this case, the correlation between X_{ID3} and X_{BPCV} is .5648. If all four cells of Table 2 were equal, the correlation coefficient would be zero. (For reference, independent runs of BPCV on the same training set, but with different random starting states have a correlation coefficient .6955.)

A weakness of Table 2 is that it shows performance values for one particular choice of training and test sets. We have replicated this study four times (for a total of 5 independent trials). In each trial, we again randomly drew without replacement two sets of 1000 words from the dictionary of 20,003 words. Note that this means that there is some overlap among the five training sets (and among the five test sets). Table 3 shows the average performance of these 5 runs. All differences are significant below the .0001 level using a t-test for paired differences.

Another weakness of Table 2 is that it only shows performance values for a 1000-word training set. It might be that the relative performance difference between ID3 and BPCV might change as the size of the training set changes. Table 4 shows that this is not the case. The rows in the table give the results of running ID3 and BPCV on several different sizes of training sets. In each case, BPCV was trained using the cross-validation training methodology outlined above (four runs each with networks having 5, 10, 20, 40, 80, 120, and 160 hidden units). The only difference from the methodology outlined above is that after training on S_{tr} and determining peak generalization performance by testing on a 200-word S_{cv}, we did *not* re-train on the union $S_{tr} \cup S_{cv}$, since this would create training sets that were too large. Instead, we simply tested the best network on the 1000-word S_{test}. We conclude that there is a consistent difference between ID3 and BPCV and that, while the performance of both algorithms will increase with the size of the training set, this difference will still be observed.

In the remainder of this paper, we will attempt to understand the nature of the differences between BPCV and ID3. Our main approach will be to experiment with modifications to the two algorithms that enhance or eliminate the differences

Table 4. Percent correct over 1000-word test set.

Sample		Level of Aggregation (% correct)				
Size	Method	Word	Letter	Phoneme	Stress	Bit (mean)
50	ID3	0.8	41.5	60.5	60.1	93.1
	BPCV	1.6	49.2***	59.7	73.1***	93.9
100	ID3	2.0	47.3	64.1	65.8	94.0
	BPCV	3.7*	55.5***	66.6**	75.4***	94.8*
200	ID3	4.4	56.6	70.5	72.2	95.1
	BPCV	7.1**	61.1***	72.2*	78.2***	95.4
400	ID3	6.2	58.7	73.7	72.1	95.5
	BPCV	11.3***	66.4***	77.0***	79.7***	96.0
800	ID3	9.6	63.8	77.8	75.6	96.2
	BPCV	15.3***	70.9***	81.0***	81.2***	96.6
1000	ID3	9.6	65.6	78.7	77.2	96.4
	BPCV	14.7***	70.9***	81.1***	81.4***	96.6

Difference in the cell significant at $p < .05^*$, $.01^{**}$, $.001^{***}$

Table 5. Results of applying three overfitting-prevention techniques.

		Level of Aggregation (% correct)				
Method	Data set	Word	Letter	Phoneme	Stress	Bit (mean)
(a) ID3 (as above)	TEST:	9.6	65.6	78.7	77.2	96.1
(b) ID3 (χ^2 cutoff)	TEST:	9.1	64.8	78.4	77.1	96.1
(c) ID3 (pruning)	TEST:	9.3	62.4	76.9	75.1	95.8
(d) ID3 (rules)	TEST:	8.2	65.1	78.5	77.2	96.1

between them. Unless stated otherwise, all of these experiments are performed using only the 1000-word training set and 1000-word test set from Table 2.

5. Three Hypotheses

What causes the differences between ID3 and BPCV? We have three hypotheses:

Hypothesis 1: Overfitting. ID3 has overfit the training data, because it seeks complete consistency. This causes it to make more errors on the test set.

Hypothesis 2: Sharing. The ability of BPCV to share hidden units among all of the h_i allows it to reduce the aggregation problem at the bit level and hence perform better.

Hypothesis 3: Statistics. The numerical parameters in the network allow it to capture statistical information that is not captured by ID3.

These hypotheses are neither mutually exclusive nor exhaustive.

The following three subsections present the experiments that we performed to test these hypotheses.

5.1. Tests of Hypothesis 1 (Overfitting)

The tendency of ID3 to overfit the training data is well established in cases where the data contain noise. Three basic strategies have been developed for addressing this problem: (a) criteria for early termination of the tree-growing process, (b) techniques for pruning trees to remove overfitting branches, and (c) techniques for converting the decision tree to a collection of rules. We implemented and tested one method for each of these strategies. Table 5 summarizes the results.

The first row repeats the basic ID3 results given above, for comparison purposes. The second row shows the effect of applying a χ^2 test (at the .90 confidence level) to decide whether further growth of the decision tree is statistically justified (Quinlan, 1986a). As other authors have reported (Mooney et al., 1989), this hurts performance in the NETtalk domain. The third row shows the effect of applying Quinlan's technique of reduce-error pruning (Quinlan, 1987). Mingers (1989) provides evidence that this is one of the best pruning techniques. For this row, a decision tree was built using the 800-word S_{tr} set and then pruned using the S_{cv} cross-validation set. Finally, the fourth row shows the effect of applying a method for converting a decision tree to a collection of rules. Quinlan (1987) describes a three-step method for converting decision trees to rules. First, each path from the root to a leaf is converted into a conjunctive rule. Second, each rule is evaluated to remove unnecessary conditions. Third, the rules are combined, and unnecessary rules are eliminated. In this experiment, we performed only the first two steps, because the third step was too expensive to execute on this rule set, which contains 6,988 rules.

None of these techniques improved the performance of ID3 on this task. This suggests that Hypothesis 1 is incorrect: ID3 is not overfitting the data in this domain. This makes sense, since the only source of "noise" in this domain is the limited size of the seven-letter window and the existence of a small number of words like "read" that have more than one correct pronunciation. Seven-letter windows are sufficient to correctly classify 98.5% of the words in the 20,003-word dictionary. This may also explain why we did not observe overfitting during excessive training in our cross-validation runs with backpropagation either.

