COMPRESSION OF EXCEPTION LEXICONS FOR SMALL FOOTPRINT GRAPHEME-TO-PHONEME CONVERSION

Joram Meron *Peter Veprek*

Panasonic Digital Networking Lab., Santa Barbara, CA 93105

(jmeron,pveprek)@research.panasonic.com

ABSTRACT

In this work we present a method to reduce the memory footprint of a grapheme to phoneme conversion (G2P) module, without sacrificing accuracy.

Since the G2P module is typically not 100% correct, it is common to augment the system with an exception lexicon - a list of words which the G2P does not handle correctly (and for which we require correct pronunciations), along with their corrected pronunciation.

Since the size of the exception lexicon is one of the major limiting factors in reducing the overall size of the G2P module, we try to compress the exception lexicon.

We suggest a novel compression method, which is closely tied to the G2P conversion method. The idea behind this compression is that even for words which are not transduced correctly, the decision trees generate a phonetic transcription which is close to the correct one. Therefore, it is sufficient to store only the correction in the exception lexicon.

The correction information is represented in terms of corrections to the transduction *process*, and thus is able to take advantage of the knowledge gained from the training data regarding the probabilities of different corrections, which is used to obtain more efficient compression.

An experiment showed that using this method, an exception pronunciation can be represented, on average, with less than 4 bits (a compression factor of 7, compared to the baseline representation).

1. INTRODUCTION

In this work we are primarily interested in the use of grapheme to phoneme conversion (G2P) for the purpose of pronouncing names, for applications with memory size restrictions (e.g. embedded speech synthesis on a cell phone).

The common approach to G2P conversion is to apply automatic learning methods to a training database, and produce more compact representations, which are able to generalize to unseen input [2] [3].

Similar to [3], our system [1] uses decision trees, but instead of purely automatic, data driven growing of the decision trees, the hybrid method allows the developer the option to directly specify the structure of initial trees (by writing 'rules'). These initial trees are then expanded automatically using a data driven method.

The accuracy of this system compared favorably with other current systems. This system also presented advantages in terms of dictionary maintenance, as well as allowing the use of linguistic knowledge for improving the G2P performance (especially to prevent some of the more 'offensive' phonetization errors).

Some applications have specific domains for which 100% correct performance is required (e.g. a list of the most frequent names). For these cases, the common solution is to collect the names, for which the system does not produce the correct transcription, into an exception lexicon - a separate list holding names and their correct phonetic transcriptions. If a word appears in the list, its transcription is directly pulled from it. Otherwise, the transcription is obtained by running the word through the transduction process.

This work concentrates on a method for compressing the transcriptions in the exception lexicon. The compression method relies on making use of the transduction *process* - storing only corrections to it, instead of the full phonetic transcription.

2. HYBRID G2P - OVERVIEW

I this section we briefly review the hybrid method that we use for G2P conversion. For more details, please see [1].

Our system trains separate decision trees for each possible input letter. To each tree node, a binary question is attached (e.g. "is the the previous letter a vowel ?" or "is the letter two positions back a 'b' ?"), which leads to the next node (depending on the answer), and ultimately lead to a leaf node, which gives the appropriate answers, and the probability of each answer. Each answer can be zero, one or more phonemes.

In this paper we primarily deal with multiple output leaves, which can be used to find the N most probable transcriptions for the given input (which is useful when sharing the G2P module with a recognition system). However, with some

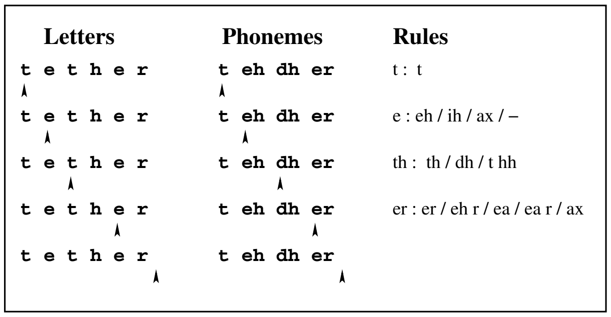

Fig. 1. *Rule machine run on the word ("tether") and its transcription ('t eh dh er'). Arrows indicate the position of the reading heads after each rule is applied.*

modifications, the suggested compression method can be used with (mostly) single output leaves.

2.1. Rules

In [1], rules were introduced in order to improve the performance of the G2P system.

The rules are standard context-sensitive rewrite rules of the general format:

$\{Left_{cntxt}; Letters; Right_{cntxt}; Cond; Results\}$

Where, '$Letters$' can be one or more letters to be transduced together as a block. The left and right contexts include zero or more letters or pre-defined letter sets (e.g. vowels, consonants, etc.). '$Cond$' can describe other feature requirements for this rule (e.g. specifying previously transduced phonemes, number of vowel clusters seen, or any other useful feature that could be computed during run time). '$Results$' specifies the resulting phonemes - zero, one, or several

The rules are interpreted by a rule machine (similar to a two heads Turing machine) which, given a word and its transcription, can either accept or reject the pair. The process of processing a word and its transcription is illustrated in figure 1.

