Performance Comparison between Human Engineered & Machine Learned Letter-to-Sound Rules for English: A Machine Learning Success Story.

Ghulum Bakiri

Department of Computer Science
University of Bahrain
P. O. Box 32038, Isa Town
State of Bahrain.

Thomas G. Dietterich

Department of Computer Science
Oregon State University
Corvallis, OR 97331-3102

Abstract

The task of mapping spelled English words into strings of phonemes and stresses ("reading aloud") has many practical applications. Several commercial systems perform this task by applying a knowledge base of expert-supplied letter-to-sound rules. This paper presents a set of machine learning methods for automatically constructing letter-to-sound rules by analyzing a dictionary of words and their pronunciations. Our results demonstrate that these methods, taken together, provide a substantial performance improvement over the best commercial system—DECtalk from Digital Equipment Corporation—-that utilizes hand-crafted letter-to-sound rules. The results are significant since these machine learning techniques are general and can be applied to many other tasks including the task of discovering pronunciation rules in other languages, there-by eliminating the need for hand-crafting a rule-base for each language.

1 Overview

The automatic conversion of English text to synthetic speech is an important task that has a wide range of practical applications. One particularly difficult step in this conversion process involves mapping words (i.e strings of letters) into strings of phonemes and stresses. This mapping is typically performed (in commercial devices such as DECtalk) by a large rule-based system that is hand-crafted to handle the "regular" conversions, while an exception dictionary is relied upon to take care of those irregular words that defy the rule base. In a typical system, words are first stripped of affixes (prefixes and suffixes) then one set of rules are applied to map them to phonemes. Then, the words are augmented with stresses using another set of rules to determine the stresses. Finally, a third set of rules are applied to modify the pronunciations while reattaching the affixes.

All these rules must be hand-cratfed by domain experts requiring considerable time and effort, which makes the process rather expensive. There are complex and numerous rules for each of the above steps. The interested reader may consult some of the references. Klatt (1979, 82, and 87) is an excellent starting point. Suffice it to say that mapping vowel strings requires some 400 rules, 14 for: "ea" alone!

1.1 An alternative Machine Learning approach

In this paper, we will explore an alternative approach that utilizes machine learning techniques to automatically build a "rule base" suitable for performing the mapping of isolated English words onto strings of phonemes and stresses.

After a brief introduction of the task of learning discrete multi-valued functions from examples (multiclass learning), we will go through the procedure we followed to cast the text-to-speech mapping task in the framework of (multiclass) learning from examples. We wil then describe a base configuration for the learning system and show the performance of this system to be used as a reference for performance comparisons. Following that, we describe the steps we followed to incrementally develop the "best" configuration of the learning system for this task. We then compare the performance of this learning system to that of DECtalk's rule-base using an objective—but strict—correctness criteria that requires an exact match of both the phoneme and stress strings to consider a pronunciation correct. Finally, we present the results of a more subjective comparative study between the two systems, one in which human subjects are asked to judge how close the pronunciations output by each system are to the correct (dictionary) pronunciations.

2 Learning from Examples

The Goal of the Machine Learning Approach in this context is to discover pronunciation rules automaically from a large number of examples of words and their pronunciations. This falls under the general task of Learning from Examples whereby we are required to find an approximate definition for an unknown function $f(\vec{x})$ given training examples of the form $\langle \vec{x}_i, f(\vec{x}_i) \rangle$. For cases in which f takes only the values $\{0, 1\}$—binary functions—there are many algorithms available. For example, the decision tree methods, such as ID3 (Quinlan, 1986b) and CART (Breiman, Friedman, Olshen & Stone, 1984) can construct trees whose leaves are labelled with binary values. Most artificial neural network algorithms, such as the perceptron algorithm (Rosenblatt, 1958) and the error backpropagation (BP) algorithm (Rumelhart, Hinton & Williams, 1986), are best suited to learning binary functions. Theoretical studies of learning have focused almost entirely on learning binary functions (Valiant, 1984; COLT 1988, 1989, 1990).

In many practical domains, however, the unknown function f takes on values from a discrete set of so-called "classes": $\{c_1, \ldots, c_k\}$. For example, in medical diagnosis, the function might map a description of a patient to one of k possible diseases. In digit recognition (LeCun, et al., 1989), the function maps each hand-printed digit to one of $k = 10$ classes. (We will see shortly that our domain of interest - the text-to-speech mapping domain - falls under this category of Learning from Examples.)

There are several methods for handling these "multiclass" learning problems.

The one that we will present in this paper takes the approach of converting the multiclass learning problem to a collection of *binary* learning problems by employing a *distributed output code*. Each class is assigned a unique binary string of length n; we will refer to these as "codewords." Then n binary functions are learned, one for each bit position in these binary strings. During training for an example from class i, the desired outputs of these n binary functions are specified by the codeword for class i. With decision-tree algorithms, n (binary) decision trees are built, one for each bit of the codewords. With artificial neural networks, these n functions can be implemented by the n output units of a single network.

New values of $\vec{x}$ are classified by evaluating each of the n binary functions to generate an n-bit string $\vec{s}$. This string is then compared to each of the k codewords, and $\vec{x}$ is assigned to the class whose codeword is closest, according to some distance measure, to the generated string $\vec{s}$.

