Automatic Letter-To-Sound Rules for Speech Synthesis

Steve Hanov

David R. Cheriton School of Computer Science
University of Waterloo

August 13, 2007

1 Introduction

A necessary first step in text to speech synthesis is deriving the pronunciation of words. Most TTS systems contain a dictionary. In a dictionary based approach, each word in the dictionary has an associated pronunciation, or sequence of phonemes. For example,

department /d ih p aa r t m an n t/;

Reagan /r ey g ah n/;

However, it is likely that a text to speech processor will encounter a word that is not in the dictionary. There are several existing pronouncing dictionarys available for study. The CMU pronunciation dictionary [1] contains 125,000 words. The Moby lexicon, a part of Project Gutenberg [2] contains 177,00 entries, many of which are compound words.

Both dictionaries are missing domain specific words, such as "quadrature" or "tesseract". Also, it missing words from the animal kingdom, such as "lovebird". A text to speech system must either supplement its dictionary with other methods, or use a rules-based approach to automatically derive the pronunciation.

In this paper, we outline ways of developing letter-to-sound rules from sample text, and then present the results of a partial implementation.

2 Prior Work

The development of letter-to-sound rules has been evolving from a fully manual process to current methods, which may be automatic or require some human assistance.

It is natural to think that the pronunciation of words can be governed by a finite set of simple rules. In early grade school, children are taught a few heuristics as they to read and write. For example, *c* makes the same sound as

English											Phoneme	Example Word
.	.	.	.	.	v	o	.	.	.	.	v	**v**oucher
.	.	.	e	s	i	d	e	n	.	.	ə	pres**i**dent
.	.	.	.	w	o	_	.	.	.	.	u	tw**o**
.	.	.	_	h	a	v	e	_	.	.	æ	h**a**ve

Figure 1: An illustration of the method of eliminating redundant context from dictionaries given in [7]

k, but the digraph *ci* and *ce* have the letter 's' sound. Another heuristic is that a short vowel sound is changed to its long form if followed by a consonant and then an *e*, as can be seen in comparing *rat* and *rate*.

In 1973, Ainswroth [3] attempted to write down all of these rules for English in the development of a text-to-speech system. Later, Elovitz *et al.* [8] improved upon them. They developed a notation for codifying the rules, so they could be easily refined by human operators. On their sample of 1000 words from the Brown corpus, they achieved 93.2% accuracy for phoneme translation, but only 65.6% of words contained no errors. As late as 1996, this method remained the most commonly used in commercial TTS systems, [10] which required skilled linguists to develop.

By 1993, machine learning algorithms were being applied to he task. In [11], the present a method of automatically learning grapheme to phoneme mapping rules using a Dynamically Expanding Context (DEC) technique. In this technique, the training data is analyzed to determine how much context is needed for each letter. For example, *t* alone is not enough context to determine a sound. However, the system will learn that *tion* means the *t* is associated with the *sh* sound. Context to the left and right of the letter is are placed into a decision tree structure. This approach is also used in [7], as well as [5]. Algorithms such as these are similar to dictionary approaches. They compress the dictionary into a tree structure and eliminate all redundancy. However, they still present challenges in dealing with word forms not found in the dictionary, and the danger of overtraining reduces the ability of the tree to generalize. This approach is illustrated in Figure 1.

Luk and Damper introduced a sophisticated modern stochastic phonographic transduction technique in [10]. Their work is cited by [5], who uses a much simpler version in the development of the *Festival* text-to-speech system. In this paper, we primarily study the methods of Daelemans [7], Luk [10], and Black [5].

2.1 Relationship to Machine Translation

The automatic learning of letter to sound rules is strikingly similar to the methods of stochastic machine translation, described by Brown [6]. Machine translation relies on having a parallel corpus of bilingual text. First, the sentences are

matched up (for example, using length). In text of different languages, many words will correspond, but be in different positions of the sentence. To find words that are translations of each-other, the frequency of the co-incidences of words in the individual sentences are counted. Words that frequently occur near the same positions in the aligned sentences probably mean the same thing. The probability is estimated iteratively using the Expectation-Maximization algorithm.

