Abstract

To improve the performance and the usability of the speech recognition devices, it is necessary for most applications to allow users to enter new words or personalize words in the system vocabulary. The voicetagging technique is a simple example of using speaker dependent spoken samples to generate baseform transcriptions of the spoken words. More sophisticated techniques can use both spoken samples and text versions of the new words to generate baseform transcriptions. In this paper, we propose a maximum context tree (MCT) based approach to the problem. Comparison is made to the common decision tree based method and Pronunciation by Analogy (PbA) approach. The new approach gives exact baseform transcription for in-vocabulary words and it shows better performance than decision tree. It performs significantly better than PbA approach with less memory usages. MCT uses the word segment probability rather than frequency count used in PbA. MCT uses the full context for the focus letter to overcome the some deficiencies in the PbA approach.

1. Introduction

Significant progress has been made in speech recognition and a growing number of speech recognition systems have been now implemented on mobile devices to provide user-friendly user interfaces. One important aspect of these efforts is to improve the design of the dictionary and its adaptations. The dictionary, that links the orthographic representations and the acoustic realizations, can play an important role in improving the performance of non-phonetic language speech recognition systems. Phoneticians therefore often manually tune the dictionary for some applications.

Automatic phonetic baseform generation in speech recognition applications plays one very important role. It allows user to add new words to their personalized vocabulary[3][4][5]. For example, the names in contact list should be automatically transcribed.

A personalized dictionary on the devices can improve the recognition performance and improve the usability of the device. Use automatic baseform generation, users with different dialects and accents are also able to personalize the dictionary entries or adapt the dictionary entries. That is because such capability that jointly uses both speech samples and a text entry would provide an enhanced mechanism for entering new vocabulary words. It also allows refinement of existing dictionary words that enables the recognizer to accommodate different dialects and accents. In this case, if the dictionary's pronunciation did not map to a particular user's pronunciation, the user could optionally supply a spoken sample and generate customized pronunciations. Once enough pronunciations were entered, the system could potentially have the capability to search the entire dictionary for similar entries and “auto-correct” for the user's particular accent or dialect, even foreign words.

2. Background

Phonetic baseform generation is the problem of determining a canonical pronunciation of a word based on its orthographic representation (or spelling). To solve this problem, most approaches described in the literature and in the prior art are based on the more elemental task of letter-to-sound prediction. One of the most referenced methods is called Pronunciation by Analogy (PbA)[1][2]. In general, linguists have sought to codify letter-to-sound rules for a language using a system of context-dependent rewrite rules of the form

$$\alpha \rightarrow \beta | \gamma_1, \gamma_2.$$

(1)

In Rule (1), the symbol $\alpha$ denotes a letter taken from the input word string. The symbol $\beta$ represents a single phoneme and is $\alpha$'s corresponding phonemic realization. The symbols $\gamma_1$ and $\gamma_2$ are strings over the input alphabet consisting of zero or more letters.

For engineering purposes, the expression given in (1) can be given a probabilistic formulation. A letter to sound prediction can be assigned a probability,
\[ P(\beta | \alpha_k \ldots \alpha_i, \alpha, \alpha_i \ldots \alpha_k) \]  \hspace{1cm} (2)

Where the random variable, \( \alpha \) represents a letter, the random sequences \( \alpha_k \ldots \alpha_i \) and \( \alpha_i \ldots \alpha_k \) represents the letter’s left and right context respectively, and \( \beta \) represents a single phoneme.

Statistical decision trees are the most commonly used method for estimating and computing these probabilities. Decision trees use a considerable amount of \textit{a priori} expert knowledge. For the specific problem of letter-to-sound prediction, the size of the context window has to be determined, as well as the proper set of questions to be asked at each node when growing the tree. The right sized tree can be produced. After growing a tree to some maximal depth, cross-validation samples are used to prune the tree to a size that represents the most effective trade-off between accuracy achieved on the training data versus predicted performance on data not seen during training.

However, in computing (2), decision trees suffer from the basic limitation of predicting one target phone at a time through a sequence of independent decisions. To accommodate the statistical dependence that neighboring phonemes and letters have on one another, one could a) reformulate the problem slightly by including elements of \( \beta \)'s context in the set of conditional variables or b) apply a statistical language model of the form \( p(\beta | \alpha_k, \beta_1, \ldots, \beta_i) \) to be used in conjunction with (2). The disadvantage of alternative (a) is that adding more terms to our probabilistic model leads to an intractable number of parameters that needs to be estimated. The second alternative is more tractable, however, the disadvantage is that the two probabilities need to be estimated from two independent sources. Combining these two sets of probabilities is only valid under certain heuristic assumptions.

