Achieving High-Accuracy Text-to-Speech with Machine Learning

Ghulum Bakiri
Department of Computer Science
University of Bahrain
Isa Town, Bahrain
microcen@batelco.com.bh

Thomas G. Dietterich
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331 USA
tgd@cs.orst.edu

Draft of May 30, 1997.

Abstract

In 1987, Sejnowski and Rosenberg developed their famous NETtalk system for English text-to-speech. This chapter describes a machine learning approach to text-to-speech that builds upon and extends the initial NETtalk work. Among the many extensions to the NETtalk system were the following: a different learning algorithm, a wider input "window", error-correcting output coding, a right-to-left scan of the word to be pronounced (with the results of each decision influencing subsequent decisions), and the addition of several useful input features. These changes yielded a system that performs much better than the original NETtalk system. After training on 19,002 words, the system achieves 93.7% correct pronunciation of individual phonemes and 64.8% correct pronunciation of whole words (where the pronunciation must exactly match the dictionary pronunciation to be correct). Based on the judgements of three human participants in a blind assessment study, our system was estimated to have a serious error rate of 16.7% (on whole words) compared to an error rate of 26.1% for the DECTalk3.0 rulebase.

1 Introduction

There have been many attempts to apply machine learning algorithms to solve the problem of mapping spelled English words into strings of phonemes that can drive speech synthesis hardware (e.g., Lucassen, 1983; Lucassen & Mercer, 1984). One of the best-known attempts was the NETtalk system, developed by Sejnowski and Rosenberg (1987). This chapter describes a line of research inspired by the NETtalk system. Our goal was to take the basic approach developed in NETtalk and see how far we could extend it to create a high-performance text-to-speech system for individual English words.

The chapter is organized as follows. First, we review the NETtalk formulation of the text-to-speech task. Second, we describe a series of studies that were carried out to evaluate various design decisions. The resulting text-to-speech system is described in detail. Finally, we present an experimental evaluation of the learned text-to-speech system.

2 The NETtalk Approach

2.1 The NETtalk Dictionary

To apply machine learning algorithms to the text-to-speech problem, the first step is to construct a dictionary of words and their pronunciations. Rosenberg and Sejnowski developed a dictionary of 20,003 English words and their pronunciations by starting with Miriam Websters's Pocket Dictionary and adjusting the pronunciations to obtain good results when played through the DECtalk speech synthesis device. The pronunciations in this dictionary are represented by two strings: a string of phoneme symbols and a string of stress symbols. Both strings are aligned letter-for-letter with the English word. For example, the word "lollypop" is encoded in the dictionary as follows:

Word:	`lollypop`
Phoneme string:	`lal-ipap`
Stress string:	`>1<>0>2<`

There are 54 possible phoneme symbols, and 6 stress symbols. The phoneme symbols are based on the ARPABET (and hence, they are very closely related to the phoneme symbols used in the DECTalk system (Digital Equipment Corporation, 1985)). Some modifications were required to maintain the one-to-one alignment. For situations where a single letter is actually pronounced as a sequence of two phonemes, Sejnowski and Rosenberg introduced a new phoneme symbol. For situations where a string of letters is pronounced as a single phoneme, that phoneme was aligned with the first letter in the string, and the subsequent letters were coded with the "silent phoneme" (represented as `/-/`). Silent letters were also coded using the silent phoneme. Table 1 gives the meaning of each symbol.

The stress symbols indicate how much stress each syllable should receive and also locate the syllable boundaries. The meanings of the stress symbols are given in Table 2

2.2 Inputs and Outputs to a Learning Algorithm

Most learning algorithms take as input training examples of the form $\langle \mathbf{x}, y \rangle$, where $\mathbf{x}$ is a fixed-length vector of feature values and y is a discrete-valued variable (the desired output "class"). Because English words have variable length, this poses a problem. The approach taken in NETtalk was to formulate the problem of pronouncing an entire word as the sequence of smaller problems of pronouncing each individual letter. Hence, one training example is created for each letter in the word. The $\mathbf{x}$ vector of features describes a 7-letter "window" centered on the letter to be pronounced (and including the 3 letters before and the 3 letters after that letter). The y output is a pair of values: the desired phoneme symbol and the desired stress symbol. For example, the word "lollypop" is converted into 8 training examples as shown in Table 3.

In principle, there are 324 possible y values (corresponding to all possible combinations of phonemes and stresses). However, many of these combinations cannot occur in practice. Approximately 140 of these combinations are observed in the NETtalk dictionary.