We have tried several other methods for handling this multiclass learning problem, but the above method gave the best performance in our experiments in this domain. (See Bakiri, G. (1991a), for example, for more elaboration on th is and other methods of handling multiclass learning problems.)

2.1 Source of examples: The NETtalk dictionary

Sejnowski and Rosenberg provided us with a dictionary of 20,002 words and their corresponding phoneme and stress strings. From this dictionary we drew at random (and without replacement) a testing set of 1000 words and used the remaining 19,002 words as a training set.

There were 54 phonemes and 6 stresses in Sejnowski and Rosenberg's (1987) NETtalk formulation of this task. The 6 stresses encoded in addition to the 3 stress symbols for "primary", "secondary" and "no" stress—3 additional symbols encoding word and syllable boundary information. We *simplified* the stress information by eliminateing these additional symbols and replacing them with the "no" stress symbol for compatibility with outputs from DECtalk's rule-base. Hence, the task of learning:

("`aback`" "`xb@k-`" "`0>1<<`")

where `xb@k-` is a string of phonemes, and `0>1<<` is a string of stress symbols, was transformed to:

(“`aback`” “`xb@k-`” “`--1--`”)

after simplifying the stresses. Note that the phonemes and stresses are aligned with the letters of the original word.

2.1.1 Decomposing the Dictionary Entries into Learning Examples

Each entry in the dictionary corresponding to a k-letter word is converted to k examples; one for each letter of the word. For each letter L in the word, a 7-letter window is created consisting of L at the center, the 3-letters preceding L and the three letters following L. The character “`-`” (symbol for space) is used if there are not enough neighbors on either side of the letter L to complete the number of letters on each side to be 3. The 7-letter windows play the part of the feature-vector (after converting them to a binary encoding). The output is the pair of phoneme and stress symbols that L maps to in this particular word.
As an example, the following entry in the dictionary

(“`aback`” “`xb@k-`” “`--1--`”)

will generate the following examples :

```
("---abac"  x  -)
("--aback"  b  -)
("-aback-"  @  1)
("aback--"  k  -)
("back---"  -  -)
```

2.2 The Learning Task

With the above transformations, our learning task is to learn the mappings (functions) that map from an individual letter (L_0)—and some context around it —to a phoneme/stress pair representing the pronunciation ($P_0 S_0$) of L_0. Sejnowski and Rosenberg, for example, employed 3-letters on each side of L_0 to depict the context as shown above, thereby mapping a seven-letter window (with L_0 at the center) to a single phoneme and a single stress:

$$f(L_{+1}, L_{+2}, L_{+3}, L_0, L_{-3}, L_{-2}, L_{-1}) = P_0 S_0.$$

We employed a similar approach, but allowed the window size to vary from 7 upto 15 letters in our experements. We also included additional information as part of the context, and varied the binary representations employed to convert the context to binary in some experiments.

2.3 Binary Input and Output Representations

The input and output representation schemes introduced by Sejnowski and Rosenberg for the seven-letter windows will be used as a "base configuration" for guaging the effectiveness of all subsequent modifications. In this scheme, the 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 (i.e., 203 binary features) for each window. Experiments by Shavlik, Mooney, and Towell (1990) showed that this representation was better than treating each letter in the window as a single feature with 29 possible values.

On the output side, Sejnowski and Rosenberg encoded 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 f 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 126 bit strings representing observed 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.) Ties were broken in favor of the phoneme/stress pair that appeared more frequently in the training data. In Dietterich, Hild, and Bakiri (1990a), we called this "observed decoding."

2.4 Learning the Individual Bits: The ID3 Algorithm

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 & Stone (1984) and to the tree-growing method developed by Lucassen & Mercer (1984).

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). Experiments in Dietterich, Hild, and Bakiri (1990b) showed that these pruning methods did not improve performance.

We did apply one simple kind of forward pruning to handle inconsistencies in the training data: If all training examples agreed on the value of the chosen feature, 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). Note that if all of the examples agree on the value of this feature, then the feature has zero mutual information with the output class (and hence, since this was the *best* feature, every other feature must

Table 1: Performance of the base configuration described on the standard training and testing data sets.

Data set used for evaluation	% correct (1000-word data set) Level of Aggregation					Decision Tree data (mean)	
	Word	Letter	Phoneme	Stress	Bit (mean)	Leaves	Depth
Training set	96.6	99.5	99.8	99.6	100.0	269.9	29.3
Test set	12.5	69.6	81.3	79.2	96.3	269.9	29.3

also have zero mutual information).

With the Sejnowski and Rosenberg 26-bit distributed output code, ID3 is applied 26 times during learning to construct 26 binary decision trees, one for each bit-position in the output code.

3 Performance Results

In this section, we will present some of the intermideate results of an extensive "hill climbing" search that we undertook to find the "best" configuration for the learning method— in preparation for a comparison with DECtalk. For the experiments conducted as part of this search, the learning systems were trained on our standard 1000-word training set, and the testing was done on an *alternate test set*—disjoint from both the training set and our standard test set.

The result of testing the best configurations obtained as a result of this search—but trained on 19,002 words—will be postponed until Section 4. There, we will present these results and compare them to the performance of DECtalk's rule-base.