### 3. Probabilities from the context trees

#### 3.1. Context tree

In decision tree approach, the words in the aligned learning dictionary are broke into equal length segment, such as a focus letter plus three left and right context letters. The statistics are collected from these word segments. However, in our approach, we organize the aligned learning dictionary into a tree representation. This dictionary is also shared as the system dictionary for speech recognition system. At each node of the tree, the mapping information between the letter and its phoneme transcription is stored as a pair. For any word or word segment, we can use letters as keys to start at any internal node of the dictionary tree to find all possible matches. At the same time, their corresponding phonemes are extracted. For example, we have a word segment “BOX”, we start with all the internal dictionary node with key “B”, then we look for the children nodes with key “O”, if we find “O”, we continue. Otherwise we start with another node starting with “B”.

![Fig. 1. Dictionary Tree. The node is a letter-phone pair.](image1)

![Fig.2. One context tree from word segment ‘BOX’, lower case letters represent phones.](image2)

Due to the different length of match string at different starting tree nodes and the multiple letter-phone mappings, these matched sub-strings are organized as a tree. When the tree nodes are searched, all the matched string segment and their phonetic baseforms are accumulated and counted.

Basically, when the search extends letter by letter from left to right, the context tree grows. If no letter match is found the context tree stops growing. For each input word string, the dictionary is searched repeatedly with all possible pronunciations of a given input sub-string are found. In other words, the search starts at each node of the dictionary tree until all the nodes being used as a starting node. The whole text string is generally not
2. Context tree probabilities

Based on what we called context trees we can now compute probabilities

\[ P(\beta_1, \beta_2, \ldots, \beta_n | \alpha_1, \alpha_2, \ldots, \alpha_m) \]  \hspace{1cm} (3)

The probability \( P(\beta_1, \beta_2, \ldots, \beta_n | \alpha_1, \alpha_2, \ldots, \alpha_m) \) indicates the how likely the phone sequence \( \beta_1, \beta_2, \ldots, \beta_n \) as a whole being generated from a given text string \( \alpha_1, \alpha_2, \ldots, \alpha_m \). In this approach, pronunciations for all possible sub-strings of the input are retrieved from the dictionary and the probability (3) is calculated as the sum of the probabilities for all possible phonetic realizations for the input sub-strings.

For the word \( \omega = \alpha_1 \alpha_2 \ldots \alpha_m \) let \( \omega^k_j = \alpha_1 \alpha_2 \ldots \alpha_k \) denote the sub-string of word to begin in position \( i \) with letter \( \alpha_i \), ending in position \( k \) with letter \( \alpha_k \), having focus letter \( \alpha_j \). In other words, \( \alpha_i \alpha_2 \ldots \alpha_{j-1} \) and \( \alpha_{j+1} \alpha_{j+2} \ldots \alpha_k \) denote \( \alpha_j \)'s left and right context respectively.

Paths \( \tau^{(j)}_{ik} \) in a context tree are a set of letter-to-sound translations of \( \omega^k_j \) found by search the dictionary tree, where \( k \geq j \). In searching the dictionary tree, the occurrence of each path \( \tau^{(j)}_{ik} \) are already accumulated. Let \( N(\alpha_1, \alpha_2, \ldots, \alpha_k) \) be the counts for string segment \( \alpha_1 \alpha_2 \ldots \alpha_k \) and \( M(\beta_1, \beta_2, \ldots, \beta_k) \) be the counts for its \( i \)th transcription. The probability for transcription can estimate as:

\[ P(\beta_1, \beta_2, \ldots, \beta_k | \alpha_1, \alpha_2, \ldots, \alpha_m) = \frac{M(\beta_1, \beta_2, \ldots, \beta_k)}{N(\alpha_1, \alpha_2, \ldots, \alpha_k)} \]  \hspace{1cm} (4)

These probabilities are recorded at the leaf nodes of the context trees. For each node in the context tree, there can be more than one probability associated with it, because the node can be more than one child node. As the first Viterbi pass, we propagate the probabilities on the leaf nodes upwards and retain the maximum probability value for each node.

