A TRAINING PROMPTS GENERATION ALGORITHM
FOR CONNECTED SPOKEN WORD RECOGNITION

Ha-Jin Yu and Jin Suk Kim
Department of Computer Science and Statistics
University of Seoul, Korea
hjyu@uos.ac.kr

ABSTRACT
This paper describes an efficient algorithm to generate compact prompts lists for training utterances. In building a connected speech recognizer such as a connected spoken digit recognizer, we have to acquire speech data with various combinations of the words contexts. However, in many speech databases the lists are made by random generators. We provide an efficient algorithm that can generate compact and complete list of words with various contexts. The algorithm begins with a series of unique digits, which is used for difference series of another digits series of wider context. The process is applied recursively until desired context range is achieved. The algorithm can be generalized to any range of contexts, such as tri-words and four-words. This paper includes proof of optimality and completeness of the algorithm.

1. INTRODUCTION
When we build a connected speech recognizer, the first job is to record speech data from many speakers. In the process, it is important to make a prompt list that can represent the words in the vocabulary in various contexts. It is also desirable that the list is compact enough so that one speaker can cover the entire list. However, most of the researches on connected spoken digit recognition do not pay much attention to the training prompts [1,2]. Many researchers rely on simple random number generators [3] or they consider only very limited context, that is, bi-words [4]. In OGI 30k-Numbers corpus, they recorded speaker’s ZIP code, street address, or other numeric information [5].

If we want to record spoken words with different left and right word contexts, that is to record all the tri-words, the amount of prompt list can be too large for one person to speak. Suppose we want to make a connected spoken digit recognizer. If we have ten digits, we have 10^3 different tri-words. If we do not consider beginning and ending contexts, a string of n words can include n-2 tri-words in general. For example, if each string has seven words, we can acquire full tri-word list with only 200 strings, which is not too large for one person to speak.

In this paper, we describe an efficient algorithm to make compact tri-word prompts lists. The next section introduces the algorithm and the following sections show the examples and analysis of the algorithm.

2. THE ALGORITHM AND ANALYSIS
2.1. Prompts generation algorithm
Suppose we make a list of strings S_i of n words indices s_1,…,s_n with T words in the vocabulary. We assume that T/(n-2) is an integer.

**Definition 1** (first-order difference string): We define a difference string D_k=(d_1^k, d_2^k, ..., d_{n-2}^k), which are the differences between the words indices in the final string S_k=(s'_1, s'_2, ..., s'_{n-2}),

\[ s_{i+1} = (s_i + d_i) \mod T \]  

**Definition 2** (second-order difference string): We define second-order difference string A_k=(a_1^k, a_2^k, ..., a_{n-3}^k) as the difference string of a difference string D_k,

\[ d_{i+1} = (d_i + a_i) \mod T \]  

**Definition 3** (tri-word): A tri-word is a 3-tuple \[(s_1, s_2, s_3)\] for zero and “yuk” and “ryuk” for six and four in Japanese (“shi” and “yon”), the number of tri-words increases drastically. The tri-word recording is needed especially for Korean digits recognition, because each digit consists of only one syllable or even one phone. However, if we have an efficient representation of word string, we can record n different tri-words with a string of n words. If we do not consider beginning and ending contexts, a string of n words can include n-2 tri-words in general. For example, if each string has seven words, we can acquire full tri-word list with only 200 strings, which is not too large for one person to speak.

This paper describes an efficient algorithm to generate compact and complete list of words in various contexts. The algorithm begins with a series of unique digits, which is used for difference series of another digits series of wider context. The process is applied recursively until desired context range is achieved. The algorithm can be generalized to any range of contexts, such as tri-words and four-words. This paper includes proof of optimality and completeness of the algorithm.
Figure 1 illustrates the definitions and Figure 2 shows an example of the string definitions.

