Efficient Compression Method for Pronunciation Dictionaries

Jilei Tian
Audio-Visual Systems Laboratory
Nokia Research Center, Tampere, Finland
jilei.tian@nokia.com

Abstract

Pronunciation dictionaries are often used with other data-driven methods to model the pronunciations in phoneme-based automatic speech recognition (ASR) and text-to-speech (TTS) systems. The dictionaries usually take a great amount of memory, which is a limiting factor in portable handheld devices. Compressing the pronunciation dictionaries results in minimal transmission bandwidth and less storage memory. In this paper we present a new procedure to efficiently compress pronunciation dictionaries. First, a novel method transforms the dictionary to a lower entropy representation. Second, the variability in the aligned pronunciation dictionary is reduced to further lower its entropy. Finally, generic lossless compression is applied on the transformed dictionary. Experiments were carried out on English names and words from US English CMU dictionary. The proposed scheme achieved 37.5% improvement over general-purpose lossless text compression.

1. Introduction

Pronunciation dictionaries are commonly used for language learning, automatic speech recognition and speech synthesis. Efficiently compressing the dictionary results in minimal bandwidth usage and less memory required. This paper focuses on the application of the compressed pronunciation dictionary in phoneme-based ASR and TTS systems. Our multilingual system, capable of handling dynamic vocabularies, consists of several units [1]. At first, the language identification module identifies the language of a given word. Next, pronunciation modeling is applied in order to obtain the phonetic transcription of the word. Finally, the recognition model for the word is constructed by concatenating multilingual acoustic monophone models according to the phonetic transcription. The recognizer can automatically cope with multilingual entries without any assistance from the user. The phonemic transcription is also used for TTS. Pronunciation modeling is of critical importance in the system. The pronunciation models are language-dependent. Different methods that can be applied are pronunciation rules, pronunciation dictionaries and data-driven approaches [2]. The advantage of pronunciation dictionary is that it provides error-free transcriptions. Its disadvantages are the potentially high memory use and that it is not able to predict the pronunciations for unseen words.

The memory and computational resources for ASR and TTS applications implemented on embedded portable devices are inherently sparse. Though these resources are expected to increase in future portable devices, the number of applications required to run simultaneously is also very likely to increase. The memory use of pronunciation dictionaries is usually high, so an efficient compression method is needed. Our aim is to minimize the size of the pronunciation dictionaries while keeping the ASR and TTS performance unaffected.

Generally, all kinds of well-known lossless text compression methods could be used for this purpose. In our particular task, however, the following requirements should also be taken into account.

1. High efficiency; though the existing compression methods are good in general, their compression performance is not sufficient for pronunciation dictionaries.
2. The algorithm should use limited run-time memory for decoding;

In this paper, we propose a new procedure to efficiently compress pronunciation dictionaries by taking their special characteristics into account. First, we present a novel method to improve the grapheme-to-phoneme alignment in the dictionary. The aligned dictionary is then transformed to a lower entropy representation. Finally, a suitable lossless compression method is applied on the lower entropy dictionary, which uses a small amount of run-time memory for decoding. Though we have applied the presented dictionary compression to ASR and TTS tasks, it can be used for other purposes, such as dictionary transmission or storage.

The paper is organized as follows. In the next section, the dictionary transform is outlined. In Section 3, the improved alignment approach is described. The general compression algorithms are briefly introduced in Section 4 followed by the experiments in Section 5. Finally, conclusions are drawn in Section 6.

2. Dictionary transform

Shannon established that there is a fundamental limit to lossless data compression, called the entropy denoted by H. The exact value of H depends on the information source, more specifically, on the statistical nature of the source. The best possible lossless compression rate is the entropy [3].

The entropy rate H in the general case is given by

$$H = \lim_{n\to\infty} -\frac{1}{n}\sum p(B_n)\cdot \log_2 p(B_n), \qquad (1)$$

where B_n represents the first n characters. It is virtually impossible to calculate the entropy directly according to the equation (1). The entropy H gives a theoretical lower bound for compression of a text source.

