Alignment Algorithms for Learning to Read Aloud

Charles X. Ling **Handong Wang**
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada N6A 5B7
E-mail: {ling, hwang}@csd.uwo.ca

Abstract

A complete system of learning spelling-to-phoneme conversion of English words consists of three major processes: alignment, mapping learning, and grapheme generation. Such a system can be used to construct prototypes of reading machines for English or other languages quickly and automatically. This paper focusses on the alignment process, which is critical to mapping learning and grapheme generation. We present several novel alignment algorithms which learn alignment without supervision. The basic alignment algorithm is a hill-climbing algorithm. Several improvements of it are studied and tested. In addition, a method that overcomes the pitfall in hill-climbing algorithms is designed. Our best alignment algorithm produces very impressive results: only 0.5% of error rate.

1 Introduction

Reading English text aloud has been studied successfully for many years with numerous laboratory systems and some commercial systems (see, e.g., Allen, 1976; Allen, Hunnicutt, & Klatt, 1987; Kurzweil, 1976; Klatt, 1982, 1987). In this paper, we focus on only one aspect of reading aloud: isolated word text-to-phoneme conversion (ignoring visual recognition, text analysis, intonation and stress analysis, speech synthesis, and so on). Our attention is on automated learning of text-to-speech conversion, rather than, for example, conversions specified by manually designed rules. Our learning system can be used to construct automatically in a very short time prototypes of reading machines for English or other languages.

Given a set of words, each with an orthographic representation (spelling) and a phonological representation (pronunciation), the basic learning task is to learn a mapping from the orthographic representation to the phonological representation, and to predict the pronunciation of unseen words with a high accuracy. A natural approach to model the *complete* learning process of single-word spelling-to-phoneme conversion requires three major steps (see Section 3 for other approaches and their weaknesses). The first step is to align the orthographic representation with the phonological representation. The second step is to learn the mapping from the orthographic to the phonological representations, and the last step is to generate graphemes. These three processes are intimately tied together, and are very complicated to model. Most previous work models only one of the three processes. For example, models by Sejnowski and Rosenberg (1987), Seidenberg and McClelland (1989) and Plaut, McClelland, Seidenberg, and Patterson (1996) only deal with the mapping learning task; the alignment was done manually.

We assume that the starting point of orthographic representation is 26 letters (plus two marks for the beginning and ending of the word). The phonological representation, on the other hand, is a small set of about 40 standard phonemes as the sound building blocks for English. Below are some examples of letter-to-phoneme mappings of single words: `speech` → `spEtS`, `thought` → `T*t`, and `thrill` → `Tril`.

The first task of aligning the orthographic representation with the phonological representation is necessary because often n letters in spelling of a word maps to m phonemes in pronunciation with $m < n$.[1] For example, the word `thought`, which has 7 letters in the orthographic representation, has only 3 phonemes, `T*t`, in the particular phonological (phoneme) representation that we use.[2] Therefore, the mapping from the orthographic to the phonological representations is not one-to-one.[3] To make the second (learning mapping) and

[1] There are four cases where a single letter maps to more than one phoneme. They are `x` as in `box` (maps to `ks` in `boks`), `j` as in `just` (maps to `dz` in `dz∧st`), `o` as in `one` (maps to `w∧` in `w∧n`), and `u` as in `fuel` (maps to `yU` in `fyUl`). To simplify learning, these "macro" phonemes are replaced by single letters not used in the original phonological representation.

[2] In the phoneme string `T*t`, `T` represents the sound `th` in `thought`, `*` represents the sound `ough` in `thought`, and `t` represents the sound `t` in `thought`.