3.1 Base Configuration

Table 1 shows the percent correct (over the 1000-word test set) for words, letters, phonemes, and stresses for the "base configuration" described earlier. A word is classified correctly if each letter in the word is correct. A letter is correct if the phoneme and stress assigned to that letter are both correct. For the distributed output code method employed, the phoneme is correct if all bits coding for the phoneme are correct (after mapping to the nearest legal codeword and breaking ties by frequency). Similarly, the stress is correct if all bits coding for the stress are correct. Also shown are the average number of leaves contained in all the decision trees built by ID3 for the phoneme and stress bits, and the average maximum depth of all the trees.

There are several things to note in Table 1. First, the performance of the learning system on the training set is uninteresting. Decision tree building algorithms such as ID3 always perform extremely well on the training set. In fact, had it not been for the presence of some inconsistencies in the training set, all the performance figures shown in the first row of Table 1 would have been 100% . For this reason, we will not report the performance of our learning systems on the training set in the rest of this paper. Second, due to the aggregation problem, performance at higher levels of aggregation (e.g. the word level) is disappointingly low when compared with the performance at lower levels of aggregation such as the stress or phoneme level. Another thing to observe is that the bit-level performance of 96.6% is somewhat misleading since only 3 to 5 bits in the 21-bit phoneme vector are set to '1'. Hence, guessing constantly '0' will already achieve a correctness higher than 85%.

3.2 Improved Output Representation: Error Correcting Codes

In Bakiri, G., & Dietterich, T. G. (1991), we show that a principled method for designing good distributed output codes is to employ error-correcting codes. Good error-correcting codes choose the individual code words so that they are well-separated in Hamming distance. This allows the system to recover from errors made in learning the individual binary functions. For example, if the minimum inter-word Hamming distance between codewords, d is chosen as 7, then we can recover any codeword even if up to 3 bits are corrupted. In general, if the codewords have a minimum distance d, then any $\lfloor (d-1)/2 \rfloor$ errors can be corrected. Error correcting codes have long been used in the communications field, but we believe, this is the first time they are being used in machine learning applications.

There are conflicting intuitions with regard to employing error-correcting codes as output representation in Machine Learning applications. For one thing, the columns which represent the individual functions being learned are meaningless concepts in the domain. Hence, the individual f_i will be harder to learn. On the other hand, errors will be corrected. Hence, the individual f_i need not be learned as accurately to obtain good performance. Another obstacle in the use of error correcting codes output representations is that errors will be correlated. Thus, even though arbitrarily long error-correcting codes are expected to work arbitrarily well with uncorrelated errors, a point of diminishing returns should be encountered with correlated errors.

Tables 2 and 3 show the performance (on our 1000-word test-set) of several *efficient*[1] BCH error-correcting codes of varying lengths. Columns headed n show the length of the code, and columns headed d show the minimum Hamming distance between any two codewords. Except for output representation, *all* other details were the same as our base configuration. In table 2, the phoneme and stresses were encoded separately each with its own code, and the two resulting codes were concatenated. In table 3, each phoneme/stress pair was treated as one entity and was assigned a unique codeword. We refer to the former case as the separate phoneme/stress case, and to the latter as the combined-ps (combined phoneme/stress) case.

The first thing to note is that the performance of even the simplest (19-bit) BCH code is superior to the 26-bit Sejnowski-Rosenberg code at the letter and word levels. Better still, performance improves monotonically as the length (and error-correcting power) of the code increases.

Not surprisingly, the individual bits of these error-correcting codes are much more difficult to learn than the bits in the Sejnowski-Rosenberg distributed code. Specifically, the average number of leaves in each tree in the error-correcting codes is roughly 665, whereas the

[1] We consider an error-correcting code to be *efficient* if the mininum inter-word Hamming distance, d, is roughly equal to half the length of the code, n. This enables the code to correct errors in t bits, where t is approximately one-fourth the length n.

Table 2: Performance (% correct on 1000-word test set) of various BCH codes. n is the codeword length and d is the minimum inter-word Hamming distance between all codewords.

BCH Code Data				% correct (1000-wod test set)				Decision Tree data (mean)	
Phoneme		Stress		Level of Aggregation					
n	d	n	d	Word	Letter	Phon.	Stress	Leaves	Depth
10	3	9	3	13.3	69.8	80.3	80.6	677.4	51.9
14	5	11	5	14.4	70.9	82.3	80.3	684.7	53.1
21	7	13	7	17.2	72.2	83.9	80.4	681.4	53.9
26	11	13	11	17.5	72.3	84.2	80.4	700.5	56.4
31	15	30	15	19.9	73.8	84.8	81.5	667.8	52.7
62	31	30	15	20.6	74.1	85.4	81.6	669.9	53.3
127	63	30	15	20.8	74.4	85.7	81.6	661.6	54.8
Base Configuraion:				12.5	69.6	81.3	79.2	269.9	29.3

Table 3: Performance of error-correcting codes applied to the combined phoneme/stress case.

Code Length n	Interword Distance d (min.)	% correct (1000-word test set)				Decision Tree data (mean)	
		Level of Aggregation					
		Word	Letter	Phon.	Stress	Leaves	Depth
63	31	20.3	74.3	83.8	80.3	1000.0	68.7
127	63	22.3	75.5	85.2	81.5	1002.9	68.3
255	127	22.4	75.5	85.2	81.6	1014.4	68.8

Sejnowski-Rosenberg trees had 270 leaves. Clearly distributed output codes do not produce results that are easy to understand!

