A TWO-STAGE SPEECH RECOGNITION METHOD FOR INFORMATION RETRIEVAL APPLICATIONS

Paolo Coletti and Marcello Federico

ITC-irst - Centro per la Ricerca Scientifica e Tecnologica
I-38050 Povo, Trento, Italy.
{coletti,federico}@itc.it

6th European Conference on
Speech Communication and Technology
(EUROSPEECH'99)
Budapest, Hungary, September 5-9, 1999

ABSTRACT

This paper presents a two-stage approach to speech recognition that is suited for information retrieval tasks, e.g. accessing a large telephone directory. The first stage performs a Viterbi beam search to decode the speech input into a sequence of phonemes. The second stage performs a graph search to match the phoneme sequence with a large list of keywords. The key issue is that the first step employs a syllable based language model that does not necessarily depend on the application domain. Experimental results are shown for a telephone directory access task of one million of entries.

Keywords: multi-stage recognition, syllable language model, lexicon tree, confusion matrices, graph search algorithm.

10.21437/Eurospeech.1999-118

1. INTRODUCTION

This work presents a two-stage speech recognition method suitable for information retrieval applications [1, 3, 5]. Isolated keywords can be recognized and used to access large databases. Potential applications can be the access to large lexicons, telephone directories, WEB pages through a search engine, etc. Important features of information retrieval applications are the use of very large vocabularies of keywords (e.g. more than 100,000 entries), the access of the services through networks, and the need to periodically update the searching indexes.

In our scenario, the user is able to access a database by uttering one or more keywords isolately. A speech recognizer decodes the input into a sequence of phonemes by employing an n-gram syllable language model (LM) [2]. Next, another module searches the index for phonetically similar keywords and returns an ordered list of candidates. Search is performed through a graph search algorithm [6] that exploits a lexicon-tree representation and three confusion matrices that model phoneme substitutions, deletions and insertions.

With respect to the beforehand mentioned issues, the proposed technique allows to keep the speech recognition module separated from the index access module. In fact, the first module could be designed independently and suited for different information retrieval domains. In particular, a syllable based LM makes the complexity of the speech recognition module only weakly bounded to the number of keywords. Moreover, assuming a client-server network architecture, the lightweight recognition module can reside on the client part while the heavyweight searching module can reside on the server and be maintained independently.

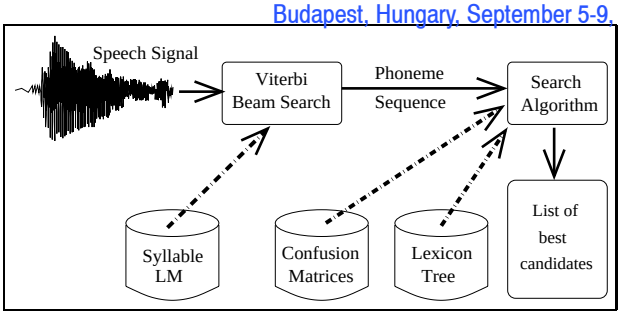

Figure 1: System architecture.

2. ARCHITECTURE

2.1. First stage

The first recognition stage maps the input speech signal into a sequence of phonemes. This stage is performed by employing a trigram LM based on phonetic syllables. The pursued idea is to use possibly general LMs that weakly depend on the application domain. For instance, a phone directory query application could use a syllable based LM estimated on a typical list of proper names, which do not necessarily correspond to those in the searched directory. In addition, the LM could be estimated on a general dictionary just to model phonological rules of the target language. The choice of using syllables derives from the observation that they seem a good level of representation of the sounds of a language. In fact, despite the number of syllables of a language is much larger than that of phonemes, syllables result acoustically better defined than phonemes and are much less than the words of a language.

In order to define a consistent set of syllabic sounds, a set of acoustic syllables has been developed by combining phonetic and orthographic rules. The main idea is to split phonetic transcriptions of words into syllabic items that can be pronounced isolately and, at the same time, can be joined to produce complete word sounds. For instance, about 7,000 phonetic syllables were isolated into a 500,000 Italian word list.

2.2. Second stage

In the second stage, an algorithm, starting from the phoneme string, searches the list of keywords and produces a sorted list of candidates. In order to perform this stage, the search algorithms exploits three confusion matrices and a lexicon tree representation of all the keywords.

Confusion matrices Confusion matrices model typical phoneme errors produced by the speech recognition module. These matrices contain log-probabilities of insertion (`ins`), deletion (`del`) and substitution (`sub`) errors of each phoneme [4]. The confusion matrices are therefore independent from the words which are expected to be recognized and can be generated only once when the syllables set is chosen.