The learning algorithm employed by Sejnowski and Rosenberg was the backpropagation algorithm for training feed-forward neural networks (Rumelhart, Hinton, & Williams, 1986). To work well with backpropagation, it was necessary to convert each input and output into a collection of binary feature values. Specifically, each letter was converted to a string of 29 binary features. There was one feature for each letter and 3 features to encode blank spaces between words and punctuation. The punctuation features were not used in this study, so only 27 binary features were necessary. However, in most of our experiments we used all 29 binary features (the punctuation features always had the value 0). This produced a total of 203 input feature (29×7).

Table 1: The NETtalk phoneme symbols

Phoneme	Examples	Phoneme	Examples
/a/	wAd, dOt, Odd	/E/	mAny, End, hEAd
/b/	Bad	/G/	leNGth, loNG, baNk
/c/	Or, cAUght	/I/	gIve, bUsy, captAIn
/d/	aDd	/J/	Jam, Gem
/e/	Angel, blAde, wAy	/K/	aNXious, seXual
/f/	Farm	/L/	eviL, abLe
/g/	Gap	/M/	chasM
/h/	Hot, WHo	/N/	shorteN, basiN
/i/	Eve, bEe	/O/	OIl, bOY
/k/	Cab, Keep	/Q/	Quilt
/l/	Lad	/R/	honeR, afteR, satyR
/m/	Man, iMp	/S/	oCean, wiSH
/n/	GNat, aNd	/T/	THaw, baTH
/o/	Only, Own	/U/	wOOd, cOUld, pUt
/p/	Pad, aPt	/W/	oUT, toWel, hoUse
/r/	Rap	/X/	miXture, anneX
/s/	Cent, aSk	/Y/	Use, fEUd, nEw
/t/	Tab	/Z/	uSual, viSion
/u/	bOOt, OOze, yOU	/@/	cAb, plAId
/v/	Vat	/!/	naZi, piZZa
/w/	We, liqUid	/#/	auXiliary, eXist
/x/	pirAte, welcOme	/*/	WHat
/y/	Yes, senIor	/^/	Up, sOn, blOOd
/z/	Zoo, goeS	/+/	abattOIr, mademOIselle
/A/	Ice, hEIght, EYe	/-/	silence
/C/	CHart, Cello	/_/	word-boundary
/D/	THe, moTHer	/./	period

To convert the outputs to binary, Sejnowski and Rosenberg employed a 26-bit distributed output encoding. The phoneme symbol was encoded using 21 bits, where each bit had a meaning. For example, six bits encoded the location of the articulation point in the mouth (i.e., alveolar, dental, glottal, labial, palatal, velar). Three bits encoded the vertical position of the articulation point (i.e., high, medium, low). Eight bits encoded other phoneme properties (i.e., affricative, fricative, glide, liquid, nasal, stop, tensed, voiced).

The stress symbol encoded as a string of 5 bits.

2.3 Training and Classification

Sejnowski and Rosenberg trained a large neural network via the backpropagation algorithm. Once the network was trained, it could be applied to pronounce a new word as follows.

First, the new word was converted into a sequence of windows. The input features for each window were given as input to the neural network, and the output of the neural network was a vector of 26 "output activations" (each between 0 and 1). The first 21 bits of this vector were

Table 2: The NETtalk stress symbols

Symbol	Meaning
`1`	Syllable nucleus with primary stress
`2`	Syllable nucleus with secondary stress
`0`	Syllable nucleus with no stress
`<`	Nucleus of this syllable is to the left
`>`	Nucleus of this syllable is to the right
`_`	Non-letter (beyond start or end of word)

Table 3: Training examples computed by converting the word `lollypop` to a series of 7-letter windows.

Window ($\mathbf{x}$)	Pronunciation (y)
`___loll`	`l>`
`__lolly`	`a1`
`_lollyp`	`l<`
`lollypo`	`->`
`ollypop`	`i0`
`llypop_`	`p>`
`lypop__`	`a2`
`ypop___`	`p<`

compared with the 21-bit encodings of each of the 54 phoneme symbols, and the symbol whose 21-bit string was closest to the output activation vector was chosen as the output symbol. A similar procedure was followed for choosing the output stress symbol.

The phoneme and stress symbols computed from each window were concatenated to obtain the phoneme string and stress string for the whole word.

3 Designing a High-Performance Machine Learning Approach

Our research was initially motivated by the goal of comparing the performance of the backpropagation algorithm with alternative machine learning algorithms. We have performed several studies comparing backpropagation with algorithms for decision trees, radial basis functions, and various forms of nearest neighbor classifiers (Bakiri, 1991; Dietterich, Hild, & Bakiri, 1995; Wettschereck & Dietterich, 1992). While these studies revealed interesting properties of the learning algorithms, they did not address the question of how well learning algorithms could do in comparison with non-learning approaches. To answer this question, we undertook the study described in this chapter.