The fact that performance continues to improve as the code gets longer suggests that we could obtain arbitrarily good performance if we used arbitrarily long codes. Indeed, Shannon's [Shannon48] theorem in information theory proves this under the assumption that the errors in the various bit positions are independent. Unfortunately, this is not the case in practice. In [Bakiri91a], we discuss these and other limitations of the error-correcting output coding approach and suggest possible means of overcoming some of these limitations.

3.3 Extending the Context

The basic idea here is to include the pronunciations of the letters already processed— as part of the context for the current-letter. Hence for a 7-letter window, the learning task is now the mapping function f defined as follows:

$$f(L_{-3}, L_{-2}, L_{-1}, L_0, L_{+1}, L_{+2}, L_{+3}, P_{-3}, S_{-3}, P_{-2}, S_{-2}, P_{-1}, S_{-1}) = P_0 S_0$$

where the $P_i S_i$ are the phoneme and stresses (pronunciations) determined for the previous letters L_i. ($i = -1, -2$ and -3).

This will not present any problems during the classification, since the pronunciations of the letters already processed will be available by the time we are ready to classify the current letter. In addition, it will introduce two new degrees of freedom:

1. Forward vs. Backward processing of the letters of the words.
2. Binary codes for converting the pronunciations of the letters already processed to binary.

Table 4 shows the performance figures when several schemes for converting the extended context to binary representation are employed. Detailed explanation of each of the schemes shown (and several others) can be found in Bakiri, G. (1991a). Briefly, in **ExtContext(2)**, weight-1 codes are employed throughout for *all* the elements of the extended context:

- All the letters in the context are converted to binary using the standard 29-bit weight-1 (local) code as usual.
- The three left phonemes are converted to binary by employing a 54-bit wieght-1 code corresponding to the 54 phonemes. The codeword for $phoneme_i$ will have all, but the i^{th} bit, set to 0.
- The three left stresses are converted to binary by employing a 6-bit wieght-1 code corresponding to the 6 stresses. The codeword for $stress_i$ will have all, but the i^{th} bit, set to 0.

ExtContext(7) is similar to ExtContext(2). Additionally:

- One bit is added to the code for each phoneme in the extended context, to produce a 55-bit code:

Table 4: Impact of alternative schemes for converting the extended context to binary represenation. Results are shown for both sequences of processing of the letters of the words: forward and backwards

Input Representation	Order of letter processing	% correct (1000-word test set) Level of Aggregation: Word	Letter	Phoneme	Stress	Bit	Decision Tree data (mean): Leaves	Depth
ExtContext(2)	Forward	15.2	66.2	78.5	76.3	95.8	225.0	27.1
	Backward	19.7	71.5	82.2	80.7	96.4	198.1	23. 8
ExtContext(7)	Forward	17.0	67.9	81.2	77.1	96.2	207.2	24.2
	Backward	24.4	74.2	83.9	82.6	96.9	180.1	22. 0
Base Configuration		12.5	69.6	81.3	79.2	96.3	269.9	29. 3

Bit_{55} is "true" for phonemes (a e i o u A O W Y)
Bit_{55} is "false" for all other phonemes.

This bit encodes whether the vowel (phoneme) is "tense" or "lax" ("nontense "). Linguists frequently make this distinction in their rules for English stress.

- A 30^{th} bit is added to the code for each letter or input symbol to distinguish vowels from consonants, and,
- A 31^{st} bit is added to the code for each letter or input symbol:

  Bit_{31} is "true" for (E I O U)
  Bit_{31} is "false" for all other letters (or input symbols).

  This bit distinguishes the "non-low vowels" from all others.
- An extra bit is also added to the code for each stress symbol in the extended context to indicate whether the letter is stressed or not.

Performance for both forward and backward processing of the letters of the words is shown. Except for the extended context, all other details are the same as the base configuration. The results unambiguously indicate that the extended context gives far superior performance than the standard context when processing the letters of the word backward— in a right-t o-left manner. On the other hand, the extended context slightly degrades the performance with the "normal" left-to-right (forward) processing. Performance improvement is observed at *all* levels of aggregation and for *all* the schemes employed to convert the extended context to binary. Furthermore, the decision trees produced as a result of the backward processing are smaller than their counterparts produced with normal processing. This indicates that the *right* phonetic context is more relevant than the left phonetic context if we want to learn rules for the pronunciation of the current letter. This finding is consistent with the fact that several researchers working on predicting English word-stress have come up with rules that refer to certain properties of the syllables of the words taken in a right-to-left order. An example of such rule is (see Halle, M. & Keyser S. J. (1971)) for a complete list):

If the last vowel is nontense, primary stress goes on the antepenultimate vowel when the penultimate vowel is nontense and followed by no more than one consonant.

Other interesting points revealed by the results of Table 4 (besides the obvious fact that input representation has a profound effect on the performance of learning systems) are:

1. Even-though including the *left* phonetic context (when processing the letters of the words forward, i.e. from left-to-right) *degrades* the performance at the bit, stress, phoneme and letter levels both schemes employed to convert the extended context to binary, it consistently improves the word level performance.

2. Including the left phonetic context results invariably in smaller trees compared to the base configuration.

3. ExtContext(7) nearly compensates for the negative effect of the left phonetic context as far as the phoneme performance is concerned.