Letter-to-sound rules follows a similar technique of building an alignment of letters to phonemes. However, the difference is that the "words" in this case are letter clusters, and there are no spaces between them to identify the where they begin and end. Therefore, a necessary first step is figure out a letter to phoneme alignment.

3 Characteristics of the Corpus

All letter-to-sound rule learning systems share some characteristics. In general, a corpus contains a list of words and their associated pronunciation. The pronunciation consists of a list of phonemes. The set of usuable phonemes can vary depending on the corpus used, even for the same languaage. For example, the CMU pronouncing dictionary has 39 phonemes, while Moby has 50. The difference can be attributed to the many foreign proper names in the latter, that require phonemes such as */ch/* in *Bach* and */WA/* in *Dubois*. It can be predetermined beforehand which phonemes correspond to vowel sounds, and which correspond to consonants. In this project, we use the Moby dictionary. Other researchers frequently use the CMU dictionary or the Oxford English Dictionary (OED).

The pronunciation of proper names is well studied in [9]. Llitjos first classifies them based on their probable language of origin to determine the proper pronunciation. In this project, however, we filter out the proper names by excluding words with capital letters. We also filter out abbreviations such as "dept" by excluding words whose letter cound and phoneme count differ too greatly.

4 Alignment

Before any statistical analysis can be done, the letters must first be aligned with their corresponding phonemes, as in this example:

Letters:	sh	a	ll	ow
Sounds:	S	a	l	o

A variety of alignment techniques has been tried by researchers in this area. Black found that errors in the alignment has a significant effect on the number of errors in the transcription [5].

Two distinct techniques are evident int the literature:

- In the first method, each letter can be aligned with a single phoneme. The other letters which may lie in between the matches are skipped over and not considered.
- In the second method, sequences of letters are aligned with sequences of phonemes. All letters are considered.

The second technique is computationally expensive, and is studied in detail by Luk [10]. However, it is not clear that it is better.

4.1 Bootstrapping

Ideally, all words would be perfectly aligned. Some of the words that cannot be aligned are abbreviations or errors in the dictionary. However, these can be removed once they are found.

The key to the alignment process is to derive the probability function $p(s|l)$ where s is the phoneme (or sequence of phonemes) and l is the letter (or sequence of letters).

There are two cases. If the number of phonemes matches the number of letters, there is only one possible alignment. If the number of letters and number of phonemes differ, then there are many different alignments possible. In English, there are usually more letters than phonemes for a given word, because the language contains many digraphs such as $/sh/$ and $/ch/$. However, a limited number of letters correspond to more then one phoneme: x corresponds to $/ks/$ and u is sometimes $/y - U/$.

A naive alignment method is to ignore the differences and count frequencies of letter/phoneme pairs, starting at the first letter/phoneme and proceeding as far as possible before running out of letters or phonemes. A slightly smarter method, described in [7], uses a sliding window. The frequency of letter/phoneme pairs counted, as in the naive method, but then the phonemes are shifted by one, two and then three positions if possible, obtaining additional counts. In the shifted version, the scores are greatly diminished, so the unshifted version has a strong bias.

Black *et al.* use yet another alignment method. They introduce epsilons (ϵ), so letters or phonemes can be skipped over in the alignment. The training phase consists of finding all possible places to fit the epsilon. In Figure 2, the ideal alignment for *chat* has the h corresponding to the epsilon phoneme.

c	h	a	t
/ch/	ϵ	/æ/	/t/

Figure 2: The alignment of a word, using epsilon scattering.