5.2. *A Test of Hypothesis 2 (Sharing)*

The second hypothesis claims that the key to BPCV's superior performance is the fact that all of the output units share a single set of hidden units. One obvious way to test this sharing hypothesis would be to develop a version of ID3 that permitted sharing among the 26 separate decision trees being learned. We could then see if this "shared-ID3" improved performance. An alternative is to *remove* sharing from backpropagation, by training 26 independent networks, each having only one output unit, to learn the 26 h_i mappings. If Hypothesis 2 is correct, then, because there is no sharing among these separate networks, we should see a drop in performance compared to the single network with shared hidden units. Furthermore, the decrease in performance should decrease the differences between BPCV and ID3 as measured by the correlation between their errors. We will call the single network, in which all hidden units are shared by the output units, BP1; and we will call the 26 separate networks, BP26.

There is a delicate issue that arises in training BP26. Ideally, we want to train a collection of 26 networks so that the only differences between them and BP1 result from the lack of shared hidden units. This means that the total summed squared error (on the training set) for BP26 should be the same as for BP1. The goal of the training procedure is to find, among all such BP26 networks, the collection whose performance on the cross-validation set is maximized.

Hence, we used the following procedure. First, we measured the sum of the squared error (over the training set) of each of the 26 bits learned by BP1. Second, to train the BP26 networks, we followed the cross-validation procedure of trying alternative random seeds and numbers of hidden units, but this time we always terminated training when each individual network attained the squared error observed in the large network. During cross-validation, we tried networks having 1, 2, 3, 4, 5, 10, and 20 hidden units (with four random seeds for each network size). Finally, we selected the network whose summed squared error was minimum on the 200-word cross-validation test set (S_{cv}).

Surprisingly, we were unable to train successfully the separate networks to the target error level on the 1000-word training set. We explored smaller subsets of the 1000-word training set (800, 400, 200, 100, and 50-words) and found that training did not succeed until the training set contained only 50 words! For the 100-word training set, for example, the individual networks often converged to local minima (even though the BP1 network had avoided these minima). Specifically, bits 4, 6, 13, 15, 18, 21, and 25 could not be trained to criterion, even after 2000 epochs.

For bit 18 on the 100-word training set, we conducted a detailed study in an attempt to understand this training problem. We performed hundreds of runs while varying the number of hidden units, the learning rate, the momentum term, and the initial random weights in an attempt to find a configuration that could learn this single bit to the same level as BP1. None of these runs succeeded. In each run of BP26, training converged such that the error on all but a handful of the training examples was 0.0, and the error on the remaining training examples was 1.0. In contrast, the errors of BP1 were not so extreme.

Table 6 shows a collection of seven-letter windows from the test set and the squared error on each of these windows for nine different training runs. The first training run is for BP1 with 120 units trained for 30 epochs. The next four columns show runs of BP26 with a 5-hidden-unit network and four different random starting seeds (these were trained with learning rate .4, momentum .8, and initial random values in the range [-0.5,+0.5]). The last four columns show runs of BP26 with a 10-hidden-unit network and four different random starting seeds (these were trained with learning rate .4, momentum .7, and initial random values in the range [-0.4,+0.4]).

This demonstrates that even if shared hidden units do not aid classification performance, they certainly aid the learning process!

As a consequence of this training problem, we are able to report results for only the 50-word training set. Cross-validation training for BP1 (see above) determined that the best network for this training set contained 120 hidden units trained to a sum-squared error of 13.228. Table 7 summarizes the training process for each of the 26 output bits for BP26. Each row gives the number of hidden units in the best BP26 network, the squared error obtained from the BP1 network, the squared error obtained from the BP26 network, and the number of epochs required for training BP26. Notice that each individual bit was slightly over-trained as compared with BP1. This is because the program accumulates the squared errors during an epoch and stops when this falls below the target error level. Because performance improves during an epoch, the final squared error is somewhat lower.

Table 8 shows the performance of these 26 networks on the training and test sets. Performance on the training set is virtually identical to the 120-hidden-unit

Table 6. Comparison of individual errors on Bit 18

window	BP1	BP26							
		5 hidden units				10 hidden units			
		1	2	3	4	1	2	3	4
"__AUSTR"								1.0	
"SOTTED_"								1.0	
"BREADWI"	0.021	1.0			1.0	1.0	1.0	1.0	1.0
"BUCKSAW"							1.0	1.0	
"MOIS___"	1.000						1.0		
"CINNAMO"	0.041	1.0				1.0		1.0	1.0
"FIGURAT"	−0.026	1.0	1.0		1.0	1.0		1.0	1.0
"CORRUPT"	0.031	1.0	1.0	1.0	1.0	1.0		1.0	1.0
"_LAWYER"	0.044	1.0		1.0	1.0	1.0	1.0	1.0	1.0
"_MUUMUU"	0.020			1.0				1.0	
"PETTIFO"	0.028	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
"ILTON__"	1.000	1.0		1.0	1.0	1.0	1.0	1.0	1.0

Values not shown are 0.00

Table 7. Training Statistics for 26 Independent Networks.