For the purposes of evaluating various design decisions, we chose a small 2,000-word subset of the NETtalk dictionary (at random, without replacement). We chose this subset because many learning algorithms are computationally expensive, and we wanted to make the experiments tractable. This 2,000-word subset was further randomly partitioned into a 1,000-word training set and a 1,000-word test set. This 1,000-word test set was used for all of the experiments reported in this paper (with one exception, noted below).

3.1 The Learning Algorithm

Training the backpropagation algorithm is difficult for two reasons: it is computationally expensive and the user must choose several parameter values. Specifically, it takes several hours to train a large neural network for the NETtalk task, even when restricted to the 1,000-word training set. The user must choose the number of hidden units, the initial random weight values, the learning rate, the momentum coefficient, and the stopping point. While there exist automated methods for making these choices, they typically involve making multiple runs and employing cross-validation or holdout datasets. These increase the computational cost, often by orders of magnitude.

To avoid these problems, we chose the ID3 decision tree algorithm (Quinlan, 1983, 1986) for all of our experiments. ID3 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 labeled 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, 1983).

In our implementation of ID3, we did not employ windowing, χ^2 early stopping, or any kind of pruning. Initial experiments (Dietterich, Hild, & Bakiri, 1990) showed that these stopping and pruning methods did not improve performance.

We did apply one simple stopping rule 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 also have zero mutual information).

We applied ID3 to the initial NETtalk formulation. Unlike backpropagation, ID3 can only learn a single function at a time, whereas a neural network can have multiple output units. Hence, we run ID3 26 times, once for each of the output bits of the distributed output representation, and obtain 26 decision trees. When classifying a new example, we evaluate each of the 26 decision trees to assemble a vector of 26 bits. We then choose the output phoneme and stress using a slight modification of the procedure employed in NETtalk. During training, we compute a list of all phoneme/stress combinations that occur in the training set. During classification, we compute the distance between the 26-bit result vector and the 26-bit encodings of each *observed* phoneme and stress pair. Ties are broken in favor of phoneme/stress pairs that occurred more often in the training data. This is called *observed decoding.* Previous studies have shown that observed decoding improves the performance of ID3 substantially, but that it is still not as good as the performance of backpropagation (Dietterich et al., 1995).

Table 4 summarizes the performance of ID3 applied using the NETtalk formulation of the text-to-speech task. Performance is given in percentage correct at various levels of aggregation. The column labeled "Stress" gives the number of stress symbols correctly predicted on the test set. The "Phoneme" column gives the number of phoneme symbols correctly predicted on the test set. The "Letter" column gives the percentage of test letters where both the phoneme and stress symbols were correctly predicted. And the "Word" column gives the percentage of words for which every phoneme and stress symbol was correctly predicted. Not surprisingly, this is a low number (12.5%) on the test data, because any single error in a phoneme or stress symbol will cause a word to be

Table 4: Performance of ID3 using the NETtalk formulation on 1000-word training and test sets. The row labeled "NETtalk" gives the test set performance of backpropagation on a network with 160 hidden units and cross-validated early stopping (see Dietterich, Hild & Bakiri, 1995).

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
NETtalk	14.3	71.5	82.0	81.4	96.7	N/A	N/A

Table 5: Six alternative output configurations. Each cell gives the number of decision trees that need to be learned. For error-correcting output coding, the number of trees depends on the length of the code(s) used.

	One-per-class	Multiclass	ECOC
Separate Phoneme/Stress	60	2	$L_p + L_s$
Combined Phoneme/Stress	126	1	L

considered incorrect. The "Bit" column gives the mean percentage of correct predictions of the 26 individual output bits. The mean number of leaves and depth of the 26 decision trees are also provided. We will refer to this configuration of ID3 as the "base configuration."

The table also shows the performance of backpropagation trained on the same training set and tested on the same test set. We see that backpropagation performs somewhat better than ID3. Nonetheless, the extremely long training times required for backpropagation forced us to adopt the very efficient ID3 for the research described in this paper.

3.2 Output Representations

After choosing the learning algorithm, the next design decision was to select an output representation. We considered six alternative output representations, as summarized in Table 5. The approached employed in NETtalk predicts the phonemes and stresses separately. We call this the "Separate Phoneme/Stress" output representation. An alternative is to treat each observed combination of phoneme and stress as a distinct class. 126 such combinations are observed in the training data, so this can be implemented by defining a single output class variable with 126 distinct values. We call this the "Combined Phoneme/Stress" method.