4. The combination of extended context, backward processing of the letters of the words and improved binary represenation for the extended context have now raised the performance at *all* levels of aggregation when compared to the base configuration. Hence, ExtContext(7), with words processed *backwards*, (or "ExtContext(7-BW)" for short) will be adopted as our "best" input configuration.

3.4 Error-correcting Codes & Extended Context

In the previous two subsections we have seen improvements in performance as a result of extending the context with backward processing of the letters and improved binary representation of the extended context ("ExtContext(2-BW)" and "ExtContext(7-BW)"). These are all on the input side. We also saw a drastic performance gain by using error-correcting codes on the output side.

An interesting question is whether the benefits provided by error-correcting output codes are independent of the benefits provided by "ExtContext(2-BW)" or "ExtContext(7-BW)". The answer, as shown in Table 5 (for the separate-ps case) and Table 6 (for the combined-ps case), is unambiguously "yes." The combination of error-correcting output codes with these improved input techniques provides the best performing text-to-speech system that we have studied.

3.5 Effect of Window Size

In all the experiments reported so far, we have been employing a 7-letter window. In this section, we will explore the effect of enlarging the window size on the performance of one configuration that employ error-correcting distributed output codes, "ExtContext(7-BW)" and the *combined* phoneme/stress approach. The performance figures for these configurations and two window sizes (7 and 13) are compared in Table 7. These results indicate that the phoneme performance degrades slightly, while the stress performance benefits from the larger window size. However, the improved stress performance is more than enough to offset the

Table 5: Performance of various BCH error-correcting codes with and without extended context and right-to-left (BW) processing of the letters of the words. See text for a precise meaning of "ExtContext(2-BW)" and "ExtContext(7-BW)". Phonemes and stresses are coded separately (the separate-ps case).

BCH Code Data				Input	% correct (1000-wod test set)				Decision Tree data (mean)	
Phoneme		Stress		Representation	Level of Aggregation					
n	d	n	d		Word	Letter	Phon.	Stress	Leaves	Depth
10	3	9	3	StdContext	13.3	69.8	80.3	80.6	677.4	51.9
				ExtContext(2-BW)	24.3	72.9	82.0	82.3	476.4	40.1
				ExtContext(7-BW)	26.6	74.8	83.4	83.7	431.0	32.0
14	5	11	5	StdContext	14.4	70.9	82.3	80.3	684.7	53.1
				ExtContext(2-BW)	24.3	73.2	83.1	81.6	503.8	42.8
				ExtContext(7-BW)	27.2	75.2	84.6	83.3	453.1	34.0
21	7	13	7	StdContext	17.2	72.2	83.9	80.4	681.4	53.9
				ExtContext(2-BW)	26.1	74.8	84.9	82.4	494.4	39.7
				ExtContext(7-BW)	28.9	76.6	86.0	83.9	453.3	37.1
26	11	13	11	StdContext	17.5	72.3	84.2	80.4	700.5	56.4
				ExtContext(2-BW)	27.0	74.5	85.1	82.0	504.9	41.7
				ExtContext(7-BW)	29.3	76.4	86.2	83.7	456.3	36.8
31	15	30	15	StdContext	19.9	73.8	84.8	81.5	667.8	52.7
				ExtContext(2-BW)	29.7	76.9	86.2	83.8	483.8	40.1
				ExtContext(7-BW)	31.5	77.8	86.7	85.0	441.5	35.9
62	31	30	15	StdContext	20.6	74.1	85.4	81.6	669.9	53.3
				ExtContext(2-BW)	29.7	77.0	86.6	84.0	495.6	41.5
				ExtContext(7-BW)	32.6	78.1	87.3	84.9	444.5	35.7
127	63	30	15	StdContext	20.8	74.4	85.7	81.6	661.6	54.8
				ExtContext(2-BW)	30.2	77.2	86.6	84.1	495.7	41.9
				ExtContext(7-BW)	32.2	78.1	87.3	84.9	458.9	36.9

Table 6: Performance of BCH error-correcting codes with and without extended context and right-to-left (BW) processing of the letters of the words for the combined phoneme/stress case.

Code Length n	Interword Distance d (min.)	Input Representation	% correct (1000-word test set)				Decision Tree data (mean)	
			Level of Aggregation					
			Word	Letter	Phon.	Stress	Leaves	Dep th
63	31	StdContext	20.3	74.3	83.8	80.3	1000.0	68.7
		ExtContext(2-BW)	29.3	76.4	85.3	82.3	774.3	53.5
		ExtContext(7-BW)	31.3	77.7	85.8	83.3	733.0	46.3
127	63	StdContext	22.3	75.5	85.2	81.5	1002.9	68.3
		ExtContext(2-BW)	29.0	77.1	86.0	83.2	782.9	53.9
		ExtContext(7-BW)	32.2	78.1	86.8	83.7	739.8	46.5
255	127	StdContext	22.4	75.5	85.2	81.6	1014.4	68.8
		ExtContext(2-BW)	30.6	77.3	86.3	83.1	792.9	55.3
		ExtContext(7-BW)	33.5	78.5	87.1	83.8	745.8	46.9

Table 7: Impact of enlarging the window for the *combined* phoneme/stress case when several *long* BCH error-correcting codes and "ExtContext(7-BW)" are employed.