Bit	Number of hidden units	Squared error in BP1 network	Squared error in BP26 network	Number of epochs
1	2	0.8179	0.678	62
2	3	0.1362	0.126	56
3	20	0.0594	0.044	46
4	3	0.1047	0.094	64
5	3	1.0514	1.049	119
6	20	0.0447	0.037	21
7	10	0.0746	0.061	23
8	1	0.0365	0.035	41
9	4	1.9208	1.894	32
10	10	0.0279	0.026	20
11	1	0.0229	0.022	53
12	5	0.0374	0.035	35
13	5	1.0665	1.055	56
14	4	0.0380	0.034	21
15	5	0.0590	0.056	41
16	10	2.0629	2.023	65
17	10	4.2645	4.157	35
18	3	0.0442	0.041	56
19	20	0.0008	0.000	2
20	20	0.0009	0.000	2
21	3	0.0413	0.039	84
22	5	0.0894	0.074	61
23	2	0.0422	0.038	269
24	4	0.0648	0.057	31
25	5	1.1184	0.877	45
26	20	0.0011	0.000	2

Table 8. Performance of 26 separate networks compared with a single network having 120 shared hidden units. Trained on 50-word training set. Tested on 1000-word test set.

			Level of Aggregation (% correct)			
Method	Data set	Word	Letter	Phoneme	Stress	Bit (mean)
(a) ID3	TEST:	0.8	41.5	60.5	60.1	92.6
(b) BP 26 separate nets	TRAIN:	92.0	99.0	99.0	100.0	99.9
	TEST:	1.7	46.3	57.9	72.2	93.2
(c) BP 120 hidden units	TRAIN:	92.0	98.7	99.0	99.7	99.9
	TEST:	1.6	49.2	59.7	73.1	93.4
Difference (b)-(c)	TRAIN:	0.0	+0.3	0.0	+0.3	0.0
	TEST:	+0.1	−2.9***	−1.8*	−0.9*	−0.2
Difference (a)-(c)	TEST:	−0.8	−7.7	+0.8	−13.0	−1.3

Difference significant at $p < .05^*$, $.001^{***}$

Table 9. Error Correlations

	Correlation Coefficients	
Replication	X_{ID3} and X_{BP1}	X_{ID3} and X_{BP26}
a	.5167	.4942
b	.5005	.4899
c	.5347	.5062
d	.4934	.4653
e	.4934	.4790
Average Decrease		.0208

network, which shows that our training regime was successful. Performance on the test set, however, shows a loss of performance when there is no sharing of the hidden units among the output units. Hence, it suggests that Hypothesis 2 is at least partially correct.

However, examination of the correlation between ID3 and BPCV indicates that this is wrong. The correlation between X_{ID3} and X_{BP1} (i.e., BP on the single network) is .5428, whereas the correlation between X_{ID3} and X_{BP26} is .5045.

We have replicated this comparison 5 times, over 5 different training and testing sets (using a less rigorous, but more efficient, un-cross-validated training procedure). Table 9 shows the resulting correlation coefficients. A paired differences t-test shows that the differences in correlation coefficients are significant below the .0001 level.

Hence, the removal of shared hidden units has actually made ID3 and BP less similar, rather than more similar as Hypothesis 2 claims. The conclusion is that sharing in backpropagation is important to improving both its training and its performance, but it does not explain why ID3 and BPCV are performing differently.

5.3. *Tests of Hypothesis 3: Statistics*

We performed three experiments to test the third hypothesis that the continuous parameters in BPCV networks are able to capture statistical information that ID3 fails to capture.

In the first experiment, we took the outputs of the backpropagation network and thresholded them (values $> .5$ were mapped to 1, values $\leq .5$ were mapped to 0) before mapping to the nearest legal phoneme/stress pair. Thresholding the values can change the distances that are measured between the outputs and the legal phoneme and stress patterns. Table 10 presents the results for the 1000-word training set.

The results show that thresholding significantly drops the performance of backpropagation. Indeed, at the phoneme level, the decrease is enough to push BPCV below ID3. At the other levels of aggregation, BPCV still out-performs ID3. These

Table 10. Performance of backpropagation with thresholded output values. Trained on 1000-word training set. Tested on 1000-word test set.

		Level of Aggregation (% correct)				
Method	Data set	Word	Letter	Phoneme	Stress	Bit (mean)
(a) ID3 (legal)	TEST:	9.6	65.6	78.7	77.2	96.1
(b) BPCV (legal)	TEST:	13.6	70.6	80.8	81.3	96.7
(c) BPCV (thresholded)	TEST:	11.2	67.7	78.4	80.0	96.3
Difference (c)-(b)	TEST:	−2.4	−2.9***	−2.4***	−1.3*	−0.4

Difference significant at $p < .05^*$, $.001^{***}$

results support the hypothesis that the continuous outputs of the neural network aid the performance of BPCV.

However, thresholding the outputs of BPCV does not cause it to behave substantially more like ID3. The correlation between X_{ID3} and $X_{BPCV thresh}$ is .5685 (as compared with .5648 for X_{BPCV})—this is only a small increase. Close examination of the data shows that the seven-letter windows "lost" (i.e., incorrectly classified) when BPCV is thresholded include 120 windows correctly classified by ID3 and 112 windows incorrectly classified by ID3. Hence, the mistakes introduced by thresholding are nearly independent of the mistakes made by ID3.

While this experiment demonstrates the importance of continuous outputs, it does not tell us what kind of information is being captured by these continuous outputs nor does it reveal anything about the role of continuous weights inside the network. For this, we must turn to the other two experiments.

In the second experiment, we modified the method used to map each output 26-bit string into one of the 324 legal phoneme/stress pairs. Instead of considering all possible legal phoneme/stress pairs, we restricted attention to those phoneme/stress pairs that had been observed in the training data. Specifically, we constructed a list of every phoneme/stress pair that appears in the training set (along with its frequency of occurrence). Appendix A.3. shows this frequency information for the 1000-word training set. During testing, the 26-element vector produced either by ID3 or BPCV is mapped to the closest phoneme/stress pair appearing in this list. Ties are broken in favor of the most frequent phoneme/stress pair. We call this the "observed" decoding method, because it is sensitive to the phoneme/stress pairs (and frequencies) observed in the training set.