[3] In Sejnowski and Rosenberg's NETtalk, a silent phoneme is inserted so the alignment is done before learning. Taking the mapping `thrill` → `Tril` as an example, it would be `thrill` → `T_ril_`; so the mapping is always one-to-one in

third tasks (grapheme generation) possible, we have to properly align letters with phonemes so that the learning programs (and children) know which letter (or letter combination) maps to which phoneme. For `thought`, the first alignment below is correct,[4] while the second and third are not.

```
thought     thought     thought
T_*___t     T*t____     T*___t_
```

Clearly, there is a total of $\binom{7}{4} = 35$ different ways of inserting 4 spaces in `T*t`, or 35 ways of aligning `thought` with `T*t`.

The second task is to learn a complicated mapping from letters to phonemes from pairs of letter strings to phoneme strings which *have been aligned.* Such a mapping is only *quasi-regular* with many exceptions, and is more complicated than some other language-learning tasks, such as the mapping from verb stems of English verbs to their past tenses. The mapping learning has been studied with symbolic learning algorithms (e.g., Dietterich, Hild, & Bakiri, 1990) and connectionist learning algorithms (e.g., Sejnowski & Rosenberg, 1987; Seidenberg & McClelland, 1989; Plaut et al., 1996).

It is important to realize that the result of alignment directly affects mapping learning: *Each possible alignment combination represents one potentially possible mapping to be learned.* As mentioned, the mapping is only quasi-regular, with many exceptions. It could be that, if `thought` is aligned with `T*t____`, `t` maps to `T`, `h` maps to `*`, `o` maps to `t`, and the rest of the letters `ught` map to a blank phoneme — *this might be a legitimate mapping* to be learned. That is, a bad alignment also constitutes a mapping. Since many words in the training set have more than one alignment (e.g., `thought` has 35), the combination of possible alignments of all the words in the training set is huge, but each represents one potentially possible mapping to be learned. As an example, the data set we use in our study originated from (Seidenberg & McClelland, 1989), and it contains a total of 2998 words, and the combination of all of these alignments of the corpus is estimated to be over 12,000. The question is, with so many possibilities, how can a learning program learn the correct alignment of all words *and* mapping based on the newly-learned alignment effectively?

Alignment should be included as a part of the learning task, instead of being manually derived as in most previous work. However, while mapping learning after alignment is supervised, alignment learning is unsupervised. This work solves this critical problem. Lawrence and Kaye (1986) designed a stand-alone alignment algorithm, but it is not a learning method (see Section 3 for more details).

NETtalk.

[4]There are other correct alignments of `thought` with `T*t`, such as `_T*___t` or `_T_*__t`. In general, as long as `T` is aligned with one letter in `th`, and `*` with one letter in `ough`, the alignment is regarded as correct.

The task of performing *simultaneously* pattern alignment and pattern learning exists in other areas of language learning as well, such as reading continuous text, word matching in long stretches of speech (Pinker, 1994, page 267), and meaning matching of words in speech (Pinker, 1994, page 153). The alignment algorithms presented in Section 2 should be applicable to other problems of this type.

The alignment process is *crucial* since the correct alignment determines the suitable mapping learning and proper grapheme generation. We have built a decision-tree learning system that models all of these three processes. However, due to limited space and the complexity of the system, we describe only the alignment algorithms in sufficient detail in this paper.

2 Learning Mappings with Automatic Alignment

In this section, we present several alignment algorithms that utilize C4.5 as the mapping learning algorithm. C4.5 (Quinlan, 1993) is one of the most popular machine learning algorithms. We first discuss the representation issues in the task of reading aloud, and then describe several algorithms for learning mapping and alignment simultaneously.

2.1 Representation Issues

The representation of learning to read aloud used in this paper is similar to that in the NETtalk (Sejnowski & Rosenberg, 1987) — an N-to-1 sliding-window representation, where N is usually called the *window size* which determines how far the neighbouring letters that may be used in the decision-making process. Siding-window representation converts the mapping between n letters to m phonemes into N-to-1 mappings; that is, it becomes a classification problem. Thus, only one phoneme is predicted at a time from the letter at the center of the window and its left and right neighbouring letters. The window slides over the letters of the word and predicts the corresponding phonemes one by one. In this paper, we choose the window size as 11 which is enough for the data set we have. Thus, a phoneme is predicted based on the middle letter, 5 left neighbouring letters and 5 right neighbouring letters.