Code Length n	Interword Distance d (min.)	Window Size w	% correct (1000-word test set) Level of Aggregation				Decision Tree data (mean)	
			Word	Letter	Phon.	Stress	Leaves	Depth
63	31	$w = 7$	31.3	77.7	85.8	83.3	733.0	46.3
		$w = 13$	35.4	78.1	85.4	84.3	667.8	44.8
127	63	$w = 7$	32.2	78.1	86.8	83.7	739.8	46.5
		$w = 13$	36.0	78.5	86.3	84.5	673.5	45.7
255	127	$w = 7$	33.5	78.5	87.1	83.8	745.8	46.9
		$w = 13$	36.4	79.1	86.5	85.4	680.7	46.1

reduction in phoneme performance, resulting in an overall improvement at the letter and word levels for the larger window size of 13.

It is instructive to compare the best performance figures shown in the above table with those of our base configuration (Table 1 of section 3.1). The word level correctness of 36.4% (achieved by the $n = 255, d = 127$ code and $w = 13$) is nearly three times the 12.5% correctness figure that we started out with. Performance figures at other levels of aggregation (phoneme, stress, and letter levels) have also dramatically increased. Indeed, we have come a long way!

Equally impressive is the fact that the machine learning techniques we have developed reached this level of performance (when tested on 1000 *unseen* words) after training on *only* 1000 words. In the next section we will show a further steep performance increase as a result of employing a much larger training set.

4 Comparison with DECtalk

In all the experiments reported in the previous sections, learning is accomplished by training on our standard 1000-word training set. This section, however, is performance oriented. The results of the best performing learning systems trained on 19,002 words are compared with the performance of the human-developed letter-to-sound rules incorporated in DECtalk; a commercially available device, marketed by Digital Equipment Corporation (DEC) for the English text-to-speech conversion task.

4.1 DECtalk's Outputs

To carry out this comparison, we had to obtain the pronunciations output by DECtalk's rule-base on our standard 1000-word test set.[2] However, before a valid comparison was possible, a number of obstacles had to be overcome. For one thing, the format of the outputs from

[2]We thank Tony Vitali of Digital Equipment Corporation (DEC), for running the entire NETtalk dictionary through DECtalk and providing us with the outputs from DECtalk's rule-base on *all* the words in the dictionary.

Table 8: Comparison of the performance of the best configuration of the learning system with the performance of DECtalk's rule-base on our standard 1000-word test set.

System and Configuration	Window Size	% correct (1000-word test set) Level of Aggregation					Decision Tree data (mean)	
		Word	Letter	Phoneme	Stress	Bit	Leaves	Depth
ID3 & Error Correcting Codes	15	64.8	91.4	93.7	95.1	95.7	5678	71.2
DECtalk's rule-base	N/A	36.4	83.3	87.4	91.8	N/A	N/A	N/A

DECtalk's rule-base was entirely different from the outputs of our learning systems. All the "silent" phonemes and "no-stress" symbols do not appear in the former, and the primary and secondary stress symbols are intermixed with the remaining phonemes. This means that the phoneme/stress string is not aligned with the letters of the words, which makes the performance evaluation at the phoneme, stress and letter levels nearly impossible. Other, less serious problems, included the disparity between some of the phonemes and stresses employed by the two systems. The solution we adopted was to painstakingly *re-align* the outputs of DECtalk's rule-base for all the 1000 words in our standard test set. The interested reader is referred to Bakiri, G. (1991a) for details of the original format for the outputs of DECtalk's rule-base and the procedure we employed to convert these to the standard *aligned* format used in the NETtalk dictionary.

4.2 Best Performance: Learning System

The best configurations determined in the previous secions for the learning method—ID3 with error-correcting output codes—was employed in the results reported in this section, but training was done on 19,002 words—the full NETtalk dictionary less our standard 1000-word test set. These generated 139,671 examples (or equivalently, "classified windows"). Testing was done on our standard 1000-word test set—disjoint from the training set. "Ext-Context(7)" input representation was employed, with the letters of the words processed backwards in a right-to-left sequence. Cross validation techniques indicated that the "optimal" window size this configuration would be 15. This implies that the extended context for each letter included upto 7 letters on each side and the *phonetic context* of the letters to the *right* of the current letter. The output code employed was a 127-bit [3] , BCH error-correcting code with a minimum inter-word Hamming distance, d, of 63.

4.3 Results: Strict Evaluation

Table 8 shows the performance of learning system—with the configurations detailed above—along with the performance of DECtalk's rule-base on our standard 1000-word test set.

[3]It took about 3 CPU-weeks to build the 127 ID3 trees with this configuration on a SUN4 workstation. A 256-bit code would probably give slightly better performance, but the training time would be twice as long.

Table 9: Serious error-rates for three commercial text-to-speech systems based on a 1987 study by Bill Huggins at BBN (Bolt, Baranek, and Newman). The test was over a set of 1,678 polysyllabic inflected forms, like "reinforce".

System	"Serious" Error Rate
DECtalk2	12.9%
Speech+ calltext 3	8.3%
DECtalk3	5.1%

The 64.8% correctness figure at the word-level for the learning system is a 78% improvement over the 36.4% word-level correctness for DECtalk's rule base. Furthermore, the dismal word level correctness figure for DECtalk indicates that our correctness criteria are too stringent. We consider a word to be mispronounced if *any* of its letters were mapped to *either* an incorrect phoneme *or* an incorrect stress—regardless of the seriousness of the errors committed. This criterion, however stringent, is nevertheless straight forward and very objective. By contrast, previous evaluations of text-to-speech systems reported only "serious" error rates. Table 9 shows error rates as relayed to us by [Vitali90] for three commercial text-to-speech systems based on this—rather subjective—measure. It is unfortunate that we can not go much beyond simply reporting these figures, since we do not know precisely what constitutes a *serious* error and what does not.