Table 11 presents the results for the 1000-word training set and compares them to the previous technique ("legal") that decoded to the nearest legal phoneme/stress pair. The key point to notice is that this decoding method leaves the performance of BPCV virtually unchanged, while it substantially improves the performance of ID3. Indeed, it eliminates a substantial part of the difference between ID3 and BPCV—the two methods are now statistically indistinguishable at the word and phoneme levels. Mooney et al. (1989), in their comparative study of ID3 and BPCV on this same task, employed a version of this decoding technique (with random tie-breaking), and obtained very similar results when training on a set of the 808 words in the dictionary that occur most frequently in English text.

An examination of the correlation coefficients shows that "observed" decoding increases slightly the similarity between ID3 and BPCV. The correlation between $X_{ID3observed}$ and $X_{BPobserved}$ is .5865 (as compared with .5648 for "legal" decoding). Furthermore, "observed" decoding is almost always monotonically better (i.e., windows incorrectly classified by "legal" decoding become correctly classified by "observed" decoding, but not vice versa). Table 12 shows these results and four replications. A paired-differences t-test concludes that the correlation coefficient increases with observed decoding (with significance level better than .0001).

From these results, we can conclude that BPCV was already capturing most of the information about the frequency of occurrence of phoneme/stress pairs, but that

Table 11. Effect of "observed" decoding on learning performance.

		Level of Aggregation (% correct)				
Method	Data set	Word	Letter	Phoneme	Stress	Bit (mean)
(a) ID3 (legal)	TEST:	9.6	65.6	78.7	77.2	96.1
(b) BPCV (legal)	TEST:	13.6**	70.6***	80.8***	81.3***	96.7*
(c) ID3 (observed)	TEST:	13.0	70.1	81.5	79.2	96.4
(d) BPCV (observed)	TEST:	14.3	71.5*	82.0	81.4***	96.7
ID3 Improvement: (c)-(a)	TEST:	3.4***	4.5***	2.8***	2.0**	0.3
BPCV Improvement: (d)-(b)	TEST:	0.7	0.9	1.2*	0.1	0.0

Difference in the cell significant at $p < .05^{*}$, $.005^{**}$, $.001^{***}$

Table 12. Correlation between ID3 and BPCV with observed decoding.

Data Set	Legal	Observed
a	.5648	.5865
b	.5844	.5945
c	.5593	.5796
d	.5722	.5706
e	.5563	.5738
Average Increase		.0136

ID3 was not capturing nearly as much. Hence, this experiment strongly supports Hypothesis 3.

A drawback of the "observed" strategy is that it will never decode a window to a phoneme/stress pair that it has not seen before. Hence, it will certainly make some mistakes on the test set. However, phoneme/stress pairs that have not been observed in the training set make up a very small fraction of the windows in the test set. For example, only 7 of the phoneme/stress pairs that appear in our 1000-word test set do not appear in the 1000-word training set. In the test set, they only account for 11 of the 7,242 windows (0.15%). If we were to train on all 19,003 words from the dictionary that do not appear in our 1000-word test set, there would be only one phoneme/stress pair present in the test set that would not appear in the training set, and it would appear in only one window.

The final experiment concerning Hypothesis 3 focused on extracting additional statistical information from the training set. We were motivated by Klatt's (1987) view that ultimately letter-to-phoneme rules will need to identify and exploit morphemes (i.e., commonly-occurring letter sequences appearing within words). Therefore, we analyzed the training data to find all letter sequences of length 1, 2, 3, ..., k, and retained the B most-frequently-occurring sequences of each length. The parameters k and B are determined by cross-validation, as described below. For each retained letter sequence, we formed a list of all phoneme/stress strings to which that sequence is mapped in the training set (and their frequencies). For example, here are the five pronunciations of the letter sequence "ATION" in the training set (Format is ($\langle$*phoneme string*$\rangle$ $\langle$*stress string*$\rangle$ $\langle$*frequency*$\rangle$)).

```
(("eS-xn" "1>0<<" 22)
 ("@S-xn" "1<0<<" 1)
 ("eS-xn" "2>0<<" 1)
 ("@S-xn" "2<0>>" 1)
 ("@S-xn" "1<0>>" 1))
```

During decoding, each word is scanned (from left to right) to see if it contains one of the "top B" letter sequences of length l (varying l from k down to 1). If a word contains such a sequence, the letters corresponding to the sequence are processed as follows. First, each of the l windows centered on letters in the sequence is evaluated (i.e., by the 26 decision trees or by the feed-forward network) to obtain a 26-bit string, and these strings are concatenated to produce a bit string of length $l \cdot 26$. Then, each of the observed pronunciations for the sequence is converted into an $l \cdot 26$-bit string according to the code given in Appendix A.1.. Finally, the "unknown" string is mapped to the nearest of these observed bit strings.

After decoding a block, control skips to the end of the matched l-letter sequence and resumes scanning for another "top B" letter sequence of length l. After this scan is complete, the parts of the word that have not yet been matched are re-scanned to look for blocks of length $l - 1$. Every letter in the word is eventually processed, because every individual letter is a block of length 1. We call this technique "block" decoding.

Table 13. Effect of "block" decoding on learning performance.

		Level of Aggregation (% correct)				
Method	Data set	Word	Letter	Phoneme	Stress	Bit (mean)
(a) ID3 (legal)	TEST:	9.6	65.6	78.7	77.2	96.1
(b) BPCV (legal)	TEST:	13.6**	70.6***	80.8***	81.3***	96.7*
(c) ID3 (block)	TEST:	17.2	73.3	83.9	80.4	96.7
(d) BPCV (block)	TEST:	18.5	73.7	83.8	81.3	96.7
ID3 Improvement: (c)-(a)	TEST:	7.6***	7.7***	5.2***	3.2***	0.6
BPCV Improvement: (d)-(b)	TEST:	4.9**	3.1***	3.0***	0.0	0.0

Difference in the cell significant at $p < .05^{*}$, $.005^{**}$, $.001^{***}$

		Backpropagation		
		Correct	Incorrect	
ID3	Correct	4773 (65.9%)	533 (7.4%)	Disagree: 1095 (15.1%)
	Incorrect	562 (7.8%)	1374 (19.0%)	Agree: 6147 (84.9%)

We employed cross-validation to determine the maximum block length (k) and the number of blocks (B) to store by evaluating different values while training on 800 words and testing on the 200-word cross-validation testing set. We tried values of 1, 2, 3, 4, 5, and 6 for k and values of 100, 200, 300, and 400 for B. For ID3, peak performance was attained with $k = 2$ and $B = 100$. For BPCV, peak performance was attained with $k = 2$ and $B = 200$. In both cases, performance was much more sensitive to k than to B.

