A Comparison of Methods for Speaker-Dependent Pronunciation Tuning for Text-to-Speech Synthesis

Gabriel Webster, Tina Burrows, Katherine Knill

Speech Technology Group, Toshiba Research Europe Ltd
Cambridge Research Lab, 1 Guildhall St, Cambridge CB3 2NH, UK
gabriel.webster@crl.toshiba.co.uk

cambridge.research-lab@toshiba.co.uk

Abstract
Unit-based text-to-speech (TTS) systems typically use a set of speech recordings that have been phonetically transcribed to create a large set of phonetic units. During synthesis, pronunciations for input text are generated and used to guide the selection of a sequence of phonetic units. The style of these system pronunciations must match the style of the phonetic transcriptions of the recorded speech database in order to maximize the quality of the synthesized speech. Furthermore, since different speakers have different speech characteristics, supporting multiple speakers for a single language generally requires applying a speaker-dependent mapping to speaker-independent pronunciations. This paper investigates three automatic methods for this process of speaker-dependent pronunciation tuning: word N-grams, decision trees, and transformation-based learning. Transformation-based learning achieved the best results, in which phones in adjacent words affect the pronunciation of phones in the current word. For example, two important continuous speech effects in American English are flapping a post-vocalic word-final [t] when the next word is vowel-initial, and changing the final vowel of the word the from [ə] to [i] before a vowel-initial word. Word-final flapping is a speaker-independent effect, but changing the vowel of the is speaker-dependent, since some speakers insert a glottal stop between the two words instead of changing the vowel of the. Since it is often difficult to distinguish speaker-dependent effects from speaker-independent continuous speech effects, it makes sense to attempt to model both types of effect at the same time; this is the goal of the present paper.

This process of speaker-dependent pronunciation tuning can be carried out in many ways [3,4,5,6]. The simplest general method is the creation of speaker-dependent pronunciation lexicons that override speaker-independent pronunciations [6], but this method cannot model continuous speech effects. Another class of methods involves developing a graph of multiple possible pronunciations, and passing the entire graph to the unit selection module, which then picks the pronunciations that result in the lowest concatenation and prosody costs [3,5]. In this method, the pronunciation graph need not be explicitly speaker-dependent. Rather, the speaker-dependent tuning is carried out implicitly by the unit selection module: units that were originally contiguous in the speech database will naturally have low concatenation costs, and pronunciations that contain more phone sequences that exist in the speech database (i.e., which are more in the style of the speech database transcriptions) will be selected more often. However, not all TTS system architectures can easily accommodate sending multiple pronunciation candidates to the unit selection module.

Due to these considerations, this paper investigates three automatic pronunciation tuning methods that take each word’s context into account and output a single modified pronunciation: word N-grams, decision trees, and transformation-based learning. The relative performance of each of these methods, both in accuracy and estimated size in a commercial system, are compared on an American English TTS system.

2. General method
All of the following experiments were conducted with Toshiba’s American English TTS system and with a speech database of 2,437 sentences spoken by a female speaker of American English. The speech database data was divided into 90% training and 10% evaluation data. All model refinement and parameter estimation was done by further dividing the training data into training and development sets, so that the test data was used only with the final version of each model.

Two phone sequences were obtained for every sentence in the speech database, the first by phonetic hand labeling, and the second by automatically generating sentence pronunciations using the TTS system. The sentence phone sequences were divided by orthographic word, and the two resulting phone sequences for each word were aligned using a dynamic programming algorithm that found the alignment that minimized the number of insertions, deletions, and substitutions between the two phone sequences. Each system
phone was aligned with zero or more speech database phones, in order to frame the problem as one of mapping each system phone to some (possibly null) speech database phone sequence. Word boundaries were included in the phone sequences by inserting a special symbol “sp” between word pronunciations. It is worth noting that the speech database phone set included released and unreleased versions of each plosive, whereas the system pronunciations generated only one version (which was taken to be the released version).