Obviously, an efficient compression scheme should consider:

1. Processing the text to have lower entropy;
2. Using optimal compression scheme to asymptotically reach the entropy.

The paper focuses on the processing of pronunciation dictionaries. For a given pronunciation dictionary, an original entry is represented in the following format (using TIMIT phoneme notations):

"ada ey d ax"

All phonemes can be represented by single byte in the aligned dictionary, so all spaces in the phonetic sequence can be removed. For example, by mapping ey->***p*$_1$**, d->***p*$_2$**, ax->***p*$_3$**, we have

"ada ***p*$_1$*p*$_2$*p*$_3$**"

So the entry size is reduced from 11 to 7 bytes. For simplicity, the second-order entropy H_2 is calculated as mentioned below.

Let $P(l_j|\ l_i)$ be the conditional probability that the present character is the jth letter in the alphabet given that the previous character is the ith letter. The entropy H_2 of the second order statistics is

$$H_2 = \sum_{i=1}^{m} P(l_i)\cdot H_{2i}$$
$$H_{2i} = -\sum_{j=1}^{m} P(l_j \mid l_i)\cdot \log_2 P(l_j \mid l_i) \qquad (2)$$

where H_{2i} is the entropy of a single letter l_i.

The second-order entropy H_2 is calculated from the original dictionary. We now have

H*$_2$*=3.81 bits/char

Since the letter->phoneme pattern has higher correlation than consecutive letters, the redundancy is increased by interleaving the pronunciation phonemes and letter sequence of a given word. For instance, the above-mentioned entry is transformed as:

"a ***p*$_1$** d ***p*$_2$** a ***p*$_3$**",

where italic and bold symbols stand for phonemes and the rest of the symbols represent letters. It is clear that the correspondence between the original and new formats is one-to-one. By the representation transform of the dictionary, the second-order entropy is decreased to

H*$_2$*=3.17 bits/char

It observed that the dictionary transform plays an important role to lower the entropy and to reach a high compression performance. The detailed analysis is given below.

Assume that the alphabet and phoneme sizes are M and the maximum number of phonemes a single letter may map to is N. For a very large amount of data, we could ease analysis if assuming that it has statistically equal conditional probability that the present character is the jth letter given that the previous character is the ith letter. We have

$$P(l_j|\ l_i)=1/M \text{ and } M > N. \qquad (3)$$

Suppose the entry is aligned. Section 3 introduces more details about alignment. Typical aligned pronunciation entry is written as:

$$l_1, l_2, \ldots, l_m, p_1, p_2, \ldots, p_m$$

$$H_{2i} = -\sum_{j=1}^{M} P(l_j \mid l_i)\cdot \log_2 P(l_j \mid l_i) \approx \log_2(M) \qquad (4)$$

By entry transform, we have

$$\overset{)}{H}_{2i} = -\sum_{j=1}^{N} P(p_j \mid l_i)\cdot \log_2 P(p_j \mid l_i)$$
$$Max(\overset{)}{H}_{2i}) = \log_2(N) \qquad (5)$$

So we have:

$$\overset{)}{H}_{2i} \le Max(\overset{)}{H}_{2i}) = \log_2(N) < \log_2(M) \approx H_{2i} \qquad (6)$$

Taking equation (2) into account, $\overset{)}{H}_2 < H_2$. Thus the entropy is decreased after entry transform.