Table 13 shows the performance results on the 1000-word test set. Block decoding significantly improves both ID3 and BPCV, but again, ID3 is improved much more (especially below the word level). Indeed, the two methods cannot be distinguished statistically at any level of aggregation. Furthermore, the correlation coefficient between $X_{ID3block}$ and $X_{BPblock}$ is .6122, which is a substantial increase compared to .5648 for legal decoding. Hence, block decoding also makes the performance of ID3 and BPCV much more similar. Table 13 shows how the 7,242 seven-letter windows of the test set are handled by ID3 and BPCV.

Table 14 shows these correlation coefficients, along with four replications. A paired-differences t-test concludes that the correlation coefficient increases with block decoding (with significance level better than .0001).

Table 14. Correlation between ID3 and BPCV with block decoding.

Table 15. Classification of test set windows by ID3 and BPCV with "block" decoding.

Data Set	Legal	Block
a	.5648	.6122
b	.5844	.6177
c	.5593	.6138
d	.5722	.5832
e	.5563	.6028
Average Increase		.0385

Note that any method that supplies additional information to both ID3 and BPCV could be expected to improve the correlation between the algorithms somewhat. Furthermore, any source of new information would probably benefit the poorer performing algorithm (ID3) more than the better performing algorithm. Nonetheless, the fact that block decoding eliminates all differences between ID3 and BPCV provides strong evidence that we have identified an important cause of the difference between the two methods and that Hypothesis 3 is correct. The experiment also suggests that the block decoding technique is a useful adjunct to any learning algorithm applied in this domain.

6. Discussion

6.1. Improving These Algorithms

There are many directions that can be explored for improving these algorithms. We have pursued several of these directions in order to develop a high-performance text-to-speech system. Our efforts are reported in detail elsewhere (Bakiri, 1991).

One approach is to design better output codes for phoneme/stress pairs. Our experiments have shown that BCH error correcting codes provide better output codes than the output code used in this paper. Randomly-generated bit-strings produce similar performance improvements (see Dietterich & Bakiri, 1991).

Another approach is to widen the seven-letter window and introduce context. Lucassen and Mercer (1984) employ a 9-letter window. They also include as inputs the phonemes and stresses of the four letters to the left of the letter at the center of the window. These phonemes and stresses can be obtained, during execution, from the letters that have already been pronounced during the scan from left-to-right. Our experiments (with a 15-letter window) indicate that this produces substantial performance gains as well. However, we find that it works better if the word is scanned from right-to-left instead.

A third technique for improving performance is to supply additional input features to the program. One feature of letters that helps is a bit indicating whether the letter is a vowel or a consonant. A feature of phonemes that helps is whether the phoneme is tense or lax.

A fourth technique to be pursued is to refine the block decoding method. Blocks should be chosen more carefully with some consideration of statistical confidence. Decoding should consider overlapping blocks.

A fifth direction that we have pursued is to implement Buntine's (1990) method for obtaining class probability estimates from decision trees. His algorithm produces fairly accurate probability estimates at the leaves of each decision tree. We then use these estimates to map to the nearest phoneme/stress pair. We were curious to know whether this approach would capture the same statistical information provided by observed and block decoding. Our experiments showed, however, that observed and block decoding are superior to simply using legal decoding (or even observed decoding) with class probability trees.

Table 16. Best configuration: ID3, 15-letter window, 127-bit error correcting code, seven-letter phoneme and stress context, domain-specific input features, observed decoding, simplified stresses.

	Level of Aggregation (% correct)				
Training set	Word	Letter	Phoneme	Stress	Bit (mean)
1,000 words	40.6	84.1	87.0	91.4	92.1
19,003 words	64.8	91.4	93.7	95.1	95.7

By combining the error-correcting output codes with a wider window, a right-to-left scan to include phoneme and stress context, and domain-specific features, we have obtained excellent performance with our 1000-word training and test sets. Table 16 shows our best-performing configuration when trained on 1000 words and when trained on 19,003 words. Details of this configuration are described in Bakiri (1991). We have been unable to test a similar configuration with BPCV because of the huge computational resources that would be required.

Bakiri (1991) describes a study in which human judges compared the output of this system to the output of the DECtalk (Klatt, 1987) letter-to-sound rule base. The results show that this system (and two other machine learning approaches) significantly out-perform DECtalk.

6.2. Applying ID3 to Aid BPCV

An interesting observation from this and other studies is that the performance of ID3 and BPCV is highly correlated. This suggests a methodology for using ID3 to aid BPCV even in domains where BPCV out-performs ID3. In many real-world applications of inductive learning, substantial "vocabulary engineering" is required in order to obtain high performance. This vocabulary engineering process typically involves the iterative selection and testing of promising features. To test the features, it is necessary to train a BPCV network using them—which is very time-consuming. Because the performance ID3 is correlated with BPCV, it can be used instead to test feature sets. Once a good set of features is identified, a BPCV network can then be trained.