The entries of the matrices are estimated starting from counting computed over a sufficiently large set of samples of recognized keywords. Given two phonemes a and b, the following statistics are collected:

- $s(a,b)$ = number of times b is recognized as a;
- $i(a,b)$ = number of times a is erroneously inserted after b;
- $d(b)$ = number of times b is erroneously deleted;
- $c(b)$ = number of times b occurs in the training set.

The following heuristic constraints have been added to s:

- $s(a,b) \geq 0.3\ c(b)$ when a and b are acoustically very similar (e.g. short versus long vowels);
- $s(a,b) \geq 0.05\ c(b)$ when a and b are single and geminate variants of the same consonant;
- $s(b,b) \geq 0.5\, c(b)$, i.e. the chance of correct recognition must be at least 0.5.

Finally, the matrices sub, ins, and del are defined as follows:

$$sub(a,b) = -\ln \frac{s(a,b)+\epsilon}{c(b)+3v\epsilon}$$

$$ins(a,b) = -\ln \frac{i(a,b)+\epsilon}{c(b)+3v\epsilon}$$

$$del(b) = -\ln \frac{d(b)+v\epsilon}{c(b)+3v\epsilon}$$

where v is the total number of phonemes, and ϵ is a small constant set to avoid undefined values of the logarithm function. Given the above matrices, one can define the following cost function of a given operation o over two phonemes a and b:

$$\mathcal{C}(o,a,b) = \begin{cases} sub(a,b) & \text{if } o = \texttt{sub} \\ ins(a,b) & \text{if } o = \texttt{ins} \\ del(a) & \text{if } o = \texttt{del} \end{cases}$$

Lexicon tree The lexicon tree represents the set of keywords that can be matched against the sequence of phonemes produced during the first stage. The tree structure provides a very efficient representation of the search space and permits to efficiently apply pruning strategies during the search. An example of lexicon tree is shown in Figure 2. Briefly, each node of the tree is labelled by a phoneme symbol and possibly by a string if the path to the node corresponds to the phonetic transcription of a keyword.

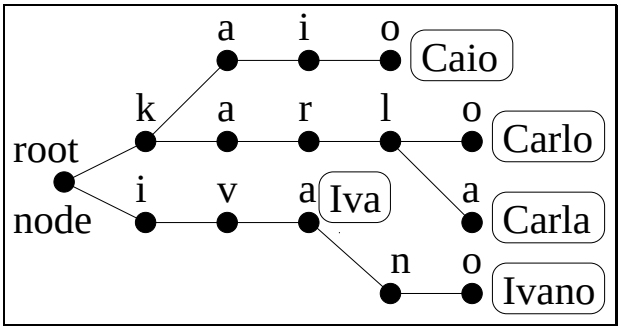

Figure 2: Lexicon tree for five names.

Initialization:

`Theory start(time=1,noper=0,score=0,node=root);`
`List openset(start);`
`List solutions();`

Algorithm:

`while (Theory th=openset.pop())`
`  if IsSolution(th) solutions.Insert(th)`
`  else`
`  while(Expand(th,node,oper))`
`  Theory newth(Time(th,oper),`
`  th.noper+1,`
`  Score(th,node,oper),`
`  node);`
`  if Check(newth) then openset.Insert(newth)`

Table 1: The graph search algorithm.

Search algorithm The search is performed by a graph search algorithm which looks for optimal matches between the input sequence and the set of keywords represented with the tree structure. The algorithm, as shown in Table 1, exploits a list (open set) of partial solutions (theories) that represent different alignments (matches) between stretches (spans) of the input sequence and a path of the tree. As spans and paths always start from the first input symbol and from the root of the tree, they can be uniquely identified by their length and last node, respectively. Hence, a theory is represented by the following quadruple:

- `time`, i.e. the span over the input sequence;
- `noper`, i.e. the number of performed operations;
- `score`, i.e. the cost associated to the partial match;
- `node`, i.e. the node representing the matched tree path.

At each step of the search, the most promising theory in the list is extracted and expanded. The expansion process generates a list of theories that extend either only the path of the theory (deletions), its span (insertion), or both of them (substitutions). Theories are inserted

into the open set list according to an evaluation function. Pruning strategies are introduced that control the number of errors. Finally, whenever a theory's node is labelled with a string and its span covers the entire input sequence, the theory is inserted into the list of solutions.