With either the Separate or the Combined Phoneme/Stress methods, there are three alternative ways of encoding the classes. The *one-per-class* approach defines a separate boolean function for each possible value of the class variable. In the Combined Phoneme/Stress method, this would involve learning 126 separate decision trees, one for each possible class. In the Separate Phoneme/Stress method, this would require 54 decision trees for phonemes and 6 decision trees for stresses. The one-per-class method is the method most commonly used in neural network problems. A drawback of the one-per-class method is that when the class of a new example is predicted, more than one of the decision trees may classify the new example as "positive" (i.e., more than one phoneme would be predicted). Such ties could be broken in favor of the more-frequently-occurring phoneme.

The second encoding of the classes is the direct multiclass method. Unlike neural networks, decision trees can handle multiple classes easily. Each leaf of the decision tree is labeled with the value of the class variable that is to be predicted for examples which reach that leaf. This is the standard output encoding used in decision tree applications. With this representation, in the Separate Phoneme/Stress approach, only 2 decision trees need to be learned.

The third encoding is the error-correcting output coding method (ECOC), which was directly inspired by the distributed output encoding of Sejnowski and Rosenberg. In the ECOC approach, each class, $k = 1, \ldots, K$, is represented by an L-bit binary codeword, C_k. The codewords are selected to form a good error-correcting code—that is, each pair of codewords differs in at least d bit positions, for some reasonably large value of d. A decision tree is learned for each bit position. The decision tree for bit position ℓ is trained to produce an output of 0 for each training example that belongs to a class y whose codeword has a 0 in the ℓ-th bit position (and to produce a 1 otherwise). To classify a new example, all L decision trees are evaluated to compute a L-bit binary string. The class k is chosen whose codeword C_k is closest to this binary string (where distance is measured by the number of bit positions where the strings differ—the Hamming distance).

An alternative way of understanding the ECOC approach is the following. Suppose we take the K possible classes and partition them into two sets of approximately equal size A and B. We could then train a decision tree to classify examples according to whether they belonged to set A or set B. If an example belongs to a class $k \in A$, then the decision tree should output a zero; otherwise, it should output a 1. Now suppose we repeat this process L times. Each time, we divide the classes into two sets A_ℓ and B_ℓ. We learn a set of L decision trees.

To classify a new example, we classify it using each of these L trees. If decision tree ℓ outputs as 0, we give one "vote" to each class in A_ℓ, otherwise we give a vote to each class in B_ℓ. After all of the decision trees have been evaluated, the class with the largest number of votes is chosen as the output class.

There are two intuitions for why the ECOC approach should work well. From the error-correcting code perspective, we can view the decision trees as making "noisy" predictions. The error-correcting code permits us to recover from errors in some of those predictions. Specifically, if fewer than $\lfloor (d-1)/2 \rfloor$ prediction errors are made by the individual trees, then the resulting class will still be correct. In practice, this permits roughly one fourth of the decision trees to make errors on each test example.

Another intuition is that ECOC is a form of committee or ensemble method, closely related to the methods of bagging (Breiman, 1996) and boosting (Freund & Schapire, 1996; Quinlan, 1996). According to this view, ECOC is an interesting method for constructing a group of diverse yet accurate classifiers.

Experimentally, the error-correcting output coding method has been shown to give excellent results with both decision tree and neural network learning algorithms (Dietterich & Bakiri, 1995). We applied BCH code design methods (Bose & Ray-Chaudhuri, 1960; Hocquenghem, 1959; Lin & Costello, 1983) to design good error correcting codes of various lengths for both the separate and combined phoneme/stress configurations.

Table 6 shows the results of the base configuration and the three alternative output encodings for the Separate Phoneme/Stress approach. Table 7 shows corresponding results for the Combined Phoneme/Stress approach. The first conclusion that can be drawn from these tables is that the ECOC method works very well. It clearly outperforms the other methods. The second-best method is the multiclass approach. The original Sejnowski-Rosenberg encoding comes next, and the one-per-class method gives the worst performance.

Table 6: Comparison of output encoding methods for the Separate Phoneme/Stress approach. L is the length of the error-correcting code, and d is the minimum Hamming distance between each pair of codewords.

				% correct (1000-word test set)					Decision Tree	
				Level of Aggregation					data (mean)	
Configuration				Word	Letter	Phoneme	Stress	Bit (mean)	Leaves	Depth
Base				12.5	69.6	81.3	79.2	96.3	269.9	29.3
One-per-class				11.8	69.5	80.6	78.9	98.7	90.4	14.0
Multiclass				13.0	69.7	82.4	79.5	N/A	1305.0	48.0
Phoneme		Stress		Error-Correcting Output Coding						
L	d	L	d	Word	Letter	Phoneme	Stress	Bit (mean)	Leaves	Depth
10	3	9	3	13.3	69.8	80.3	80.6	90.8	677.4	51.9
14	5	11	5	14.4	70.9	82.3	80.3	91.0	684.7	53.1
21	7	13	7	17.2	72.2	83.9	80.4	91.2	681.4	53.9
26	11	13	11	17.5	72.3	84.2	80.4	91.0	700.5	56.4
31	15	30	15	19.9	73.8	84.8	81.5	91.6	667.8	52.7
62	31	30	15	20.6	74.1	85.4	81.6	92.0	669.9	53.3
127	63	30	15	20.8	74.4	85.7	81.6	92.4	661.6	54.8