3. Improved dictionary alignment

The entries in the dictionary must be aligned in order to transform the dictionary into lower entropy. The entries in the dictionary are composed of the letter sequence of words and their phonetic transcriptions. Take English language as example, a letter may correspond to zero, one, or two phonemes. The alignment is obtained by inserting graphemic or phonemic epsilons between the letters in the letter sequence, or between the phonemes in the phoneme sequence. The use of grapheme epsilons can be avoided by introducing a short list of pseudophonemes that are obtained by concatenating more phonemes that are known to correspond to a single letter, for example, "x->k s". In order to align the entries, the set of allowed phonemes has to be defined for each letter. The phoneme list includes the pseudophonemes for the letter and the possible phoneme epsilon. For a given letter-phoneme pair, the penalties $p(p|l)$ are initialized with zero if the phoneme p can be found in the list of allowed phonemes for the letter l, otherwise it is initialized with a large positive value.

Given the initial penalty values, the dictionary is aligned in two steps [2]. In the first step, all the possible alignments are generated for each entry in the dictionary. Based on all the aligned entries, the penalty values are then re-scored. In the second step, only a single best alignment is found for each entry.

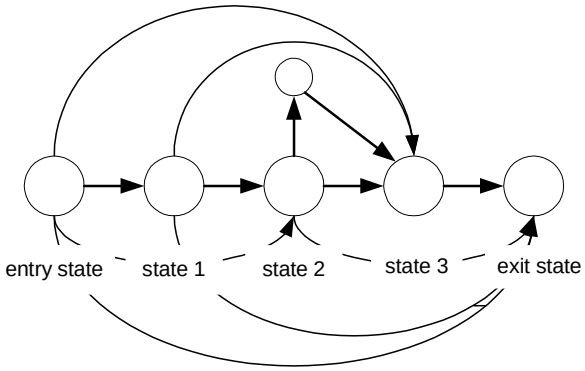

Figure 1. The grapheme HMM for aligning the phoneme sequence to the letters of the word. State 1-3 corresponds to the letters of the word.

For each entry, the optimal alignment is found with the Viterbi algorithm on the grapheme HMM shown in *Figure 1*. The grapheme HMM has an entry, an exit and letter states. The letters that may map to pseudophonemes are handled by having a duration state. The states 1 to 3 in *Figure 1* are the states that correspond to the spelled letters in the word. State 2 corresponds to a letter that may produce a pseudophoneme. The state skips from all the previous states to the current state

are allowed in order to support phoneme epsilons. Each state and the duration state hold a token that contains the cumulative penalty of aligning the phoneme sequence against the grapheme HMM and the state sequences that correspond to the cumulative score. The phoneme sequence is aligned against letters by going through the phoneme sequence from the beginning to the end one phoneme at a time. Token passing is carried out in order to find the Viterbi alignment between the letters and the phonemes. The token that has the lowest cumulative penalty is found over all the states of the HMM. Based on the state sequence of the token, the alignment between the letters of the word and the phonemes can be determined.

Obviously, large pronunciation dictionaries typically contain "outlier" entries characterized as following list:

- Non-native names and words, like "Juan, Xiong, etc" may be included e.g. in an English pronunciation dictionary.
- It is inevitable to have some wrong transcriptions in the dictionary due to typos and some other unpredictable reasons. They increase the inaccuracy of pronunciations.
- Invalid alignments, for example "apple -> ae p ah l _". By basic linguistic knowledge, we know that letter "p" never maps to vowel phoneme "ah". This makes the grapheme-to-phoneme pairs more irregular.

All the cases listed above make the pronunciation more irregular, and irregular pronunciations decrease the compression efficiency and make the grapheme-to-phoneme mapping more inaccurate. If the above-mentioned problems are identified, the aligned pronunciation will be more regular leading to high compression efficiency and mapping accuracy. A new improved alignment algorithm is proposed in this paper to alleviate these problems.

As described above, the final Viterbi alignment is done based on *P(p,l)* that is estimated from the aligned dictionary during the first step of the alignment algorithm. Obviously, the first step produces a very rough alignment, so *P(p,l)* is by no means very accurate. Another drawback is that a non-zero value is assigned to *P(p,l)* even in the case of linguistically impossible mappings. For example, *P*("ah","p") has a non-zero value, but it is clearly against linguistic knowledge. In order to avoid this and to overcome the difficulties listed above, the following constraint is imposed on the Viterbi decoding.