The classifier learns a hypothesis, and uses it to predict all phonemes (including the blank phoneme _) one by one when given the spelling of a new word for testing. More specifically, one phoneme is predicted from the corresponding letter and 5 left and 5 right neighbouring letters at a time, and these phonemes are concatenated to be the phoneme string of the word (with the blank phoneme removed). If one or more phonemes are predicted incorrectly, the whole word is regarded as predicted incorrectly.

We use the most popular machine learning classification algorithm C4.5 (Quinlan, 1993) with its default parameter setting as our mapping learning mechanism. C4.5 is an improved implementation of the ID3 learning algorithm (cf. Quinlan, 1986). C4.5 induces classification rules in the form of decision trees from a set of classified examples. It uses information gain ratio as a criterion for selecting attributes as roots of the subtrees. The divide-and-conquer strategy is recursively applied in building subtrees until all remaining examples in the training set belong to a single concept (class); then a leaf is labeled as that concept. The information gain guides a greedy heuristic search for the locally *most relevant* or *discriminating* attribute that maximally reduces the entropy (randomness) in the divided set of the examples. The use of this heuristic usually results in building *small* decision trees instead of larger ones that also fit the training data.

2.2 The Basic Alignment Algorithm

As we discussed in Section 1, learning mapping while aligning words is a challenging task, alignment learning is unsupervised, since each possible alignment represents a potentially possible mapping to be learned, and there is a huge number of possible alignments (12,000 in our data set). The key idea in solving this difficult problem is that most of the 12,000 mappings do not contain much regularity at all. That is, if many words are aligned incorrectly or inconsistently, there is little regularity to be learned, and it becomes almost impossible to predict the phonemes of a new word. Therefore, our basic alignment algorithm is based on the fact that the proper alignment should be consistent among words, and the prediction based on aligned words should be consistent with the correct alignment of the new word.

The basic alignment algorithm is a hill-climbing algorithm that gradually builds up the set of aligned words. From a set of words that have already been aligned (we call this set a *converged set*), a decision tree is built using C4.5, and it is used to choose the best alignment of an unaligned word. The best alignment is then added into the converged set. More specifically, an unaligned word from the *unconverged set* (containing all unaligned words) is aligned in the following way: whenever the new word has more than one possible alignment (that is, the word is an $n \rightarrow m$ mapping with $n > m$), a prediction of the word using the decision tree built on the current converged set (of aligned words) is produced first. As we discussed earlier, the prediction based on aligned words should be consistent with the correct alignment of the new word. The prediction is thus compared with all possible alignments of the word, and the alignment most consistent[5] with the prediction is chosen as the correct one. The chosen alignment, which hopefully is correct, is then added into the converged set, the decision tree is updated[6] with the inclusion of the newly aligned word, and the process is repeated. As the set of the aligned words increases, the decision tree algorithm learns more varieties of mappings from letters to phonemes, the alignment of the new words becomes more and more accurate. The basic alignment algorithm is presented in Table 1. It is very similar to the one proposed by Bullinaria (1994) for the connectionist model, except that the difference between two words is calculated on the output units in his model.

```
Conv = empty    (* Converged set, empty to begin with *)
Unconv = list of training examples   (* Unconverged set *)
repeat
  take one word w from Unconv
  If w is n -> n        (* no alignment is needed *)
    then add w to Conv, delete w from Unconv
         update the tree T based on Conv
    else insert space in w at different places
           obtaining possible alignments w_1, w_2,...
         use T to predict w, the prediction is u
         compare u with w_i, let e_i = difference(u, w_i)
         let e_k = min(e_i)
            w_k is chosen as the correct alignment of w
            insert w_k to Conv, delete w from Unconv
            update the tree T based on the new Conv
until the Unconv set is empty
```

Table 1: The basic alignment algorithm.