The comparison of the Separate versus Combined Phoneme/Stress encodings is less dramatic, but in general the Combined approach works a little better, particularly at the Letter and Word levels of aggregation. Based on these observations, we decided to employ the Combined approach with error-correcting output codes in our final system. We also decided to use the 127-bit code (rather than the 255-bit code), because the two codes gave almost identical performance and the 127-bit code requires only half as much CPU time and memory to learn.

We performed various other experiments to determine how sensitive these results were to the exact assignment of codewords to classes and to the choice of training and test sets. These experiments showed that the results are the same regardless of how codewords are assigned to classes.

Table 7: Comparison of output encoding methods for the Combined Phoneme/Stress approach.

		% correct (1000-word test set)					Decision Tree	
		Level of Aggregation					data (mean)	
Configuration		Word	Letter	Phoneme	Stress	Bit (mean)	Leaves	Depth
Base		12.5	69.6	81.3	79.2	96.3	269.9	29.3
One-per-class		8.7	66.7	76.4	74.5	99.5	34.9	10.5
Multiclass		13.5	70.8	81.1	78.3	N/A	1987.0	54.9
L	d	Error-Correcting Output Coding						
63	31	20.3	74.3	83.8	80.3	87.4	1000.0	68.7
127	63	22.3	75.5	85.2	81.5	87.8	1002.9	68.3
255	127	22.4	75.5	85.2	81.6	87.8	1014.4	68.8

3.3 Input Representation

In parallel with our experiments on output encoding, we explored ways of improving the input representation for our system. One promising approach is to use the results of previous decisions when deciding how to pronounce a letter in a word. Specifically, suppose that we scan a word from left-to-right. After we predict the phoneme and stress of the first letter, we use this information as part of the input features to predict the phoneme and stress of the second letter. Then we use the results of the first two predictions as input to help predict the phoneme and stress of the third letter, and so on. One reason to expect this to work is that the stress patterns of English generally alternate between stressed and unstressed syllables. Hence, if the previous syllable was stressed, this could help the learning algorithm choose the correct stress symbol for the next syllable. This approach was first tested by Lucassen and Mercer (1983).

To construct training examples for this approach, we encode the letters, phonemes, and stresses of the three letters to the left of the target letter as well as three letters to the right of the target letter. Each training example has the following general form:

$$L_{-3}, L_{-2}, L_{-1}, L_0, L_{+1}, L_{+2}, L_{+3}, P_{-3}, S_{-3}, P_{-2}, S_{-2}, P_{-1}, S_{-1} \rightarrow P_0 S_0,$$

where L_i is a set of features representing the letter at position i (0 being the position of the target letter whose phoneme and stress we are trying to prediction), P_i is a set of features representing the phoneme at position i and S_i is a set of features representing the stress at position i. During classification, we do not know the true value for $P_{-1}, S_{-1}, P_{-2}, S_{-2}, P_{-3}$, and S_{-3}, so we use the values computed by previous predictions.

We experimented with various methods for representing the phonemes and stresses. After performing several experiments, we chose the following features:

- **Phoneme:** Each phoneme P_i is represented by 54 binary features, one for each possible phoneme symbol.
- **Tense:** Each phoneme P_i is also represented by a binary feature that is 1 if the phoneme is one of the symbols `/a/`, `/e/`, `/i/`, `/o/`, `/u/`, `/A/`, `/O/`, `/W/`, or `/Y/`. Otherwise it is 0. This encodes the difference between "tense" and "lax" vowels.
- **Stress:** Each stress S_i is represented by 6 binary features, one for each possible stress symbol.
- **Nucleus** and **Stressed:** Each stresss S_i is also represented by two additional binary features. The first feature is 1 if the stress is one of `/0/`, `/1/`, or `/2/`, and 0 otherwise (i.e., this letter is a "syllable nucleus"). The second feature is 1 if the stress is `/1/` or `/2/`, and 0 otherwise (i.e., this letter receives some stress).
- **Letter:** Each letter L_i is represented by 29 binary features, one for each possible letter symbol (including blanks and punctuation).
- **Consonant** and **Non-Low-Vowel:** Each letter L_i is also represented by a two additional binary features. The first feature is 0 if the letter is one of A, E, I, O, U, Y, blank, or one of the punctuation symbols. Otherwise it is 1. The second feature is 0 if the letter is one of E, I, O, or U, and 1 otherwise.