Dynamic Time Warping can simplify the alignment process. DTW is a general technique for finding similarity between two sequences that are not necessarily the same length. It is used heavily in [5] and [10] for finding alignments

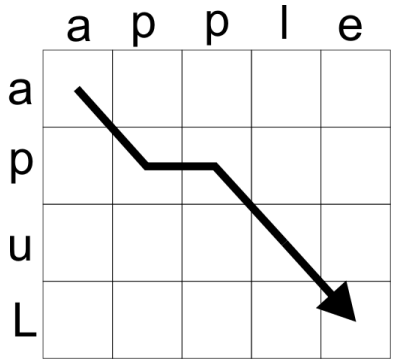

Figure 3: The *Discrete Time Warp* algorithm can find the most probable alignment between letters and phonemes for a word.

between the letter and phoneme representation words. It is a dynamic programming technique to find all possible alignments between single letters and sounds. It can also be used to find the most probable alignment, once the $p(s|l)$ probabilities are known. The technique is illustrated in figure .

In each cell, the algorithm calculates a probability for all possible transitions from other cells. Any given alignment can be found by following the path backwards from the lower right corner of the table.

Luk [10] uses this technique for bootstrapping the alignment probabilities. In the initial pass, transitions are assumed to be able to come from the cells immediately above, from the left, and from the top left, even if the alignment would be impossible (not reaching the bottom right square).

However, we found that a simple recursive method will work in an acceptable time too, if the number of alignments isn't too great.

4.2 Epsilon Scattering Method

In [5], the authors take a unique approach. They only allow single letter to single phoneme pairings. To handle words with fewer phonemes than letters, they allow some letters to map to the special epsilon phoneme. Finding all possible alignments, then, becomes the number of possible ways of placing epsilons between the letters or sounds of the word.

The EM algorithm [4] is a way for estimating hidden parameters of a statistical model. At each step, the parameters are refined in increase the likelihood of the data, given then model and the hidden parameters. In the alignment rules, the hidden parameters are the probabilities $p(s|l)$.

- In the *estimation step,* estimates for the hidden parameters are obtained, either by choosing any values, or preferably, by roughly guessing given the data.

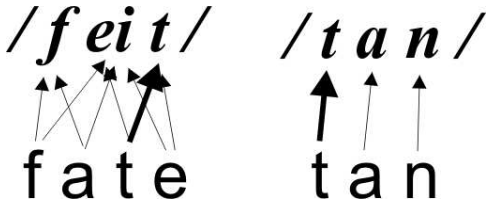

Figure 4: Each time a letter and sound co-occur, it strengthens the probability that the sound results from the letter.

- In the *maximization step,*, new parameters are obtained that maximize the likelihood of the data. With trivial models, one can find the best parameters by taking the derivative to find a local maximum. However, in most cases one uses an iterative approach to find the maximum.

Black uses a variation on this iterative approach. At each iterative step, they use the previous $p(s|l)$ table to find the best possible alignment for all words, while updating $new_p(s|l)$ the new frequency counts. After normalizing, the procedure is repeated using new_p for the alignments., if any of the probabilities changed.

4.3 Comparison of techniques

In [5], Black shows that errors in the alignment can significantly affect the number of errors in the resulting phonetic transcription. However, it is difficult to evaluate an alignment technique. The best thing to do is to compare them to how a human would perform the alignment, but there is no pre-aligned corpus available for testing. Here, we compare the techniques used.

The sliding window method, despite being very simple, works well in our tests. Part of the resulting probabilities are shown in Figure 6. In the majority of cases, it segments words in the same way a human would. It works particularly well with double consonants, as in *a-cc-o-mm-o-d-a-te* However, it is by no means perfect. A notable failure is *ap-p-le*. It appears that the *l* has a low score for the */u/* sound, because it often takes this sound at the end of words, requiring shifts and therefore a diminished score.

According to [5], the epsilon scattering method also did not achieve the same quality of results as a human would. According to the 1998 paper, the alignments of 63% of the words were error free. In the Festival system, Black finally resorts to a "hand-seeded" method, whereby in a manual step, each letter is associated with all possible sounds that it can produce. With the hand-seeded algorithm, 78% of the words are correct.