The accuracy of each model was calculated at the phone and word accuracy levels, with each insertion, deletion, and substitution considered to be an incorrect phone, and the number of actual speech database phones considered to be the total number of phones. The baseline phone error rate, in which the unmodified system phone sequences are scored against the speech database phone sequences, was 8.9%; the baseline word error rate was 37.1%. Experimental results for the models considered are presented in Section 6 and summarized in Table 2.

3. Word N-grams

Inspired by the work of Fackrell et al. [4], an N-gram based method of speaker-specific pronunciation tuning was developed. The general idea is to use the recorded speech database to find a set of word N-grams that correlate with the system pronunciations; when some part of the input text matches an N-gram identified in the speech database.

Fackrell et al. focused on word trigrams, backing off to right bigrams in cases where no trigram matched some input text word. For the current experiments, a somewhat different approach was taken. Firstly, in context words (such as the two words at either end of a trigram), word affixes were considered along with full words, on the grounds that a word’s pronunciation often depends simply on the first letters of the following word or the last letters of the previous word, rather than on the whole word. Furthermore, conditioning on affixes is more likely to generalise to unseen data. To accomplish this, each possible affix of each context word was simply added to the speech database histograms as if it were any other word.

A second difference to Fackrell et al. is that the selection of N-grams began with unigrams, proceeded to bigrams and finished with trigrams, selecting longer-context N-grams only when the pronunciations they predicted were different from those predicted by shorter-context N-grams, or by the system where no shorter-context N-gram existed. To accomplish this, the speech database was used to find the set of word unigrams/pronunciation pairs that improved accuracy over the system pronunciations by a threshold of 2 or more phones. Then, the set of left and right bigrams that further improved system accuracy by 3 or more phones (beyond what any unigrams had already achieved) were found. Finally, the trigrams that further improved accuracy by 4 or more phones were found. Threshold values were determined empirically; the larger threshold values for larger context N-grams presumably help to prevent overfitting. Selecting N-grams in this order ensures that larger context N-grams are not chosen when using a smaller-context N-gram could be used to achieve the same effect, thereby reducing the number and size of the set of selected N-grams, while increasing generalization to unseen data.

During testing, all sentence word pronunciations whose word contexts matched an N-gram selected during training were replaced with the selected pronunciation. In cases where multiple N-grams matched, the pronunciation associated with the longest context N-gram was chosen. If a left bigram and right bigram both matched some word, the left bigram pronunciation was chosen (this was an arbitrary choice).

As an example, the word want is predicted by the system to be pronounced with a final [t]: [w a n t]. During training, the unigram want was chosen with a pronunciation with a final unreleased voiceless alveolar stop [tc]: [w a n tc]; and the right bigram want t* (where t* is an affix “word”, meaning any word that begins with t) was chosen with a pronunciation without any final stop at all: [w a n]. Thus in an unseen sentence containing the word want, the pronunciation [w a n t] generated by the system would be changed by the N-gram approach to [w a n t] if the word were followed by a word beginning with the letter r (a match to the right bigram context), and changed to [w a n t c] in all other contexts (a match to the unigram).

4. Decision trees

Decision trees map from a set of input features to an output class [7]. Training is an iterative process which finds the input feature that partitions the set of training instances into subsets with the best gain in entropy relative to the unpartitioned set. Then, each subset is taken as the set to be partitioned, and a new best input feature is found. When the entropy gain of the best partition falls below a threshold, no partition is made, but rather a best output class is chosen for that set. The result is a tree that is traversed from root to leaf to map a set of input features to an output class. At every branching node, the input value for the feature specified by that node determines which daughter to choose; when a leaf node is reached, the class specified by that node is output by the tree.