We will call this the Extended representation.

A question that arises when applying the extended representation is whether the word should be scanned left-to-right or right-to-left. This point was (briefly) investigated in the work of Mercer

Table 8: Performance of the Extended input representation applied left-to-right and right-to-left.

Input Representation	Order of letter processing	% correct (1000-word test set) Level of Aggregation				Decision Tree data (mean)	
		Word	Letter	Phoneme	Stress	Leaves	Depth
Base Configuration		12.5	69.6	81.3	79.2	269.9	29.3
Extended	Left-to-Right	17.0	67.9	81.2	77.1	207.2	24.2
Extended	Right-to-Left	24.4	74.2	83.9	82.6	180.1	22.0
Extended + 127-bit ECOC	Right-to-Left	32.2	78.1	86.8	83.7	739.8	46.5

and Lucassen. The result of their investigation was summarized Lucassen as follows (Lucassen, 1983, p. 11):

> The [system] operates on a word *from left to right*, predicting phones in left-to-right order. This decision was made after preliminary testing failed to indicate any advantage to either direction. The left-to-right direction was chosen to simplify the interface to the linguistic decoder, as well as because of its intuitive appeal.

Our findings were entirely different. Table 8 shows the benefit of the extended representation being applied left-to-right and right-to-left. (The outputs were encoded using the original Sejnowski-Rosenberg distributed output representation.) We can see that the right-to-left order works much better (and yields smaller decision trees) than the left-to-right order. This result makes sense when you consider word pairs such as "photograph" and "photography". In this case, the final "y" influences the pronunciation of the "a". This then influences the pronunciation of the second "o", which in turn influences the pronunciation of the first "o". So even though the final "y" is more than 3 letters away from the first "o", and hence, far outside the 7-letter window, the right-to-left scan can "carry" the information from the end of the word back to the beginning. The table also shows that both forms of context dramatically improve performance compared to the base configuration.

After we developed the extended context, we checked to see how well it worked when combined with the ECOC method. The last line of Table 8 shows the performance of a 127-bit ECOC with the Combined Phoneme/Stress representation. We see that the combination of the extended context, right-to-left processing, and the error-correcting codes gives a substantial increase in performance.

We studied one other question concerning the input representation. All of our experiments thus far have employed a 7-letter window. What is the effect of using a larger window? A larger window can provide more useful information, but it can also provide more "noise" in the form of irrelevant features. Also, the extended context may already be providing most of the benefit that could be obtained from a larger window. Table 9 compares the performance of the Extended Right-to-Left input representation (with the Sejnowski-Rosenberg output representation) for different window sizes. The table shows that larger windows perform slightly better, particularly at the level of whole words.

Based on these (and many other) experiments, we chose our final configuration to use a 15-letter window with the following input features:

- **Phoneme** (same as Extended Context)
- **Tense** (same as Extended Context)

Table 9: Effect of enlarging the window on the Extended Context encoding (scanned right-to-left).

Window Size w	% correct (1000-word test set) Level of Aggregation					Decision Tree data (mean)	
	Word	Letter	Phoneme	Stress	Bit	Leaves	Depth
7	24.4	74.2	83.9	82.6	96.9	180.1	22.0
9	25.1	72.9	82.8	82.2	96.9	158.5	20.1
11	25.4	73.4	82.9	82.5	97.0	155.3	19.9
13	26.2	73.4	82.8	82.6	97.0	152.7	19.5
15	25.9	73.4	82.7	82.7	96.9	151.9	19.1

Table 10: Performance of the final configuration compared to the base configuration. Training and testing on 1000 words.

	% correct (1000-word test set) Level of Aggregation				Decision Tree data (mean)	
Configuration	Word	Letter	Phon.	Stress	Leaves	Depth
Base[a]	12.5	69.6	81.3	79.2	269.9	29.3
Final[b]	37.3	83.9	86.9	91.6	550.1	45.9

[a]Evaluated on our original 1000-word test set.

[b]Evaluated on a disjoint 1000-word test set.

- **Stress:** The stress symbols /</, />/, and /0/ were replaced by the stress symbol /-/. This leaves only 3 stress symbols, so they were encoded as 3 binary features.
- **Stressed** (same as Extended Context)
- **Letter** (same as Extended Context)
- **Consonant** and **Non-Low-Vowel** (same as Extended Context)

The output was encoded using a 127-bit error-correcting output code with the Combined Phoneme/Stress method. The performance of this method is given in Table 10. We can see that the modifications introduced above produced a very large improvement in performance.