Table 8 also shows that the learning method outperform DECtalk's rule-base at the phoneme, stress, letter and word levels of aggregation. While one can argue that the lower phoneme level performance for DECtalk[4] could be attributed to the manner in which DECtalk's phonetic rules were encoded,[5] no such argument can be ma de in the case of the stresses. The stress error rate of 4.9% for the best learning method represents a reduction of 40.2% from the 8.2% error rates committed by DECtalk's rules for determining the stresses. This unambiguously suggests that the machine learning techniques developed as part of our research work compete rather well with human-engineered rules for the text-to-speech domain.

A two way detailed categorization of how each of the two systems classifies each of the 7,242 "windows" in the test set is shown in Table 10. These data show that while there are 458 windows (6.32%) correctly classified by DECtalk's rule-base that are missed by the learning method, there are 1,047 windows (14.46%) correctly classified by the learning method that are misclassified by DECtalk's rule-base.

The two systems agree on 5,737 windows (79.22%); they both correctly classify 5,574 windows (76.97%) and both misclassify only 163 windows (2.25%).

[4]The particular version of DECtalk that is referred to throughout this paper is DECtalk3.

[5]These rules may be biased towards some particular *accent* or way of pronunciation of certain words which may not be entirely wrong but nevertheless different from the way the NETtalk dictionary encodes these pronunciations.

Table 10: Classification of test set windows by the ID3 & error-correcting codes learning method and DECtalk's rule-base.

		Error-correcting Codes: Correct	Error-correcting Codes: Incorrect	
DECtalk's Rule-base	Correct	5,574 (76.97%)	458 (6.32%)	Disagree: 1,505 (20.78%)
	Incorrect	1,047 (14.46%)	163 2.25%	Agree: 5,737 (79.22%)

4.4 A Human Study Comparison

The criteria for correctness employed in the comparative study of the previous section were very strict: The phoneme and stress classes were required to match *exactly.* A little thought shows that this is overly strict. Some phoneme errors (e.g., substituting `/k/` for `/e/`) are very serious, while others (e.g., substituting `/x/` for `/@/`) are virtually indistinguishable in the pronunciation of some words. Similarly, some stress errors (e.g., substituting *primary* stress with *no* stress) are generally much more serious than other stress errors (e.g., substituting *primary* stress with *secondary* stress). Hence, while these comparisons are valuable, the performance figures shown in the previous section do not give a good indication of the *quality* of the speech produced by the learning system relative to that produced by DECtalk's rule-base.

In the next subsection, we present results of a different kind of comparative study: one in which the correctness criteria were relaxed to consider pronunciations that sound similar to human subjects.

4.4.1 Evaluations by Human Subjects

In order to get a good indication of the *quality* of the speech produced by the learning systems—considered in this chapter—relative to that produced by DECtalk's rule-base, we decided to let human subjects decide how "correct" the pronunciation of each system is when listened to through DECtalk's hardware.[6]

For each of the methods considered, three subjects were asked to compare how close the pronunciation output by the method—for each of the 1000-words in our standard test set—compared with the correct (NETtalk dictionary) pronunciation.[7] The subjects were allowed to listen to the correct (NETtalk dictionary) pronunciation of each word followed by the method's pronunciation, then select one of the following responses:

1. There is *No* difference between the two pronunciations, or,

[6]The DECtalk machine optionally allows passing the *Klattized* format of the pronunciations (see [Bakiri93a] for details) of the wor ds through the speech synthesis hardware, by-passing DECtalk's dictionary and rule-base.

[7]If the method pronunciation (phoneme/stress strings) matched the dictionary *exactly*, the word was *automatically* considered correct.

Table 11: Results of the human subject comparative study if an error is counted when the majority of the subjects responses was either 2 (noticeable difference) or 3 (serious difference). See text for details.

System and Configuration	Direct Words Correct	% correct (1000-word test set)			
		Level of Aggregation			
		Word	Letter	Phon.	Stress
ID3 & Error Correcting Codes	71.2	72.9	93.9	95.1	97.3
DECtalk's rule-base	55.7	58.5	90.1	91.7	96.5

2. There is a *Noticeable* difference between the two pronunciations, or,

3. There is a *Serious* difference between the two pronunciations, or,

4. The method pronunciation is better than the dictionary pronunciation. Therefore, count the method pronunciation as correct.

Each of the three subjects was free to respond with any of the above, so the responses had the form: 111, 122, 233, etc. Majority vote was taken, so that 122 would be counted as 2, 233 would be counted as 3, and so on.

The above procedure would give us a basis for evaluating errors at the word level but not at the individual phoneme, stress, or, letter level. To get a subjective evaluation of the performance at these lower levels of aggregations, we extended the above experimental procedure to count the individual phoneme/stress errors as follows:

1. If the response was 1 or 4, the Phonemes and Stresses corresponding to *All* the letters of the word was taken as correct.