In the proposed alignment algorithm the alphabetic and phonetic sets need to be defined for the target language. Once the sets are defined, the list that specifies all the possible phonemes and pseudophonemes for each letter is created. This list includes truly language-dependent information.

In the second step of the presented method, *P(p,l)* is estimated as usual if phoneme *p* can be found in the list of phonemes that are linguistically allowed for the letter *l*. If phoneme *p* can not be found in the list for a given letter *l*, then we apply the constraint to set *P(p,l)* to be the highest value without any re-estimation. In this way, we obtain the following benefits:

- The aligned dictionary is in line with linguistic knowledge; incorrect entries can be identified. Therefore, the irregularity is reduced and alignment is improved in terms of accuracy.
- The entries for foreign words/names can also be detected if they obey different pronunciation rules than targeted language. Obviously the irregularity is reduced too.
- To some extent, wrong transcriptions are identified and corrected.
- Since the entries containing undefined mapping pairs are clipped out, the defined mapping pairs having small probabilities can be correctly modeled (they are badly affected by the undefined mapping pairs in the original method.)
- The linguistic information can be build up. By checking the entries that are clipped out, the linguistic information can be tuned, e.g. by defining new pseudophonemes, adding missing phonemes into letter-dependent phoneme set, etc. By having better linguistic information, the alignment can be improved correspondingly.

The alignment procedure works for most of the entries. Some entries, however, may not be aligned. In such cases, forced alignment is applied. The grapheme or phoneme epsilons are added to the end of letter or phoneme sequences when a given entry is not aligned.

4. Compression

Broadly speaking, most of the lossless data compression methods in common use today fall into one of two categories [4]: dictionary-based or statistical methods.

The dictionary-based compression methods are based on the observation that different types of data contain repeating code sequences. Good examples of such data are text files (code words represent characters) and raster images (code words represent pixels). These methods can be subdivided into two main groups. The methods of the first group try to find if the character sequence currently being compressed has already occurred in the input data. Then instead of repeating it, only a pointer to the earlier occurrence is used. The implicit dictionary here is represented by the previously processed data. All the methods of this group are based on the Lempel-Ziv-77 (LZ77) algorithm. The algorithms of the second group create a dictionary of the phrases that occur in the input data. When they encounter a phrase already present in the dictionary, they just output the index number of the phrase in the dictionary. The refinement, which is the basis for the later methods, is called Lempel-Ziv-Welch (LZW).

Arithmetic coding is a method of encoding data using a variable number of bits. The number of bits used to encode each symbol varies according to the probability assigned to that symbol. Low probability symbols use many bits, high probability symbols use fewer bits. So far, this makes arithmetic coding sound very similar to Huffman coding. However, there is an important difference. An arithmetic coding does not have to use an integral number of bits to encode a symbol. This means an arithmetic coding can usually encode a message using fewer bits than Huffman coding.

The performance of arithmetic coding can further be improved by combining it with powerful statistical modeling techniques. The performance of these improved arithmetic compression schemes can, in some cases, come close to the Information theoretical lower bound. The disadvantage of these schemes is that they require an extensive amount of run-

time memory, especially when higher statistical order is used. Thus they are not applicable in handheld devices with limited memory resources. In the world of embedded systems, dictionary based data compression techniques are often chosen because they can operate using small decoding buffers.

5. Experiments

Firstly, the preliminary experiments, as shown in *Table 1*, compare the compression performances by applying LZW algorithm [5] on the dictionary pre-processed by different techniques. We have used a lookup table containing 2000 English names and their pronunciations. The lookup table is compressed from the original 37.0 KB to 17.6 KB by applying the LZW algorithm directly, as baseline result. The size is further reduced to 12.0 KB on the transformed dictionary without using improved alignment. Finally proposed alignment can improve the compression to 10.9 KB as shown in *Table 1*. The compression rate is improved by 38% compared to the baseline performance.