4 Evaluation and Comparison with DECtalk

4.1 Methods

DECtalk is an English text-to-speech device marketed by Digital Equipment Corporation. It contains hardware for synthesizing speech and software for mapping English words into strings of phonemes and stresses. At the heart of the software is a human-constructed rule base of letter-to-sound rules. This is augmented by a dictionary of exceptions—words that the rules mispronounce. The purpose of this section is to describe an experiment comparing the performance of our best machine learning system with the performance of the DECtalk rule base.

We trained our final machine learning configuration on all 19,002 words in the NETtalk dictionary that are not part of our test set. Tony Vitali of Digital Equipment Corporation graciously agreed to run our 1000-word test set through the letter-to-sound rules for DECtalk version 3.0. So for each word in our test set, we had the DECtalk pronunciation and the ID3 pronunciation. In addition, we had the NETtalk dictionary pronunciation.

We performed an experiment with the help of three human judges. The experiment was designed as follows. First, the pronunciations produced by ID3 and by the NETtalk dictionary were converted into phoneme and stress strings suitable for playing through the DECtalk machine. This was performed using a program call KLATTIZE written by Charles Rosenberg and distributed by Thinking Machines Corporation. If the pronunciations produced by ID3 or DECtalk exactly matched the pronunciation computed from the NETtalk dictionary, then the word was automatically considered correct.

Otherwise, the judge was asked to compare the pronunciation produced by the DECtalk rules and by ID3 with the NETtalk dictionary pronunciation. The judge was first shown the spelled word. Then the judge was played the NETtalk dictionary pronunciation and then one of the proposed pronunciations. Then the judge was asked to give one of the following responses:

1. There is *no* difference between the two pronunciations,
2. There is a *noticeable* difference between the two pronunciations,
3. There is a *serious* difference between the two pronunciations, or,
4. The proposed pronunciation is better than the dictionary pronunciation. (Therefore, count the proposed pronunciation as correct.)

The order of the DECtalk and ID3 pronunciations was randomized so that the judge did not know which pronunciation was produced by which method.

4.2 Results

Each of the three judges performed this procedure independently (and on separate occasions). A response of 4 was considered equivalent to a response of 1, because both responses indicate that the word is correctly pronounced. The responses can be summarized by a three-digit string, such as "111" (all three judges gave a rating of 1), "112" (two judges rated 1 and one judge rated 2), and so on. Table 11 gives statistics showing the reliability of these judgements. The full experiment involved comparing two other machine learning systems as well as DECtalk and the final ID3 configuration described above, and this table reflects all of those judgements.

The results are encouraging. Out of the 494 responses that were classified as (1), (i.e. no difference), 426 (86.2%) were by a unanimous decision. Similarly, out of the 871 responses that were classified as (3), (i.e. serious difference), 755 (86.7%) were by a unanimous vote.

The situation, however, is not so clear cut for responses classified as (2), (i.e. noticeable difference). Only 505 of the 723 such cases (69.9%) were by a unanimous decision. The rest (218 cases), were decided by a vote of two out of three. In 131 out of these 218 cases (60.1% of the time), the third judge who disagreed thought that difference was serious rather than noticeable, while in the remaining (39.9% of the 218 cases) the judge who disagreed thought that difference was negligible.

It is interesting that there were *no* cases of the "113" response (two judges responding with "1", the third with a "3"), only 2 cases of "331", and only one case of a "123" response (which was later changed to a "222"). Furthermore, all three judges agreed on 1,686 responses out of 2,088 (80.7%).

Table 11: Breakdown of the individual subject responses when evaluating correctness at the word level. Figures shown reflect the cumulative responses for all four systems compared.

Majority Response	Individual Responses*	Actual Count	% of Total	% of *all* 2,088 Responses
(1) No difference:				
1	111	426	86.2%	
1	112	68	13.8%	
1	113	0	0.0%	
	Total:	494	(100%)	23.66%
(2) Noticeable difference:				
2	221	87	12.0%	
2	222	505	69.9%	
2	223	131	18.1%	
	Total:	723	(100%)	34.63%
(3) Serious difference:				
3	331	2	0.2%	
3	332	114	13.1%	
3	333	755	86.7%	
	Total:	871	(100%)	41.71%

*Example: 332 means two subjects responded with 3, and the third responded with 2.

Table 12: Percentage of whole words correct as judged by majority vote of three human judges. A response of 2 indicated "Noticeable Difference". A response of 3 indicated "Serious Difference"

	Majority Response	
System	2 or 3	3
DECtalk rule base	55.7	73.9
ID3 (final configuration)	71.2	83.3

These observations suggest that the results of the experiment are fairly robust against variations caused by differences between individual human judges.