At each step, one rule is selected from the rule set, and if possible (i.e. if one of the possible results matches the transcription) applied - the heads are advanced according to the number of letters and phonemes consumed by the rule. If no applicable rule is found, the pair is rejected (rejected words, which typically account for around 1-2% of the words, can be treated by a 'conventional' exception lexicon, and are excluded from the discussion in this paper).

2.2. Decision Trees

We use decision trees in a way similar to [3][5]. The only modification to the decision trees is that instead of holding one output phoneme in each tree leaf, the leaf now holds: 1) an output phoneme *sequence*, and 2) the length of the grapheme block to be consumed (i.e. after output is produced for the current letter, the G2P will consume the indicated number of letters, and advance to the first letter after the current block).

For storage efficiency, we define an **output codebook** for each of the possible input graphemes. This codebook contains all the different possible combinations of
$\{$output phoneme sequence ; letter block length$\}$
(the size of this codebook can reach a few hundreds for some letters). In each leaf, we store an **output code** (which refers to the output codebook).

The data for training the trees is produced by the rule machine. For each step of the rule machine (each line, except the last, in figure 1), one output vector is added to the training data. This vector includes the letter context around the location of the letter reading head, the previous phonemes produced prior to the location of the phoneme reading head, auxiliary features calculated for that point, and the corresponding output code mentioned before.

The manually created rules are used to automatically create initial decision trees (one tree for each possible input grapheme). For each of the manual rules, one branch is added, which consists of all the components of the rule (one component per each branch node). It's easy to see that the rules force an initial segmentation of the training data - they let the developer decide what cases are "similar" or "belong together" - as opposed to the results of automatic tree building, where disparate cases might be grouped together based on irrelevant similarities. This also allows manual control over the structure of the tree, thus alleviating the 'black box' nature of the data driven method.

The final step is an automatic extension of the initial tree, using a data-driven process, which is similar in principle to the methods in the references (but trained on data produced by the rule machine, as explained above).

First, each training vector is associated with its corresponding terminal nodes in the initial tree (corresponding to a rule). Then, a standard tree building process is used to extend the tree from each terminal node (using its associated training vectors).

3. COMPRESSING EXCEPTION LEXICONS

As mentioned above, the size of a G2P is an important factor for some applications in which we're interested. The G2P system is made of the decision trees and the exception lexicon. There is a trade-off between the sizes of these components - as we try to shrink the trees (e.g. by a pruning process), more incorrect pronunciations are generated, which requires increasing the size of the exception lexicon.

Depending on the degree of pruning of the trees, and the composition of the exception lexicon, the limiting factor for making the G2P system smaller is often the size of the exception lexicon.

We suggest a method for compressing the transcriptions in the exception lexicon, in which, instead of storing the whole transcription, only corrections to the transduction *process* are stored. In this way, we are able to apply the knowledge represented in the trees, even for words in the exception lexicon, for all but a small part of the transduction process.

3.1. Transcription Corrections

As explained above, the training set, used for the automatic growing of the trees, is generated by running pairs of $\{spelling, transcription\}$ through the rule machine, and collecting the output vectors, which include letter context, available features, and the correct output at each step

After training the trees, we can run the G2P transducer (without an exception lexicon) over the *training* data, and compare the generated output at each step, to the correct output (as it appears in the corresponding training vector).

For most words, the transduced outputs are the same as the correct ones. For those words which differ, we note the correct output for the points where they differ, and the location (step number) of this point. Thus a correction is represented as a pair: $\{step\#; output\#\}$.

Having stored this information, we can obtain the correct transcription for the word in the following way:

1. Start by setting the current position pointer to the its first letter, and set step number to 0.
2. Use the tree of the letter in the current position
3. Traverse tree - obtaining the *active leaf* (the matching terminal node).
4. If no correction for this step - output phoneme(s) indicated by leaf's output code, and advance current position as indicated by this code.
5. If a correction is given, output letter(s) indicated by correction output code, and advance current position as indicated by it.
6. Advance step by 1
7. If current position hasn't reached word end - repeat from step 2.

Storing these correction pairs can already achieve some compression, since we need to store corrections only for a small number of the transduction *steps* (as even for wrongly transduced words, most transduction steps are correct).

To test this claim, trees were trained on a train set containing 94130 words (British names). The same words were transduced using these trees, and the number of corrections per word were counted. The results are shown in table 1. For words with at least one error (the 16031 words, which would consist the exception lexicon), only 17875 out of 91086 transductions (19.7%) need correction.

Number of errors	Occurrences
0	78099
1	14372
2	1520
3	138
4	11

Table 1. *Number of transduction errors per word.*

3.2. Transcription Alternatives

The output code contained in the corrections refers to a grapheme's output code which can be quite large (hundreds of output combinations for some graphemes) - which puts a limit on the reduction of the number information bits needed to represent a correction.