For speaker-dependent pronunciation tuning, a decision tree was built using C4.5 [7]. Feature subsetting was turned on, and all other parameters were kept at their default values. Each system phone was mapped to a possibly null “speech database” phone sequence via a separate call to the decision tree. The set of input features, determined empirically on development sets, were a context window of five phones centered around the input system phone, five binary features indicating whether each of the five context window phones is a vowel, and a feature representing the orthography that contains the input system phone. The orthography feature was clustered based on the word N-gram model: all words in the unigram portion of the N-gram model that led to net improvements of three or more words in the training data were left unclustered, and all other words were clustered into the class “<other>”. In this way, the decision tree is made sensitive to the orthographic information that is most important to the accuracy of the N-gram model.

5. Transformation-based learning

Transformation-based learning (TBL) involves automatically inducing a set of rules that maps one kind of symbol sequence to another kind of symbol sequence [8]. In [8], Brill points out that TBL is in some ways similar to decision trees, but
proves that the modeling power of decision trees is a proper subset of that of TBL. In the case of speaker-dependent pronunciation tuning, the task is to map the phone sequence that is output by the system to a phone sequence that matches the pronunciation style of a particular speaker. The first rule is induced by finding the rule that transforms the sequences of system pronunciation phones for the training data text into the sequence most closely matching the speech database phone sequences for the same text. This rule is then applied to the training data, and new rules are iteratively found and applied in this fashion until no rule that improves accuracy by more than a threshold can be found.

The search space of possible rules is defined by a set of rule templates, each specifying a different combination of features to consider together as a possible rule. Because the set of rule templates defines the set of feature interactions that can be modeled, producing the optimal set of rule templates is a crucial step in TBL. In the present task, the features available to be used in templates were the same as those used in the decision tree approach. So for example, one template might consist of the current system phone, the following phone, and the (clustered) orthography containing the current system phone, whereas another might consist of the current and two previous phones. The complete set of templates used, which was determined empirically, is shown in Table 1.

At the end of training, every rule consists of a certain combination of features that must be matched, along with the phone sequence to output when a successful match occurs. For example, in the current task, the first rule induced was

\[
\text{if } \langle \text{cur} \rangle = [d] \text{ and } \langle \text{orth} \rangle = \text{sand}, \text{ then output } [\text{sil}]\]

that is, if the current phone is a [d] and the orthography containing the current phone is and, then delete the [d] (replace it with silence). A later rule gives an example of phone classes:

\[
\text{if } \langle p1 \rangle = [0] \text{ and } \langle \text{cur} \rangle = [s] \text{ and } \langle \text{n1} \rangle = [sp] \text{ and } \langle \text{n2} \rangle = V, \text{ then output } [i]\]

that is, if the previous phone is [0] and the current phone is [s] and the following phone is [sp] (a word boundary) and the following phone but one is a vowel, then replace the [s] with an [i]. This is of course the vowel of the word the changing when the next word begins with a vowel, as automatically induced by TBL.

To apply the induced rules to an unseen system phone sequence, the rules are applied in the same order they were learned. Each rule is matched to the features of each phone in the phone sequence; when the match succeeds, the phone is replaced by the rule output phone sequence. The output phone sequence of each rule is used as the input to the following rule; the final output phone sequence is the output of the last TBL rule.

6. Results

Results for the baseline (the unmodified system pronunciations) and for each of the three methods are presented in Table 2. Error rates are presented at the phone and word level, and for the three models, relative error reduction over the baseline is also given. All three methods give an improvement, but the largest improvement comes from TBL, and the smallest improvement is given by the N-gram approach.

<table>
<thead>
<tr>
<th>Phone error rate (PER)</th>
<th>PER % reduction over baseline</th>
<th>Word error rate (WER)</th>
<th>WER % reduction over baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>11.2%</td>
<td>--</td>
<td>36.9%</td>
</tr>
<tr>
<td>N-gram</td>
<td>9.7%</td>
<td>13%</td>
<td>31.8%</td>
</tr>
<tr>
<td>DT</td>
<td>8.9%</td>
<td>21%</td>
<td>30.2%</td>
</tr>
<tr>
<td>TBL</td>
<td>8.3%</td>
<td>26%</td>
<td>28.3%</td>
</tr>
</tbody>
</table>

One limitation of the N-gram approach is that it can only modify words that appeared in the training data since no N-grams can exist for words not seen in training. To measure the effect of this limitation, word-level results were broken down into test words that were seen in the training data and test words that were not. Altogether, there were 2403 seen and 612 unseen test word tokens. Results are shown in Table 3.