Second order difference \( a_1 \ a_2 \ a_3 \ a_4 \ a_5 \)
First order difference \( d_1 \ d_2 \ d_3 \ d_4 \ d_5 \)
Final string \( s_1 \ s_2 \ s_3 \ s_4 \ s_5 \ s_6 \ s_7 \)

**Figure 1:** The definitions.

Second order difference \( 1 \ 2 \ 3 \ 4 \ 5 \)
First order difference \( 1 \ 2 \ 4 \ 7 \ 1 \ 6 \)
Final string \( 1 \ 2 \ 4 \ 8 \ 5 \ 6 \ 2 \)

**Figure 2:** An example of the string definitions.

**Algorithm 1** (Generate tri-word strings)

**Input:**
- \( T \) is the total number of words in the vocabulary.

**Output:**
- \( T(n-2)^t \) strings \( s_1^t \), \( s_2^t \), …, \( s_n^t \) of the tri-words

1. Choose \( n \) such that \( T(n-2)^t \) is an integer.
2. Generate \( T(n-2)^t \) strings of second-order differences \( A_k = (a_{i1}^t, a_{i2}^t, \ldots, a_{in}^t) \). The order of the digits is not important, but each digit should appear exactly once.
3. By Definition 2, for each string of second-order differences \( A_k \), generate \( T \) difference strings \( D_k = (d_{i1}^t, d_{i2}^t, \ldots, d_{in}^t) \), with \( d_{ik}^t = k, k=0,1,\ldots,T-1 \).
4. By Definition 1, for each string of differences \( D_k \), generate \( T \) strings \( S_k = (s_{i1}^t, s_{i2}^t, \ldots, s_{in}^t) \), with \( s_{ik}^t = k, k=0,1,\ldots,T-1 \).

**2.2. Optimality and Completeness of the Algorithm**

We say an algorithm is **optimal** if it produces minimum number of strings that cover all possible tri-words given the length of the strings \( n \). When we use the strings of length three, we have to generate \( T^3 \) strings to have all possible tri-words. When we use the strings of length four, we can reduce the number of strings to \( T^2 / 2 \), because each string contains two tri-words. In general, when we use the strings of length \( n \), we need at least \( T^n / (n-2)^2 \) strings. The algorithm generates exactly \( T^n / (n-2)^2 \) strings. Step 2 generates \( T(n-2)^t \) strings and step 3 and step 4 multiply the number of strings by \( T \).

Now we show that there is not a duplicated tri-words in the list the algorithm generates. In step 2, the digits \( a_{i1}^t \) in the strings are made unique. In step 3, Equation 2 is applied \( T \) times for each \( a_{i1}^t \) with \( a_{i1}^t = 0,1,2,\ldots,T-1 \). Therefore, there is not a duplicated pair \((d_{i1}^t, d_{i2}^t)\) because different \( d_{i1}^t \)'s are applied to Equation 2 for each \( d_{i1}^t \). In step 4, Equation 1 is applied \( T \) times for each unique pair \((d_{i1}^t, d_{i2}^t)\) with \( d_{i1}^t = 0,1,\ldots,T-1 \). Therefore, there is not a duplicated 3-tuple \((s_{i1}^t, s_{i2}^t, s_{i3}^t)\). Hence the algorithm is optimal.

**3. EXAMPLES**

This section shows some examples of the algorithm. Suppose there are four words in the vocabulary. Then we can let \( n=6 \), because \( T(n-2)^t = 4(6-2)^t = 1 \) is an integer. The final number of strings will be \( T(n-2)^t = 4^3(6-2)^t = 16 \). In step 2 of the algorithm, we can make a second order difference string \( A_2 = (a_{i1}^t, a_{i2}^t, \ldots, a_{i6}^t) = (0, 1, 2, 3) \). In step 3, we can make four difference strings \( D_2 = (d_{i1}^t, d_{i2}^t, \ldots, d_{i6}^t) \),

\[
D_2 = ( \begin{array}{cccc}
0 & 0 & 1 & 3 \\
1 & 2 & 0 & 3 \\
2 & 2 & 3 & 1 \\
3 & 3 & 0 & 2 \\
\end{array} )
\]

Note here that \( d_{i4}^t = (d_{i3}^t + 1) \mod T \), and there is not a duplicated pair of consecutive digits \((d_{i1}^t, d_{i2}^t)\).

Finally, we can make \( T^3 / (n-2)^t = 4^3(6-2)^t = 16 \) strings, each containing four tri-words. There is not a duplicated 3-tuple of consecutive digits. We can also see that each pair is included evenly in the list. For example, the pair of
consecutive digits (1,1) is included five times in the list.
This will be discussed in section 4.

```
000102 012003 020300 032201
111213 123110 131011 103312
222320 230221 202122 210023
333031 301332 313233 321130
```

The most practical application of this algorithm will be connected digit recognition. If we use ten digits in all, we only need 200 strings of seven digits long as the following example with the second-order differences:

```
A1 = (0, 1, 2, 3, 4)  
A2 = (5, 6, 7, 8, 9).
```

4. GENERALIZATION OF THE ALGORITHM

In the last section, we only generated tri-word strings. However, this algorithm can be generalized to arbitrary range of contexts. If we want to generate word strings with context of four words long, the result of the tri-words strings can be used as the difference strings. Then, step 4 of the algorithm can be applied to generate \( T'(n-2) \) strings of length \( n+1 \). This process can be applied recursively until the strings of desired range of context is achieved.