As mentioned above, the leaves of the decision trees contain lists of several possible outputs, along with their probabilities, which are used to find the best transcription (or transcriptions) for the input word.

This can be used to compress the correction information, as for a given transduction step, the correct output to be produced is one of the entries in the active leaf's list of possible outputs. In most cases, the first output on the list is the correct output (since the list is sorted by descending order of frequency), so no correction is needed. If a correction is needed, it will be one of the other possible outputs.

The way the trees were built guarantees that the correct output will be in the list of possible outputs of the active leaf. This is because the same step, for the same word (when used in the training), was classified to the same leaf, and therefore its output is in that leaf's possible output list.

The correction, therefore, can be specified as the one out of the possible outputs for the active leaf, rather than one of all the possible entries in the current grapheme's output codebook. Thus, a correction takes the form $\{step = i; alt = j\}$ - meaning: "take the j-th alternative in the i-th step". The leaf's list of possible outputs is much shorter (typically 1-4 entries) than the grapheme's output codebook, which makes for a significant saving in storage space.

Switching from specifying the letter's output code number to specifying the leaf's alternative number provides further savings when we consider the compressibility of the distributions - e.g. when applying a Huffman coding (which reduces average bit number by allocating less bits to more common codes). Table 2 shows the distribution of the different alternative numbers.

Alternative	Occurrences
1	15425
2	1990
3	350
4	72
5	21
6	10
7	5

Table 2. *Frequency of use of alternatives.*

The frequency distribution of the *grapheme's* output codebook is collected over the whole tree - all the appearances of the letter, and is therefore too general. For the leaf's alternatives, on the other hand, the distribution is calculated locally, and is more specific. Another way of looking at it is that it takes full advantage of the knowledge which is already embedded in the tree. Given a specific context, the trees tell us to go to a specific leaf, and output a specific alternative. Even if the suggested alternative is not the correct one, the leaf is correct, which saves us a few bits of information.

Last, we observe the different combinations of corrections - all the different combinations of $\{step; alternative\}$ which appear together for any one word. For the database used in our experiment, there were 259 different correction combinations. A histogram of their frequency is shown in figure 2, and can be significantly compressed using Huffman coding.

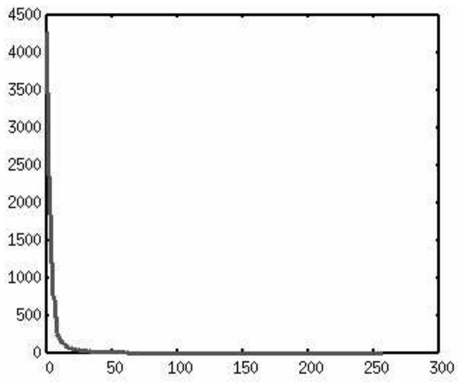

Fig. 2. *Number of occurrences of all the different correction combinations.*

3.3. Compression Experiment

We've used the same database, to compare the required size for representing the transcriptions in the exception lexicon (16031 words), using the different methods.

For the baseline method, the transcription is represented as a string of phonemes, each phoneme represented by a minimum length code (Huffman code). For the lexicon used, the average transcription length is 5.75 phonemes per word. The average length of a phoneme's code (calculated using entropy) is 5.01 bits. Therefore, on average, each transcription requires **28.8 bits**.

With the proposed compression method, the average storage required per transcription is **3.9 bits** (obtained from the entropy of the distribution in figure 2).

4. CONCLUSION

A method of compressing exception lexicons was presented, which takes advantage of the knowledge represented inside the trees, even when the trees' default output is wrong. Using this compression, a significant reduction in the size needed to represent a transcription was shown (a factor of 7).

Further work will be done on trying to reduce the size of the trees by pruning the lists of possible outputs for the leaves. One way to achieve this is to hold a full list of alternatives in some ancestral node, and having the corrections refer to these lists instead. In addition to compressing the transcriptions, compression should also applied to the list of the words (spellings) in the exception lexicon.

5. REFERENCES

[1] Meron J., "Using Rules to Improve Letter to Sound Conversion of Names", IEEE Workshop on Speech Synthesis, 2002, Santa Monica

[2] Damper R.I. et al., "A Pronunciation-by-Analogy Module for the Festival Text-to-Speech Synthesizer", Proc. 4th ISCA workshop on speech synthesis, 2001, pp. 97-101

[3] Pagel V., Lenzo K., Black A.W., "Letter to Sound Rules for Accented Lexicon Compression", Proc. ICSLP 1998, pp. 2015-2018

[4] Andersen O., Kuhn R., et al., "Comparison of Two Tree-Structured Approaches for Grapheme-to-Phoneme Conversion", Proc. ICSPL 1996, pp. 1700-1703

[5] Kuhn R., Junqua J.C., Martzen P.D., "Rescoring Multiple Pronunciations Generated from Spelled Words", Proc. ICSLP 1998, pp. 2707-2710

[6] Pearson S., et al., "Automatic Methods for Lexical Stress Assignment and Syllabification", Proc. ICSLP 2000, pp. II 423-426