In out implementation of the algorithm, we found a possible reason for its failure. Unlike sliding window method, in epsilon scattering, there is no particular bias on where to put the epsilons. For example, for the word *chat*, both a human and the sliding window method would align it as *ch-a-t*. However with epsilon scattering this is no reason why it would not associate the c with an

epsilon, making it ϵ-h-a-t. Thus, in some cases it would conclude that h is associated with the */ch/* sound, and in other cases it would use c. This weakens the association between the correct letters and sounds. The more naive sliding window approach has a strong build-in bias for putting scoring letters highly at the first letter of a digraph. This suggests that it may be appropriate to combine the two methods, perhaps seeding the epsilon scattering probabilities with the sliding window technique.

5 Learning Method

At the end of the alignment process, all of the words in the training set are split into chunks, where each chunk corresponds to exactly one phoneme. An example is shown here with the word *high*:

h	igh
/H/	*/AH/*

Figure 5: Alignment of a word containing a trigraph

During the learning phase, the algorithm must analyze the letter/phoneme correspondences to deduce rules. This can be done with either decision tree methods or a probabilistic grammar.

5.1 Decision Trees

In [7], the authors use an Information Gain (IG) tree to derive pronunciation. This is a method based on *analogy* and dynamically expanding context. The IG tree is a trie structure containing the letters of a word, alternating first to the left and then to the right of the letter of interest, and moving farther away until just enough context is included. For example, for the i in reside, *eside* may be enough context to determine the pronunciation. Thus the path through the trie would be *i.s.d.e.e.* The final node would contain a link to the */AY/* sound. For the i in *president*, more context would be included.

This method has the advantage of being relatively easy to implement. By pruning all redundant information from the input dictionary, one hopes that it will generalize to other cases. According to [7], it achieves 85 to 95% accuracy on test words

The authors in [5] also store the knowledge in the dictionary in compressed form. They use a Classification and Regression Tree, which is also type of decision tree. However, any implementation of a decision tree technique method may suffer from over training, and care must be taken to select an appropriate stop parameter. Usually, the parameter is obtained through many trials.

5.2 Probabilistic Grammar

In [10], Luk *et al.* take a completely different approach. They derive a probabilistic grammar whose symbols are letter/phoneme pairings. In addition, the symbols may be sequences of letter/phoneme pairings. The approach is computationally expensive. It consists of three passes.

- During pass 1, the probabilities of single letter to single phoneme pairs are bootstrapped using the naive alignment method.
- During pass 2, the aligned words are further broken down into groups of consonants and vowels matching the pattern C* or VC*. For example, the aligned word (sh,/S/)-(a,/AY)-(pe,/P) would be broken into the separate units (sh,/S/) and (a-pe, /AY-P/). These larger units will form the symbols in the grammar. Note that it is already known which phonemes correspond to vowel-type sounds, so this step is trivial.
- During pass 3, the words are analyzed a third time, and the frequency of the possible transitions between the symbols are analyzed to determine the rule probabilities for the grammar. This can be done as bigrams, where the probability of transitions between every symbol is recorded.

At the end of the process, one has a probabilistic grammar that simultaneously generates both words and the corresponding sequences phonemes and for the language. To determine the pronunciation of a word, the letters of the word word are parsed into the most probable sequence of symbols. Once the representation is found, the pronunciation is the phoneme portion of the symbols. As in the alignment, the most probable parse is obtained using a dynamic programming algorithm (in this case, the well known Viterbi variant).

Luk's algorithm is ambitious, but very computationally expensive. In our own implementation of pass 2, we found that the 170 000 words of the Moby dictionary contain about 10 400 distinct symbols (or vowel-consonant sequences). As part of pass 2, Luk removes redundant sequences that contain other sequences already in the dictionary, but we did not try this.

6 Results

Several different methods of alignment were compared for this project. The effectiveness of the technique is difficult to measure, since there is no existing corpus of aligned words. However, by directly observing the $p(s|l)$ table, one can get a sense of how well an algorithm derives the probabilities. For example, in an ideal model, the letter c should have high probability of resulting in at least /k/, /ch/, or /s/ sounds. In Figure 6, note that the letter l has high probability of matching sound /l/, but also matches sound 'ai', corresponding to the third sound in apple.