To examine this idea in more detail, consider Table 17. This shows the performance of ID3 and BPCV on each of the 26 individual bits (i.e., without decoding them at all). (Each algorithm was trained on the 1000-word training set and tested on the 1000-word test set. A 160-hidden unit network was employed with BPCV.) The correlation coefficient is .9817, which is significant well below the .001 level. Hence, we conclude that the generalization performance of ID3 is a very good predictor of the generalization performance of BPCV.

Table 17. Performance, complexity, and difficulty of learning. 1000-word training set, 1000-word test set.

	ID3		BP	
bit	windows	(%)	windows	(%)
1	6984	96.4	6964	96.2
2	6779	93.6	6767	93.4
3	7104	98.1	7110	98.2
4	6936	95.8	6908	95.4
5	6584	90.9	6627	91.5
6	7065	97.6	7057	97.4
7	7207	99.5	7191	99.3
8	7213	99.6	7205	99.5
9	7206	99.5	7203	99.5
10	7237	99.9	7236	99.9
11	7240	100.0	7238	99.9
12	7202	99.4	7167	99.0
13	6810	94.0	6845	94.5
14	7148	98.7	7120	98.3
15	6944	95.9	6922	95.6
16	6903	95.3	6974	96.3
17	6629	91.5	6623	91.5
18	6863	94.8	6987	96.5
19	7242	100.0	7242	100.0
20	7242	100.0	7242	100.0
21	6863	94.8	6987	96.5
22	6658	91.9	6738	93.0
23	6682	92.3	6811	94.0
24	6542	90.3	6578	90.8
25	6729	92.9	6781	93.6
26	7242	100.0	7242	100.0

7. Conclusions

The relative performance of ID3 and Backpropagation on the text-to-speech task depends on the decoding technique employed to convert the 26 bits of the Sejnowski/Rosenberg code into phoneme/stress pairs. Decoding to the nearest legal phoneme/stress pair (the most obvious approach) reveals a substantial difference in the performance of the two algorithms.

Experiments investigated three hypotheses concerning the cause of this performance difference.

The first hypothesis—that ID3 was overfitting the training data—was shown to be incorrect. Three techniques that avoid overfitting were applied, and none of them improved ID3's performance.

The second hypothesis—that the ability of backpropagation to share hidden units was a factor—was shown to be only partially correct. Sharing of hidden units does improve the classification performance of backpropagation and—perhaps more importantly—the convergence of the gradient descent search. However, an analysis of the kinds of errors made by ID3 and backpropagation (with or without shared hidden units) demonstrated that these were different kinds of errors. Hence, eliminating shared hidden units does not produce an algorithm that behaves like ID3. This suggests that the development of a "shared ID3" algorithm that could learn multiple concepts simultaneously is unlikely to produce performance similar to BPCV.

The third hypothesis—that backpropagation was capturing statistical information by some mechanism (perhaps the continuous output activations)—was demonstrated to be the primary difference between ID3 and BPCV. By adding the "observed" decoding technique to both algorithms, the level of performance of the two algorithms in classifying test cases becomes statistically indistinguishable (at the word and phoneme levels). By adding the "block" decoding technique, all differences between the algorithms are statistically insignificant.

Given that with block decoding the two algorithms perform equivalently, and given that BPCV is much more awkward to apply and time-consuming to train, these results suggest that in tasks similar to the text-to-speech task, ID3 with block decoding is clearly the algorithm of choice. For other applications of BPCV, ID3 can play an extremely valuable role in exploratory studies to determine good sets of features and predict the difficulty of learning tasks.

This paper has also introduced a new method of experimental analysis that computes error correlations to measure the effect of algorithm modifications. We have shown that this method can be applied to discover the ways in which algorithms are related. Broader application of this methodology should improve our understanding of the assumptions and biases underlying many inductive learning algorithms.

8. Acknowledgements

The authors thank Terry Sejnowski for providing the NETtalk phonemic dictionary, without which this work would have been impossible. Correspondence with Jude Shavlik, Ray Mooney, and Geoffrey Towell helped clarify the possible kinds of decoding strategies. Discussions with Lorien Pratt aided in the design of the cross-validation studies. This research was supported by NSF grant numbers CCR-87-16748 and IRI-86-57316 (Presidential Young Investigator Award) with matching support from SUN Microsystems.

9. References

Bakiri, G. (1991). *Converting English text to speech: A machine learning approach.* Doctoral Dissertation (Technical Report 91-30-1). Corvallis, OR: Oregon State University, Department of Computer Science.

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

Buntine, W. (1990). *A theory of learning classification rules.* Doctoral dissertation. School of Computing Science, University of Technology, Sydney, Australia.

Dietterich, T. G. (1989). Limits of inductive learning. In *Proceedings of the Sixth International Conference on Machine Learning* (pp. 124–128). Ithaca, NY. San Mateo, CA: Morgan Kaufmann.

Dietterich, T. G., & Bakiri, G. (1991). Error-correcting output codes: A general method for improving multiclass inductive learning programs. *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, CA: AAAI Press.

Dietterich, T. G., Hild, H., & Bakiri, G. (1990). A comparative study of ID3 and backpropagation for English text-to-speech mapping. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 24–31). Austin, TX: Morgan Kaufmann.

Klatt, D. (1987). Review of text-to-speech conversion for English. *J. Acoust. Soc. Am., 82*, 737–793.

Lucassen, J. M., & Mercer, R. L. (1984). An information theoretic approach to the automatic determination of phonemic base forms. *Proc. Int. Conf. Acoust. Speech Signal Process.* ICASSP-84, 42.5.1–42.5.4.

Lang, K. J., Waibel, A. H., & Hinton, G. E. (1990). A time-delay neural network architecture for isolated word recognition. *Neural Networks, 3*, 33-43.

Martin, G. L., & Pittman, J. A. (1990). Recognizing hand-printed letters and digits. In D. Touretzky (Ed.), *Advances in Neural Information Processing Systems 2*, 405–414. San Mateo, CA: Morgan Kaufmann.