The final results from the human judges are shown in Table 12. We can see that final ID3 configuration scores much better than the DECtalk rule base.

4.3 Discussion

The experiment provides clear evidence that the machine learning approach described in this paper gives better results than the DECtalk rule base. However, there are two points to note, which suggest that these results should be interpreted with caution.

First, the error rates reported by our judges for DECtalk were higher than those reported elsewhere. It may be that our experimental setup, which required direct comparisons against a dictionary pronunciation was responsible for this, since the dictionary pronunciation is implicitly regarded as correct (even though we gave subjects the option of declaring the proposed pronunciation to be better).

This raises a second point. The ID3 system has been trained on other words from the NETtalk dictionary, so it can be expected to be better at mimicking the dictionary even if there are alternative legitimate pronunciations. If DECtalk produced those other pronunciations and they were "seriously" different from the NETtalk dictionary, then our study would count those as DECtalk errors.

The third point to note is that most of the experiments we performed during the design process used the same test set that was evaluated by the judges. Hence, information from this test set could have contaminated the learning process and yielded an overly optimistic rating for the ID3 pronunciations. In short, we may have "overfit" the test set because of our extensive design experiments.

There are three arguments against this third point. First, given the size of the test set (1000 words, or 7,242 letters), overfitting not very likely. Larger data sets are more difficult to overfit. Second, we replicated several of our design experiments on a different independent test set and obtained very similar results. Third, the authors of the DECtalk rule base probably also made use of some words from our test set when they were writing and debugging their rules. One advantage of computers is that you can explicitly train them on a fixed set of words and keep them ignorant of all other words in English. This is not possible with people. So in this sense, the DECtalk ratings are also potentially optimistic.

Despite these shortcomings in the experimental procedure, it is clear that our final ID3 system performs quite well. When evaluated according to how well it can exactly match the NETtalk dictionary pronunciations, our final system (trained on 19,002 words) correctly pronounces 64.8% of words, 91.4% of letters, 93.7% of phonemes, and 95.1% of stresses.

5 Conclusions

The task of mapping English to speech is very difficult, because of the multiple sources of words in the English language and the shifts in pronunciation over the years. Because of this complexity, the text-to-speech task is a good one for the development and testing of machine learning algorithms. The most important outgrowth of our experiments was the discovery and development of the error-correcting output coding method. We have demonstrated that this is a general method that can be applied to many application problems.

This paper has described a machine learning approach that combined several general techniques (error-correcting output codes, sliding windows, and extended context) with some application-specific methods (special input features and the right-to-left scan). The result is a system that does a very good job of mapping English words to phonemes and stresses. An experimental evaluation by human judges provides evidence that this machine learning approach performs better than the hand-written rules in the DECtalk system. We hope that the techniques described here will be useful for other groups working on text-to-speech mapping and for researchers studying similar difficult problems.

References

Bakiri, G. (1991). Converting English text to speech: A machine learning approach. Tech. rep. 91-30-2, Department of Computer Science, Oregon State University, Corvallis, OR.

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

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

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123–140.

Dietterich, T. G., & Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, *2*, 263–286.

Dietterich, T. G., Hild, H., & Bakiri, G. (1995). A comparison of ID3 and backpropagation for English text-to-speech mapping. *Machine Learning*, *18*, 51–80.

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

Digital Equipment Corporation (1985). Dectalk dtc01 owners manual, third edition. Tech. rep. EK-DTC01-OM-003, Digital Equipment Corporation, Maynard, MA.

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In Saitta, L. (Ed.), *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 148–156 San Francisco, CA. Morgan Kaufmann.

Hocquenghem, A. (1959). Codes corecteurs d'erreurs. *Chiffres*, *2*, 147–156.

Lin, S., & Costello, D. J. J. (1983). *Error Control Coding: Fundamentals and Applications.* Prentice-Hall, Englewood Cliffs, NJ.

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

Lucassen, J. M., & Mercer, R. L. (1984). An information theoretic approach to the automatic determination of phonemic base forms. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, ICASSP-84*, pp. 42.5.1–42.5.4.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess endgames. In *Machine learning: An artificial intelligence approach*, Vol. 1, pp. 463–482. Tioga Press, Palo Alto, CA.

Quinlan, J. R. (1996). Bagging, boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 725–730 Cambridge, MA. AAAI Press/MIT Press.

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

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E., McClelland, J. L., & the PDP research group. (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1: Foundations.* MIT Press.

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

Wettschereck, D., & Dietterich, T. G. (1992). Improving the performance of radial basis function networks by learning center locations. In Moody, J. E., Hanson, S. J., & Lippmann, R. P. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 4, pp. 1133–1140. Morgan Kaufmann, San Francisco, CA.