4.1. Choosing \( n \)

If we choose \( n \) such that \( T/(n-2) \) is not an integer, the algorithm cannot produce an optimal result. In that case, we have to set the number of second-order difference strings \( A_k = (a_{k1}, a_{k2}, \ldots, a_{kn-2}) \) to be \( \lceil T/(n-2) \rceil \), because there must be all the digits \( 0, 1, \ldots, T-1 \) at least once. Then, there will be some redundancies that result from the duplicated \( a_{ki} \)'s. To remove the redundancies, one of the \( A_k \)'s should have less number of digits than \( n-2 \). That will result in some final strings of less than \( n \) words indices.

As an example, let \( n=5 \) in the example in section 3 where \( T=4 \). Here, \( T/(n-2) = 4/(5-2) = 4/3 \) is not an integer. Therefore, let us make \( \lceil 4/3 \rceil = 2 \) second-order difference strings of length \( n-2=3 \). The final number of strings will be \( T/(n-2) \times T^2 = 2 \times 16 = 32 \). In step 2 of the algorithm, we make two second order difference strings.

\[
A_1 = (0, 1, 2)  
A_2 = (3, 0, 1) 
\]

Note that the last two digits in \( A_2 \) are redundant.

In step 3, we can make eight difference strings \( D_k \),

\[
D_0 = ( 0, 0, 1, 3, )  
D_1 = ( 1, 1, 2, 0, )  
D_2 = ( 2, 2, 3, 1, )  
D_3 = ( 3, 3, 0, 2, )  
D_4 = ( 0, 3, 3, 0, )  
D_5 = ( 1, 0, 0, 1, )  
D_6 = ( 2, 1, 1, 2, )  
D_7 = ( 3, 2, 2, 3, ) 
\]

Here, the redundant digits produced by the redundant second order differences are underlined.
Finally, the final strings will be:

00010 01200 02030 03220
11121 12311 13101 10331
22232 23022 20212 21002
33303 30133 31323 32113
00322 01112 02302 03132
11033 12223 13013 10203
22100 23330 20120 21310
33211 30001 31231 32021

Here, the boxed last two digits in some strings are useless results from the redundant second-order difference strings. The redundant digits can be removed from the list. Moreover, the results can be merged to produce more compact list. The three tri-words 003, 031, 312 can be merged to make a list 00312. In the same way, the short strings can be merged to produce final optimal results:

00312 01102 02332 12213
3201 1300

4.2. More lists

The words series in the final list are determined by the second-order differences string A_k’s in the algorithm. Hence, we can make different lists by just shuffling the digits in A_k’s.

If we want to make different list for each speaker, it is a good idea to make a list of wider contexts. That is, if we make a list of four-word contexts, then each tri-words will be included in the list evenly as shown in section 3, which is not difficult to prove. When we make a list of four-word contexts, D_k’s in the algorithm for tri-words will be used as the second-order difference strings, and S_k’s will be used as the difference strings. The difference list includes each pair of digits evenly because the second-order difference list includes each digit evenly. This can be understood easily from the fact that d^{2k} = (d^{k} +1) mod T. Hence, the final four-word contexts list contains each tri-words evenly. This can be generalized to wider contexts as well, although they may not be practical.

5. CONCLUSIONS

We introduce an algorithm to generate prompts lists for training connected word recognition system. The algorithm generates series of words indices with different left and right words contexts. For a set of words of size T, the algorithm generates a list of T^{l(n-2)} strings, which is optimal. We prove that the algorithm produces an optimal and complete list of tri-words. We also show some examples with T=4 and T=10. The algorithm can be generalized to word strings of wider contexts. The algorithm is shown to be useful even when T(n-2) is not an integer.

Currently, we have recorded the speech data with the prompts lists made by the algorithm, and the performance of the recognizer trained by the speech data is being examined.

6. REFERENCES