Let us see an example. If we have a set of words that have been properly aligned, and we want to align a new word `speech` mapping to `spEC`. Since this is a $6 \rightarrow 4$ mapping, there are 10 ways of inserting two blank phonemes in `spEC`, or 10 possible alignments. For example, `__spEC`, `s_p_EC`, `spE_C_`, and so on. Obviously, the third one is the correct alignment. If the set of aligned words (in the converged set) contains words with letters to phoneme mappings for `s`, `p`, `ee`, and `ch`, then the prediction of `speech` from the current decision tree would be `spE_C_`, which is correct. In this case, the prediction of the new word is the same (consistent) as one of the possible alignments, hence it (i.e., `spE_C_`) is taken as the correct alignment and is added into the converged set of the aligned words. However, when the current training set does not contain enough varieties of words, the prediction may not be entirely correct. For example, the prediction of `speech` from the current decision tree could be `spE_kh`, i.e., the `ch` part has not been learned yet. Still, we find `spE_C_` (or `spE__C`) among other 10 possible alignments is closest to the prediction `spE_kh`, since there are only two errors (the last two phonemes) between them; while there are, for example, four errors between `spE_kh` and `s_p_EC`. In this case, we take the best alignment, still `spE_C_` (or `spE__C`), as the correct alignment, and add it into the converged set of aligned words. This time, the chosen alignment is correct.

[5]The consistency between two words (i.e., prediction and alignment) is determined simply by the number of different phonemes in the two words at the corresponding positions.

[6]Currently C4.5 is applied to the enlarged training set directly. However, ID5 (Utgoff, 1989) could be used and the decision tree would only be updated. The resulting decision tree would be equivalent to the one obtained by applying C4.5 (ID3) on the enlarged training set.

2.3 Improvements on the Basic Alignment Algorithm

We found that the basic alignment algorithm makes an excessive number of misalignments: the error rate[7] is over 10%. Thus, we study several extensions and improvements of it in the following subsections. These improvements include incorporating a tie breaking policy, ordering the words from easy to complex, employing a conservative criterion for accepting aligned words, and correcting previously misaligned words.

Tie Breaking Policy

In an initial implementation of the basic alignment algorithm, we found that ties often occur: several possible alignments have the same closest distance to the prediction produced by the decision tree. (In the example of the previous subsection, two alignments `spE_C_` and `spE__C` tie with the prediction `spE_kh` with two errors). Breaking tie randomly ends up with many incorrect and inconsistent alignments. A tie breaking policy is introduced: when there is a tie in alignment, the word is put back at the end of the current list of unconverged words. That is, if there is no *unique* best alignment, the decision is delayed until more knowledge of alignment is learned.

When the alignment algorithm (with the tie breaking policy) takes words from the training set in a random order, it still produces quite a few misaligned words — a total of 116 misaligned words among 2998 words in the training set.[8] The reason for this is that before a large body of alignment knowledge is accumulated, the prediction of a more complex word (such as an $n \rightarrow n-3$ word) contains many errors. In this case, the difference between the prediction and every possible alignment of the word is large; thus, even if the best alignment *is* unique, it is often incorrect. Misalignments in the converged set spread — they cause, in turn, more misalignments in the training set.

Learning from Easy to Complex

Clearly, if a word is an $n \rightarrow n$ mapping, then there is only one possible alignment. The word can be added directly into the converged set for learning the mapping from letters to phonemes. Therefore, $n \rightarrow n$ words are regarded as "easier" words in learning alignment than $n \rightarrow n-1$, than $n \rightarrow n-2$ words, and so on. Hence, if $n \rightarrow n$ words are learned first, then the predictive accuracy of $n \rightarrow n-1$ words would be high, since the mapping of all phonemes except one (to which two letters map) have likely been learned already in $n \rightarrow n$ words. In this case, the correct alignment will more likely be found.