McClelland, J. L., & Rumelhart, D. E. (1988). *Explorations in parallel distributed processing,* Cambridge, MA: MIT Press.

Mingers, J. (1989). An empirical comparison of pruning methods for decision tree induction. *Machine Learning, 4*, 227–243.

Mooney, R., Shavlik, J., Towell, G., & Gove, A. (1989). An experimental comparison of symbolic and connectionist learning algorithms. *IJCAI-89: Eleventh International Joint Conference on Artificial Intelligence*, (pp. 775–80).

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess endgames. In R. S. Michalski, J. Carbonell, & T. M. Mitchell, (eds.), *Machine learning: An artificial intelligence approach, 1*, San Mateo, CA: Morgan Kaufmann.

Quinlan, J. R. (1986a). The effect of noise on concept learning. In R. S. Michalski, J. Carbonell, & T. M. Mitchell, (eds.), *Machine learning: An artificial intelligence approach, 1*, San Mateo, CA: Morgan Kaufmann.

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

Quinlan, J. R. (1987). Simplifying decision trees. *International Journal of Man-Machine Studies, 27*, 221–234.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, & J. L. McClelland (Eds.), *Parallel distributed processing*, (Vol 1). Cambridge, MA: MIT Press.

Rosenberg, C. R. (1988). *Learning the connection between spelling and sound: A network model of oral reading.* Doctoral Dissertation. (CSL Report 18). Princeton, NJ: Princeton University, Cognitive Science Laboratory.

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

Touretzky, D. S. (Ed.) (1989). *Advances in neural information processing systems 1.* San Mateo, CA: Morgan Kaufmann.

Touretzky, D. S. (Ed.) (1990). *Advances in neural information processing systems 2.* San Mateo, CA: Morgan Kaufmann.

Appendix

A.1. The 26-bit Code for Phoneme/Stress Pairs

Sejnowski and Rosenberg developed the following distributed code for representing the phonemes and stresses. The examples were supplied with their database.

Phoneme Code

Phoneme	Codeword	Examples
/a/	000010000000100100000	wAd, dOt, Odd
/b/	000100000001010000000	Bad
/c/	000001000000000010000	Or, cAUght
/d/	100000000001010000000	aDd
/e/	010000000000100010000	Angel, blAde, wAy
/f/	000100010000000000000	Farm
/g/	000001000001010000000	Gap
/h/	001000001000000000000	Hot, WHo
/i/	000100000000101000000	Eve, bEe
/k/	000001000001000000000	Cab, Keep
/l/	010000000100010000000	Lad
/m/	000100000010010000000	Man, iMp
/n/	100000000010010000000	GNat, aNd
/o/	001000000000100010000	Only, Own
/p/	000100000001000000000	Pad, aPt
/r/	000010000100010000000	Rap
/s/	100000010000000000000	Cent, aSk
/t/	100000000001000000000	Tab
/u/	001000000000101000000	bOOt, OOze, yOU
/v/	000100010000010000000	Vat
/w/	000100001000010000000	We, liqUid
/x/	000010000000000010000	pirAte, welcOme
/y/	000010001000010000000	Yes, senIor
/z/	100000010000010000000	Zoo, goeS
/A/	110000000000100010000	Ice, hEIght, EYe
/C/	000010100000000000000	CHart, Cello
/D/	010000010000010000000	THe, moTHer
/E/	010100000000000010000	mAny, End, hEAd
/G/	000001000010010000000	leNGth, loNG, baNk
/I/	000100000000001000000	gIve, bUsy, captAIn
/J/	000010100000010000000	Jam, Gem
/K/	000011110000000000000	aNXious, seXual
/L/	100000000100010000000	eviL, abLe
/M/	010000000010010000000	chasM
/N/	000010000010010000000	shorteN, basiN
/O/	100010000000100010000	OIl, bOY

Phoneme Code

Phoneme	Codeword	Examples
/Q/	000101100001010000000	Quilt
/R/	000001000100010000000	honeR, afteR, satyR
/S/	000010010000000000000	oCean, wiSH
/T/	010000010000000000000	THaw, baTH
/U/	000001000000001000000	wOOd, cOUld, pUt
/W/	000011000000101010000	oUT, toWel, hoUse
/X/	110000100000000000000	miXture, anneX
/Y/	110100000000101000000	Use, fEUd, nEw
/Z/	000010010000010000000	uSual, viSion
/@/	010000000000000100000	cAb, plAId
/!/	010100100000000000000	naZi, piZZa
/#/	000011100000010000000	auXiliary, eXist
/*/	100100001000010100000	WHat
/^/	100000000000000100000	Up, sOn, blOOd
/+/	000000000000000000000	abattOIr, mademOIselle
/-/	000000000000000001001	silence
/_/	000000000000000001010	word-boundary
/./	000000000000000000110	period

Here are the meanings of the individual bit positions:

Bit Position	Meaning
1	Alveolar = Central1
2	Dental = Front2
3	Glottal = Back2
4	Labial = Front1
5	Palatal = Central2
6	Velar = Back1
7	Affricative
8	Fricative
9	Glide
10	Liquid
11	Nasal
12	Stop
13	Tensed
14	Voiced
15	High
16	Low
17	Medium
18	Elide
19	FullStop
20	Pause
21	Silent

The stress code actually encodes syllable boundary information as well as stresses.

Stress Code

Stress	Codeword	Meaning
`<`	10000	a consonant or vowel following the first vowel of the syllable nucleus.
`>`	01000	a consonant prior to a syllable nucleus.
`0`	00010	the first vowel in the nucleus of an unstressed syllable.
`2`	00100	the first vowel in the nucleus of a syllable receiving secondary stress.
`1`	00110	the first vowel in the nucleus of a syllable receiving primary stress.
`-`	11001	silence

A.2. Replication of Results on Four Additional Data Sets

To simplify the presentation in the body of the paper, we presented data only for one choice of training and test sets. This appendix provides that same data on all five training and testing sets to demonstrate that the results hold in general.