Methods	Original	LZW	Interleaving	Alignment
Memory	37.0	17.6	12.0	10.9

Table 1. Comparison of compression performance between different pre-processing schemes.

The experiments are also carried out on the CMU English pronunciation dictionary [6] that contains 109,409 entries. The whole size of the dictionary is 2,580 KB. Both dictionary and statistical compression methods are tested. The compression algorithms in our experiment are LZ77, LZW and arithmetic coding with second order statistics. The baseline performance is evaluated on the original pronunciation dictionary directly. The compression methods are then applied to the transformed pronunciation dictionary denoted as low entropy dictionary. *Table 2* shows that the proposed dictionary transform method performs well with all tested compression algorithms. In the table all figures are given in kilobytes (KB).

Method	Baseline	Low-entropy	Improvement
LZ77	1181	940	20.4%
LZW	1315	822	37.5%
2^{nd} order Arithmetic	899	501	44.3%

Table 2. Comparison of compression performance among different lossless compression algorithms.

As shown in *Table 2*, the proposed dictionary transform method can improve the compression rates with all well-known compression algorithms. In the following, we compare the properties of the presented compression algorithms according to the following factors:

1. Total memory usage including temporally used memory for decompression;
2. Compression/Decompression speed;
3. Compression rate.

The LZW algorithm [5] decompresses in an on-line manner. It iteratively decompresses small piece of compressed data and stores into buffer, then reload and process next decompressed data until a given entry is found or the compressed data is completely decoded. This decoding buffer can be small.

Table 3 shows the overall performance of three compression algorithms based on the factors listed above. LZW algorithm has the best decompression speed and a good decompression rate with the smallest memory requirement. All in all, the LZW algorithm fits the best our requirements. It improves the compression rate by 37.5%.

Methods	Compress rate	Memory	Speed
LZ77	Low	Smallest	Medium
LZW	Medium	Small	Fast
Arithmetic	High	Large	Slow

Table 3. Overall performance comparison among three typical compression algorithms.

6. Conclusions

In this paper we focused on the implementation aspects of low-memory pronunciation modeling for embedded devices. More specifically, we developed a novel procedure for compressing pronunciation dictionaries.

The proposed method first transforms the dictionary to a lower entropy representation. Secondly, it reduces the variability in the aligned pronunciation dictionary to further reduce its entropy, and finally uses a generic lossless data compression method to obtain the compressed pronunciation lexicon. All steps involved are lossless, so the ASR or TTS performance is not affected by the compression.

We have carried out experiments with US English names and words from the US English CMU dictionary. The proposed scheme achieved a 37.5% improvement over general-purpose lossless text compression. The presented method can also be applied to other languages as well as be combined with other lossless compression algorithms.

7. References

[1] Viikki, O., Kiss, I. and Tian, J., "Speaker- and Language-Independent Speech Recognition in Mobile Communication Systems", in *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, Salt Lake City, USA, 2001.

[2] Suontausta, J. and Häkkinen, J., "Decision Tree Based Text-To-Phoneme Mapping for Speech Recognition", in *Proceedings of International Conference on Spoken Language Processing*, Beijing, China, 2000.

[3] Hankerson, D., Harris, G. and Johnson, P., *Introduction to Information Theory and Data Compression*, CRC Press, USA, 1998.

[4] Bell, T., Cleary, J. and Witten, I., *Text Compression*, Prentice Hall Inc., New Jersey, USA, 1990.

[5] M. Nelson, "LZW Data Compression", *Dr. Dobb's Journal*, October, 1989.

[6] http://www.speech.cs.cmu.edu/cgi-bin/cmudict: "The CMU Pronouncing Dictionary", Carnegie Mellon University, US.