In the baseline results, pronunciations for seen test words can be seen to be significantly more accurate than for unseen test words. This is to be expected, since words present only in the test data are more likely to be rarer words and thus more likely to have automatically generated system.
pronunciations, which are of course less accurate than pronunciations from a lexicon.

Table 3: Word-level results partitioned into results for seen and unseen test words

<table>
<thead>
<tr>
<th></th>
<th>Seen words (2403)</th>
<th>Unseen words (612)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Word error rate (WER)</td>
<td>WER % reduction over baseline</td>
</tr>
<tr>
<td>Baseline</td>
<td>34.2%</td>
<td>--</td>
</tr>
<tr>
<td>N-gram</td>
<td>27.8%</td>
<td>19%</td>
</tr>
<tr>
<td>DT</td>
<td>27.1%</td>
<td>21%</td>
</tr>
<tr>
<td>TBL</td>
<td>25.1%</td>
<td>27%</td>
</tr>
</tbody>
</table>

The N-gram model gives an improvement only for seen test words (as expected). Sizeable decreases in WER are achieved by the decision tree and TBL models for both seen and unseen test words. For TBL the improvement over N-grams on seen words is large (27% versus 19% error reduction). Even for decision trees, the apparently small improvement (21% versus 19% error reduction) actually accounts for the larger part of the decision tree’s overall superiority to N-grams (as seen in Table 2), since the number of seen test words is so much larger than the number of unseen test words. So although the unseen word limitation on N-grams is certainly a contributing factor, a more important factor may be the fact that the N-gram model operates entirely at the word level, and is thus unable to make any generalizations regarding a part of a word’s pronunciation in some context.

The fact that TBL is more accurate than the decision tree might be expected given Brill’s proof that TBL is a more powerful classification technique than decision trees [8]. This increase in power comes from two sources. Firstly, in a decision tree approach, the training data is divided in two parts at each question node, which can result in data sparsity at questions further down the tree. In TBL, the training data is never divided. Secondly, in the present task, each decision tree classification is fully independent: the decision tree makes each classification without any knowledge of the classification outputs of the phones on either side of the current system phone. In TBL, by contrast, each rule is applied to the entire input text before the next rule is applied; therefore, the output of each rule can affect the applicability of the next rule, making more complex transformations possible. In particular, the classification of each system phone can effectively depend on the classifications of both the preceding and following phones, sensitivity that the decision tree lacks. However, the set of TBL rule templates must be optimized for the task at hand for TBL to be able to realize its advantage over decision trees. For example, in a task similar to the present task, that of mapping phone sequences from one dialect to another, Hoste et al. found that a decision tree outperformed a TBL system in which no optimization of the rule template set was performed [9].

In terms of system resources, the size of the decision tree, in a compressed binary format, was 23K, while the size of the TBL rules in simple binary format was 6K (figures for the N-gram model were not calculated). Decision trees and TBL are both computationally linear (O(n)) in the number of phones in the input, because each input phone requires a single traversal of the decision tree or a single application of each TBL rule. Whether the relatively large number of TBL rules generated for the present task (464) can be applied to a single phone as quickly as a decision tree depends on the TBL implementation. As no fast implementation of TBL was produced for these experiments, the issue of relative speed remains a question for future research.

7. Conclusion

Of the three methods presented here for speaker-dependent pronunciation tuning, transformation-based learning achieved the best results, overall as well as on seen and unseen test words. The next research step is to extend these results to different speakers and to other languages. Furthermore, by extending the feature set available to TBL to include other possibly relevant features such as lexical stress, phrasal stress, pausing, and part of speech, further improvements in speaker-dependent pronunciation tuning accuracy may be possible.

8. References