2. Otherwise (i.e. response was 2 or 3):

   - Method PS list was compared with dictionary PS list to find all the errors (differences).
   - Dictionary PS list is mutated with one error at a time (either P or S), and the resulting pronunciation presented to the subjects to judge the seriousness of the error.
   - Acceptable responses were 1, 2 or 3 as before.

Table 11 shows performance results when *noticeable* or *serious* errors are counted, while table 12 compares the two systems when *only* the "*serious* difference" responses are counted as errors. These results give further support to the validity of the previous *strict* comparison and show that the machine learning system considered outperform the human-developed letter-to-sound rules incorporated in DECtalk's rule-base *regardless* of the correctness criteria employed.

Table 12: Results of the human subject comparative study if an error is counted *only* when the majority of the subjects responses was 3: serious difference. See text for details.

System and Configuration	Direct Words Correct	% correct (1000-word test set)			
		Level of Aggregation			
		Word	Letter	Phon.	Stress
ID3 & Error Correcting Codes	83.3	85.3	97.6	97.7	99.8
DECtalk's rule-base	73.9	81.6	97.1	97.2	99.7

5 Conclusions

The experiments in this paper demonstrate that machine learning techniques compete effectively with expert systems in the text-to-speech domain. They substantially outperformed expert-developed letter-to-sound rules incorporated in DECtalk in all the comparative tests we performed. We have also shown that error-correcting output codes provide an excellent method for applying binary learning algorithms to multiclass learning problems. They outperform the domain-specific distributed output code (the Sejnowski-Rosenberg code), and in Bakiri, G., & Dietterich, T. G. (1991), we show that they outperform the direct multiclass method and the one-per-class method for the text-to-speech domain. Finally, we should point out that machine learning techniques are general and can be applied to the task of discovering pronunciation rules in other languages. In fact this is one of our current research efforts in the computer science department in the University of Bahrain.

6 References

Abramson, N. (1963). Information theory and coding. McGraw-Hill Book Company, Inc.

Bakiri, G. (1991a). Converting English Text to Speech: A Machine Learning Approach. A PhD Thesis, Oregon State University, 1991. (Available through U.M.I, 300 N. Zeeb Rd., Ann Arbor, MI 48106, U.S.A.)

Bakiri, G., & Dietterich, T. G. (1991). Boosting the Performance of Inductive Learning Programs via Error-Correcting Output Codes. *Proceedings of the Eighth International Conference on Machine Learning.* Anaheim Ca: Morgan Kaufmann.

Bakiri, G., & Dietterich, T. G. (1993). Applying machine learning techniques to construct high-performance letter-to-sound rules for English. Forthcoming.

Bose, R. C., & Ray-Chaudhuri, D. K. (1960). On a class of error-correcting binary group codes. *Inf. Control*, 3, pp. 68-79, March 1960.

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

Church, K. W. (1985). Stress assignment in letter-to-sound rules for speech synthesis. *Proc. 23rd Meeting Assoc. Comp. Ling.*, 246-253.

Cole, R. Muthusamy, Y. and Fanty, M. (1990). The ISOLET spoken letter database. Technical Report No. CSE 90-004. Beaverton, OR: Oregon Graduate Institute of Science and Technology, Department of Computer Science and Engineering.

Dietterich, T. G., Hild, H., Bakiri, G. (1990a) 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.

Halle, M. & Keyser S. J. (1971). English stress: its form, its growth, and its role in verse. New York: Harper and Row.

Klatt, D. H. (1979). Synthesis by Rule of Consonant-Vowel Syllables. In Speech Communications Group Working Papers, Cambridge, Massachusetts: Massachusetts Institute of Technology.

Klatt, D. H., & Shipman, D. W. (1982). Letter-to-phoneme rules: A semi-automatic discovery procedure. *J. Acoust. Soc. Am. Suppl. 1* 72, S48.

Klatt, D. H. (1987). Review of text-to-speech conversion for English. *J. Acoust. Soc. Am., 82*, (3), 737–793.

LeCun, Y. Boser, B., Denker, J. S., Henderson, D. Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation, 1* (4), 541–551.

Lucassen, J. M. (1983). Discovering phonemic base forms automatically: An information theoretic approach. Research Report RC 9833. Yorktown Heights, NY: IBM T. J. Watson Research Center.

Lucassen, J. M., and 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.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess endgames, in Michalski, R. S., Carbonell, J., and Mitchell, T. M., (eds.), *Machine learning: An artificial intelligence approach, Vol. I*, Palo Alto: Tioga Press. 463–482.

Quinlan, J. R. (1986a). The effect of noise on concept learning. In Michalski, R. S., Carbonell, J., and Mitchell, T. M., (eds.), *Machine learning, Vol. II*, Palo Alto: Tioga Press. 149–166.

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

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

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review, 65* (6), 386–408.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E., and McClelland, J. L., (eds.) *Parallel Distributed Processing*, Vol 1. 318–362.

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

Shavlik, J. W., Mooney, R. J., and Towell, G. G. (1990). Symbolic and neural learning algorithms: An experimental comparison (revised). Technical Report 955. Madison, WI: University of Wisconsin-Madison, Computer Sciences Department.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell Syst. J.*, 27, pp. 379-423 (part I), 623-656 (Part II), July 1948.

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

Tony Vitali (1990). Personal communication (electronic mail) between Tony Vitali of Digital Equipment Corporation (DEC) and Tom Dietterich.