3.3. Phonetic baseform generation

3.3.1. Build maximum context trees

To produce transcriptions for a new word, we basically transcribe word segments, or partial words. In MCT method, we always search for the longest context available in the learning dictionary. The following procedure describes the process:

For the given new word,

1. Grouping vowel and consonant letters and segment them according to the groups. For example, “BOXT” is marked as “.B.O.XT”.
2. Let the pointer p point to the first “.”
3. Use the entire letter after the p to build a context tree.
4. Set p point to the next mark “.”
5. Goto step 3.

It should be emphasized that the focused letters for which we are looking for corresponding phonemes consist of a consonant string or a vowel string. That is, we obtain corresponding phonemes without breaking the consonant or vowel strings. This avoids a lot of unnecessary and misleading conversions.

3.3.2. Baseform generation

We have built the maximum length context trees for the new word “BOXT”. Because each context tree just represent the partial word transcription. To find the full transcription, we need to align the trees and merge the overlap paths. We use another Viterbi search to accomplish this.

For the purpose of Viterbi search, we translate all the paths \( \tau^{(j)}_{ik} \) in the sub-trees into a lattice representation for generating N-best baseform transcriptions with a Viterbi search. The goal is to connect all possible segment combinations and the path with highest scores is chosen.

For example, the following lattice could be generated for word “BOXT”.

<table>
<thead>
<tr>
<th>Paths</th>
<th>O2</th>
<th>X1</th>
<th>T1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phones</td>
<td>B1</td>
<td>O3</td>
<td></td>
</tr>
<tr>
<td>Phones</td>
<td>B1</td>
<td>O1</td>
<td></td>
</tr>
<tr>
<td>Word</td>
<td>B</td>
<td>O</td>
<td>X</td>
</tr>
</tbody>
</table>

The possible transcriptions are B1O1X1T1, B1O2X1T1, and B1O3X1T1.

To consider the edge effects where the cut point could lose very important context information, a window function, which is centered on the focused grapheme letters, is used to reduce the contribution of the
probabilities near both ends of the text string. Since we estimate the probabilities for each grapheme in the text with all possible lengths of context, the probability of each grapheme is a mixture of all windowed segment probabilities. Penalties are also added to adjust the weights for sub-strings of differing lengths. The shorter the context is, the higher the penalty. This is because a longer context offers more disambiguation than shorter ones.

4. Experiment results

Training data from a CMU dictionary cmu58k94 contains 58k words. The context width seven is used. The total number of seven-letter context segment we have collected is 404405. We randomly selected a number of 322209 segments for training and the remained for testing purpose. It is worth noting that the training set and test set are mutually exclusive. It is very important to keep them mutually exclusive because non-parametric methods like the decision tree method can have perfect performance for the test samples that are contained in the training set. Results in Table 1 are from an independently created test set of 110 k words.

The alignment is done incrementally. We started with a basic set of words being aligned manually or automatically. We sift through the whole training dictionary and select the word segment(s) for which the prediction is most frequently wrong. We correct them and add them to the training dictionary. We iterate the process until we reach a satisfactory result.

For MCT approach, the learning data is the dictionary in the device, which covers the basic needs for the speech recognition (ASR) applications. Actually, this dictionary can be built up recursively so that it covers the data where basic rules can be learned. These basic rules should predict the bulk of the ASR dictionary accurately. Only those words that cannot be predicted by the letter-to-sound rules need to be included in the ASR dictionary embedded in the devices.

Table 3: Performance comparison between MCT and PbA

<table>
<thead>
<tr>
<th></th>
<th>Memory(kbytes)</th>
<th>String Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCT1</td>
<td>75</td>
<td>52%</td>
</tr>
<tr>
<td>MCT2</td>
<td>110</td>
<td>62%</td>
</tr>
<tr>
<td>MCT3</td>
<td>206</td>
<td>78.70%</td>
</tr>
<tr>
<td>PbA1</td>
<td>63</td>
<td>35.50%</td>
</tr>
<tr>
<td>PbA2</td>
<td>111</td>
<td>41.28%</td>
</tr>
<tr>
<td>PbA3</td>
<td>229</td>
<td>47.54%</td>
</tr>
<tr>
<td>PbA4</td>
<td>1,479</td>
<td>65.06%</td>
</tr>
</tbody>
</table>

In Table 1, we see the performance between PbA and MCT methods under memory constraints. We found MCT performs significantly better.

5. Conclusion

We propose a maximum context tree based approach to automatically generate baseform transcriptions. Additionally, it shares resources with the system dictionary, which is important for embedded devices.

6. Reference