The implementation of this improvement algorithm is the same as the one in Section 2.3 except the initial list of Unconv contains words ordered from easy to complex. That is, $n \rightarrow n$ words go first, $n \rightarrow n-1$ words next, and then $n \rightarrow n-2$ words, $n \rightarrow n-3$ words, and $n \rightarrow n-4$ words. The number of misaligned words produced by the algorithm is much smaller; only 30 among a total of 2998 words (only about 1% of error). Some examples of misaligned words are `breast` $\rightarrow$ `bres_t` (should be `bre_st`), `shed` $\rightarrow$ `Se_d` (should be `S_ed`), `says` $\rightarrow$ `sez_` (should be `se_z`), and `shall` $\rightarrow$ `S__al` (should be `S_a_l` or `S_al_`). However, some mistakes are consistently made for several words (such as `days`, `jays` and `says`). This indicates again a pitfall in the hill-climbing algorithms — since there is no backtracking mechanism for correcting previously made mistakes, mistakes are likely to spread further.

Learning More Conservatively

Another improvement over the basic alignment algorithm described in Section 2.2 (which takes words in random order) is to adopt a conservative policy that restricts words accepted into the converged set. This conservative policy reflects the idea that, if the current learned knowledge does not produce an answer close enough to the correct one, its alignment decision is delayed until more knowledge of alignment is learned. As we have noted, if the difference between the prediction and every possible alignment of a word is large, it signals the possibility that not enough alignment is learned, and that the alignment decision on this word may be premature. This conservative policy puts a bound on such a difference; only when the difference between the prediction and the best alignment is within the bound, is it chosen as the correct alignment. This bound is increased gradually (from 1 to 5).

The results of this algorithm (which takes words in random order) are very impressive. The number of alignment mistakes is very small; among a total of 2998 words, there are only 14 misaligned words. This represents less than 0.5% of error rate in alignment. Again, since this is an unsupervised learning task, the error rate is the testing error rate, instead of the training one.

The misaligned words by our algorithm are listed in Table 2. As we can see, several types of errors occur consistently in more than one word. Five words have an alignment error in `iff` part, three words in `ays` part, and two words in `ugh` part. This again points out the pitfall in the hill-climbing strategy used in all of these alignment algorithms. In the next section we present a method of overcoming this pitfall.

Correcting Misalignments

From the results of the misaligned words in the previous subsections we observed one phenomenon: certain mistakes occur consistently among several similar words (e.g., `days`, `jays`, and `says`). This reflects the pitfall of

[7] Since no "teacher" provides correct alignment to the learning program, this is essentially an unsupervised learning task. The error rate here is thus the testing error rate instead of the training error rate in supervised learning.

[8] These 116 misaligned words are too many to list in the paper. Note that there are normally several correct alignments for a word, and the 116 words are those certainly misaligned. Also note that misalignment does not necessarily imply an incorrect prediction. However, a training set with many misaligned words constitutes a much more complicated mapping, and thus, predictive accuracy of new words would be lowered.

Table 2: Misaligned words; training with a conservative policy. Only one correct alignment is listed for each word.

Spelling	Phoneme (output)	Phoneme (correct)
skiff	sk_if	skif_
cliff	kl_if	klif_
sniff	sn_if	snif_
stiff	st_if	stif_
whiff	w__if	w_if_
says	sez_	se_z
grays	grAz_	grA_z
ways	wAz_	wA_z
laugh	laf__	la_f_
draught	draf__t	dra_f_t
trough	tr*f__	tr*_f_
fixed	fiXt_	fiX_t
thee	DE__	D__E
loose	lUs__	lU_s_

the general hill-climbing algorithm — there is no backtrack mechanism. Wherever for some reason a mistake is made at an early stage, it is likely to propagate, and no correction is performed. Admittedly, mistakes should be allowed, for this is part of the life in human learning as well. People, however, after learning more knowledge, often realize mistakes made earlier and correct them. We present a *correction algorithm*, which corrects alignment mistakes made in the hill-climbing algorithms.