A.2.1. Performance of ID3 and BP under legal decoding

Table 1 shows the performance, under legal decoding, of ID3 and BP when trained on each of the 5 training sets and tested on the corresponding test sets.

Table 1. Percent correct over 1000-word test set

		Level of Aggregation (% correct)				
Data Set	Method	Word	Letter	Phoneme	Stress	Bit (mean)
a	ID3	9.6	65.6	78.7	77.2	96.1
	BPCV	13.6**	70.6***	80.8***	81.3***	96.7*
b	ID3	10.4	65.6	79.6	76.4	96.1
	BPCV	15.7***	71.5***	81.7***	81.4***	96.7*
c	ID3	10.5	64.4	78.9	75.7	96.0
	BPCV	15.2***	71.4***	81.4***	81.7***	96.7*
d	ID3	10.9	65.8	80.0	76.2	96.2
	BPCV	16.3***	71.3***	81.4*	81.6***	96.7*
e	ID3	9.5	64.7	78.2	77.1	96.0
	BPCV	14.5**	71.6***	81.3***	82.3***	96.7*

Difference in the cell significant at $p < .05$*, .005**, .001***

A.2.2. Tests of the Sharing Hypothesis

For replications b, c, d, and e, the training procedure for each of the 26 separate networks was slightly different from the procedure described for replication a. Starting with $H = 1$, a network with H hidden units was trained for 1000 epochs. If this did not attain the desired fit with the data, the next larger value for H was tried. If a network with 5 hidden units failed to fit the data, the process was repeated, starting again with $H = 1$ and a new randomly-initialized network. No network required more than 4 hidden units. Table 2 shows the observed performance differences. The training figures show that, with the exception of word-level and stress-level performance, the 26 separate nets fit the training data slightly better than the single 120-hidden-unit network.

Table 2. Performance difference (in percentage points) between a single 120-hidden unit network and 26 separate networks. Trained on 50 words and tested on 1000 words.

		Level of Aggregation (% point differences)				
Replication	Data set	Word	Letter	Phoneme	Stress	Bit (mean)
a	TRAIN:	0.0	−0.2	−0.2	0.0	0.0
	TEST:	0.2	3.4	2.8	1.8	0.6
b	TRAIN:	4.0	0.1	−0.2	−0.5	−0.1
	TEST:	0.6	3.4	2.9	0.8	−0.2
c	TRAIN:	4.0	0.0	0.0	0.0	−0.1
	TEST:	1.1	3.0	2.2	2.9	0.0
d	TRAIN:	0.0	−0.6	−0.5	0.0	−0.1
	TEST:	0.3	3.7	2.8	2.1	0.0
e	TRAIN:	4.0	−0.5	−0.6	0.0	−0.1
	TEST:	0.7	2.7	2.1	1.6	−0.1
Averages	TRAIN:	2.4	−0.2	−0.2	0.1	−0.1
	TEST:	0.6	3.2	2.6	1.8	0.1

A.3. Frequency Information for Phoneme/Stress Pairs Observed in the 1000-Word Training Set

The following table lists each phoneme/stress pair observed in the 1000-word training set, and shows how many times it appears in that training set.

Table 3. Phoneme/stress pairs observed in the 1000-word training set.

Phoneme	Stress	Count	Phoneme	Stress	Count
a	<	3	p	<	58
a	0	15	p	>	165
a	1	97	r	<	139
a	2	21	r	>	230
b	<	32	s	<	158
b	>	121	s	>	188
c	<	2	t	<	249
c	0	2	t	>	173
c	1	39	t	0	1
c	2	15	u	<	2
d	<	99	u	>	1
d	>	97	u	0	8
e	<	3	u	1	30
e	0	4	u	2	5
e	1	86	v	<	39
e	2	38	v	>	53
f	<	17	w	>	37
f	>	81	w	0	4
g	<	26	w	1	10
g	>	57	w	2	3
h	>	31	x	<	57
i	<	2	x	>	4
i	0	144	x	0	515
i	1	51	x	1	3
i	2	16	x	2	17
k	<	114	y	>	5
k	>	201	y	0	6
l	<	108	y	1	2
l	>	163	z	<	48
m	<	95	z	>	9
m	>	107	A	<	2
n	<	382	A	0	12
n	>	81	A	1	45
o	0	17	A	2	31
o	1	67	C	<	13
o	2	26	C	>	22

Phoneme	Stress	Count
D	<	7
D	>	2
E	<	2
E	>	1
E	0	3
E	1	138
E	2	35
G	<	37
I	<	5
I	0	151
I	1	133
I	2	43
J	<	21
J	>	40
K	<	1
L	<	54
L	>	38
M	0	11
N	<	14
O	1	6
O	2	2
R	<	119
S	<	27
S	>	49
T	<	5
T	>	18
U	0	5
U	1	12
U	2	7
W	0	1
W	1	17
W	2	4
X	<	14
Y	0	20
Y	1	25
Y	2	8
Z	<	6
Z	>	5
@	<	1
@	0	15
@	1	164
@	2	35
!	>	1
#	<	1
*	>	4
^	0	1
^	1	54
^	2	4
+	1	1
-	<	542
-	>	233
-	0	241
-	1	34
-	2	8

Notes

1. A target threshold is an output activation value that is considered to be correct even though the output activation does not equal the desired activation. For example, if the target thresholds are .1 and .9, then an output activation of .1 or below is considered correct (if the desired output is 0.0) and an activation of .9 or above is considered correct (if the desired output is 1.0).
2. It would have been better if we had stored a snapshot of the random starting network before beginning training, but we failed to do this. Nevertheless, the procedure we followed is still safe, because it obeys the rule that no information from the test set should be used during training.