In one attempt to derive probabilities, the EM algorithm was applied using an iterative dynamic programming technique to estimate the probabilities of

clusters of 1, 2, or 3 letters to 1 or 2 phonemes at the same time. This was unsuccessful, as the table tended to find only many mappings of 3 letters to 2 phonemes. Intuitively, there should be at least some mappings involving one letter to one phoneme. This model may be more successful if performing one combination at a time (for example, 1 to 1, then 2 to 1, then 2 to 2, etc.). However, after that it is not clear how to combine the probabilities.

Letter/sound	$p(s\|l)$	Letter/sound	$p(s\|l)$
j / g	0.75195	q / k	0.746876
d / d	0.716836	b / b	0.711541
m / m	0.708338	f / f	0.70712
v / v	0.706131	r / R	0.6482
x / k	0.616174	l / l	0.610426
n / n	0.603342	p / p	0.600449
k / k	0.597993	s / s	0.572008
t / t	0.569154	c / k	0.54891
y / ee	0.44466	w / w	0.397284
g / g	0.397114	u / ai	0.357447
i / i	0.342331	a / ai	0.219938
a / ae	0.206886	o / ai	0.204525
g / g	0.192043	h / h	0.180872
e / e	0.175175	o / oh	0.164283
x / s	0.158519	y / eye	0.155942
e / ai	0.151028	c / s	0.135325
o / ao	0.130111	y / i	0.118607
p / f	0.108407	e / ee	0.108002
a / ay	0.104941	h / ai	0.102463
v / i	0.102243	i / ai	0.101986
u / oo	0.100222	l / ai	0.0992996
i / eye	0.0970222	u / s	0.0933466
b / ai	0.0930889	m / ai	0.0903304
r / ai	0.0809116	i / ee	0.0788786
e / R	0.0773087	k / ai	0.0769748

Figure 6: The top 50 values of the $p(s|l)$ table resulting from the sliding window method.

7 Future Work

Based on the research, it is our belief that the ideal system would borrow elements from both [7] and [5]. The stochastic grammar system of Luk [10] is very computationally expensive but doesn't make up for it in terms of accuracy over decision tree methods.

A good system would use the method of Daelemans [7] to seed the $p(s|l)$

alignment probabilities. However, Daelemans does not iterate, so the resulting probabilities would seed the the epsilon scattering method. The increased bias for the first letters in digraphs should result in better alignments, perhaps approaching the hand-seeded method of Black.

Then the alignments would be placed into an IG tree structure, since other methods didn't perform significantly better.

Although stress assignment is an essential part of speech, it is not considered in this project. Also, a production TTS system must take into account the part of speech, as this can affect pronunciation.

References

[1] CMU Pronouncing Dictionary. `http://www.speech.cs.cmu.edu/cgi-bin/cmudict`.

[2] Project Gutenberg. `http://www.gutenberg.org`.

[3] W. Ainsworth. A system for converting english text into speech. *Audio and Electroacoustics, IEEE Transactions on*, 21(3):288–290, 1973.

[4] J.A. Bilmes. A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. *International Computer Science Institute*, 4, 1998.

[5] A.W. Black, K. Lenzo, and V. Pagel. Issues in Building General Letter to Sound Rules. 1998.

[6] P.F. Brown, S.D. Pietra, V.J.D. Pietra, and R.L. Mercer. The Mathematic of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics*, 19(2):263–311, 1994.

[7] W. Daelemans and A. Van den Bosch. Language-independent data-oriented grapheme-to-phoneme conversion. *Progress in Speech Processing*, pages 77–89, 1996.

[8] H. Elovitz, R. Johnson, A. McHugh, and J. Shore. Letter-to-sound rules for automatic translation of english text to phonetics. *Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on*, 24(6):446–459, 1976.

[9] A.F. Llitjós. Improving Pronunciation Accuracy of Proper Names with Language Origin Classes. *Future*, 96:10–1.

[10] RWP Luk and RI Damper. Stochastic phonographic transduction for English. *Computer Speech & Language*, 10(2):133–153, 1996.

[11] K. Torkkola. An efficient way to learn English grapheme-to-phoneme rules automatically. *Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on*, 2, 1993.