The correction algorithm takes as input the list of aligned words (output containing misalignments from one of the previous alignment algorithms). Since the knowledge of alignment can now be built on a large number of aligned words and thus is more complete, mistakes (misalignments), especially early mistakes based on a small set of the converged words, may now be corrected. The correction of such words may help in correcting other misaligned words.

Given a list of aligned words ordered by the output of the alignment algorithm, the correction algorithm takes out the first 10% of the words (those aligned very early when little about alignment and mapping had been learned), learns the mapping based on the remaining 90% of the words, and re-aligns that first 10% of the words. That is, words aligned earlier are more likely to have alignment mistakes, and they are re-aligned by the large number of more recently aligned words, in the hope that some earlier, premature mistakes can be corrected. Those 10% re-aligned words are then added to the end of the list, and the next 10% of the words are taken out for correction (by the rest of the words, including previously re-aligned words). After one round of 10 correction sessions, every word in the list has been re-aligned once. The correction algorithm terminates if no correction has been made after one round.

We apply the correction algorithm to the list of the aligned words from the algorithm with the tie breaking policy in Section 2.3 with a total of 116 incorrectly aligned words,[9] and the result is presented in Table 3. Numbers listed under the column "Correction" in the table represent the numbers of words that have been corrected. That is, they are misaligned before correction, and properly aligned after correction (such as `boss` → `b_os` before, `boss` → `bo_s` after). Numbers listed under the column "Miscorrection" represent the mistakes made by the correction algorithm. These are the words that are properly aligned before correction, but misaligned after correction (such as `hill` → `hi_l` before, `hill` → `h_il` after). This happens because the misaligned words in the 90% of the words can affect the re-alignment of the 10% of the words. Numbers under the column "Improvement" are simply the difference between "Correction" and "Miscorrection". They represent the net improvement accomplished by the correction algorithm. There are several other outcomes of the correction algorithm: both words (before and after correction) are correct, both words are correct but the alignments are different, or both words are incorrectly aligned. However, these results are not reflected in the table since they do not affect the net improvement of the correction algorithm.

Table 3: The outcomes of the correction algorithm.

	Correction	Miscorrection	Improvement
1st 10%	13	1	12
2nd 10%	5	2	3
3rd 10%	2	1	1
4th 10%	3	4	−1
5th 10%	2	2	0
6th 10%	3	0	3
7th 10%	3	1	2
8th 10%	2	2	0
9th 10%	11	6	5
10th 10%	9	1	8
Total	53	20	33

From Table 3 we can see that, in general, the number of corrections is high for the early part of the list (especially the first 10%). This confirms our expectation that early learning is less mature and more prone to errors. Overall, there is a marked improvement after one round (10 correction sessions) of correction (i.e., each word is re-aligned once). The net improvement is 33 for the first round. In the second round (details not shown here), the total number of "corrections" is 12, and "miscorrections" is 2. Overall, there are 43 (33+10) improvements after two rounds, thus reducing the total number of misaligned words from 116 to 73, a 37% reduction.

[9] Results from subsequent improvements contain too few misalignments to show the effect of the correction algorithm.

3 Relation to Past Work

Much research in text-to-speech conversion has been done with a few commercial systems (see Klatt, 1987, for an excellent review). However, most commercial systems are not based on the learning or automated knowledge acquisition that we study in this paper. Our learning system can be used to construct prototypes of reading machines for English or other languages quickly and automatically.

Some past work on learning to read aloud came from the connectionist researchers. Sejnowski and Rosenberg (1987) first designed a connectionist model for text-to-speech conversion, but their model only solved the mapping learning problem — it did not deal with the alignment problem and grapheme generation. Phonemes of words had already had a special symbol inserted by hand for the silent phoneme. We adopted the sliding-window representation from NETtalk in our system.

Bullinaria (1994) recently extended Sejnowski and Rosenberg (1987)'s NETtalk by adding an alignment algorithm. Our basic alignment algorithm described in Section 2.3 is inspired by his method. However, the basic alignment algorithm produces an excessive number of misalignments. We have improved it in several directions (see Section 2.3). The error rate of our best alignment algorithm using the same dataset as his is very low (0.5%). Bullinaria's model does not deal with grapheme generation.

Lawrence and Kaye (1986) designed a stand-alone alignment algorithm, but it is not a learning method. A table of phonological-to-orthographic correspondences is designed by hand and given to the alignment algorithm. The table is actually quite large — it has 592 entries. When testing this method on 33,121 words, 347 words were misaligned, representing an error rate of 1.05%. Our alignment algorithms learn the alignment without supervision.

The decision-tree learning algorithm ID3 had been applied to the NETtalk data previously (Dietterich et al., 1990), but the method was applied to the NETtalk data set, and thus the alignment problem was not studied.

4 Conclusions

We describe several methods for aligning letters with phonemes. Alignment is critical to the mapping learning and grapheme generation. Our best alignment algorithm produces very impressive results: less than 0.5% of a total of 2998 words are misaligned. We also discuss a correction method for correcting previously misaligned words. The idea can be used in other hill-climbing search algorithms to improve their results. In future, we plan to use our method to construct prototypes of reading machines for other languages.

Acknowledgments

We like to thank gratefully John Bullinaria for providing with us the data set used in his study, and for numerous discussions on the topic. The data set originally came from (Seidenberg & McClelland, 1989). Discussion with David Plaut has also been helpful. Reviewers also provided useful suggestions to the paper.

Reference

Allen, J. (1976). Synthesis of speech from unrestricted text. In *Proc. IEEE 64*, pp. 422–433.

Allen, J., Hunnicutt, S., & Klatt, D. (1987). *From Text to Speech: the MITalk System.* Cambridge U.P., Cambridge, UK.

Bullinaria, J. (1994). Representation, learning, generalization and damage in neural network models of reading aloud. Submitted to *Psychological Review.*

Dietterich, T., Hild, H., & Bakiri, G. (1990). A comparative study of ID3 and backpropagation for English text-to-speech mapping. In *Proceedings of the 7th International Conference on Machine Learning*, pp. 24–31. Morgan Kaufmann.

Klatt, D. (1987). Review of text-to-speech conversion for English. *Journal of the Acoustic Society of America, 82*(3), 737–793.

Klatt, D. (1982). The Klattalk text-to-speech system. In *Proc. Int. Conf. Acoust. Speech Signal Process. ICASSP-82*, pp. 1589–1592.

Klatt, D. (1987). How Klattalk became DECtalk: An academic's experiences in the business world. *Speech Tech, 87*, 293–294.

Kurzweil, R. (1976). The Kurzweil reading machine: A technical overview. In Reden, M., & Schwandt, W. (Eds.), *Science, Technology and the Handicapped*, pp. 3–11.

Lawrence, S., & Kaye, G. (1986). Alignment of phonemes with their corresponding orthography. *Computer Speech and Language, 1*, 153–165.

Pinker, S. (1994). *The Language Instinct.* William Morrow and Company, Inc.

Plaut, D., McClelland, J., Seidenberg, M., & Patterson, K. (1996). Understanding normal and impaired word reading: Computational principles in quasi-regular domains. *Psychological Review, 103*, 56 – 115.

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

Quinlan, J. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann: San Mateo, CA.

Seidenberg, M., & McClelland, J. (1989). A distributed, developmental model of word recognition and naming. *Psychological Review, 96*, 523–568.

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

Utgoff, P. E. (1989). Incremental induction of decision trees. *Machine Learning, 4*, 161 – 186.