In the following, the main functions of the algorithm are briefly explained.

`Expand(th,node,oper)`: this function expands a partial theory by applying every allowed operation, i.e `ins`, `del`, and `sub`. It returns the applied operation and the node resulting from the operation. A symbol insertion causes the theory to stay on the same node of `th`, while a symbol deletion or substitution lets the theory advance on the successor node of `th` labelled with that symbol.

`Time(th,oper)`: this function returns `th.time` if `oper==ins`, and `th.time+1` otherwise.

`Score(th,node,oper)`: this function evaluates the score of the expansion of `th` by means of `oper` and `node`. The resulting function sums the cost of all single operations applied so far. Indicating with b the phoneme associated to `node` and with a the phoneme of the input sequence at time `th.time`, the score of the new theory can be incrementally updated:

$$\texttt{Score}(\texttt{th},\texttt{node},\texttt{oper}) = \texttt{th.score} + \mathcal{C}(\texttt{oper},\texttt{a},\texttt{b}). \quad (1)$$

`Insert(newth)`: this function inserts `newth` into a list according to the evaluation function:

$$\begin{aligned} f(th) &= g(th) + h(th) \\ &= \texttt{th.score} + (\texttt{T} - \texttt{th.time})\frac{\texttt{th.score}}{\texttt{th.time}} \\ &\propto \frac{\texttt{th.score}}{\texttt{th.time}} \end{aligned}$$

where `T` indicates the length of the input sequence. The heuristic function h tries to predict the cost of completing the theory by extrapolating from the current cost `th.score`. As the estimate of the completion may be higher than the real one, the resulting search strategy is *not admissible*, i.e. the first found solution could not be the best one. Nevertheless, the resulting evaluation function permits to focus the search on a relatively few set of promising theories.

`Check(newth)`: this function evaluates if a theory is worth to be included into the open set list. The evaluation is based on thresholds that consider the so far performed operations, the score of the best solution and the total number of explored nodes. As a matter of fact, this function allows to significantly reduce the computation time at the cost of losing admissibility of the algorithm.

`IsSolution(th)`: this function checks weather the current theory is a solution, i.e. `th.node` is labelled with a keyword and `th.time` spans the whole input sequence.

3. POTENTIAL APPLICATIONS

The proposed architecture is suitable for a wide range of information retrieval applications, i.e. every situation in which remote users search information on a huge database by means of keywords.

Namely, an application example can be the retrieval of telephone numbers or addresses from the directory of a large town via computer or some hand-hold device. A conventional architecture would require sending the speech signal through a computer net or through the telephone line. In both cases, one would have to face the trade-off between efficient encoding and original content preservation.

The proposed architecture explores the possibility of using a local lightweight recognizer that decodes input speech into a sequence of phonetic symbols, sends it to a remote database server, which tries to retrieve the items with the most acoustically similar keywords. Further advantages of using the local speech recognizer are:

- it can depend on the user's idiom and possibly take advantage of the application domain (e.g. telephone directory inquiry);
- it can be speaker independent or adapted to a single user during usage;
- it can be independent from the content of the queried databases.

Another application domain is the retrieval of e-mail addresses from huge databases, an operation that can actually be performed through the Internet too. Usually these databases are accessible via the LDAP (acronym of Lightweight Directory Access Protocol) protocol. Currently, the LDAP protocol manages even requests that require a search for "orthographic similar" entries. The same principle could be extended to "phonetic similarity" if the input would be represented as a string of phonemes and the database entries would be annotated with phonetic transcriptions.

Also the retrieval of WWW pages by means of a special search engine could be a suitable application for this technique.

4. EXPERIMENTS

The proposed technique has been tested on a telephone directory inquiry task. Experiments have been performed in two conditions. In the first condition (C1) it is assumed that the application domain is known, i.e. the syllable LM has been trained on a huge list of names and surnames. In the second condition (C2), the syllable LM is instead trained on a general sample of the Italian language.

Experiments have been conducted by considering increasing sizes of the searched directory: 20,000, 200,000, and 1,000,000 people. Entries were extracted from the Italian white pages. To ensure a proper geographical coverage of the entries, the entries were randomly chosen from 12 Italian towns.

Performance was measured by letting the phoneme decoder run on a Pentium II 266 Mhz PC, and the lexical search algorithm on a Pentium II 400 Mhz PC.

The training list of C1 is composed of 63,000 Italian surnames and 3,000 Italian names taken from a 3,000,000-entry corpus. This corpus represents a good collection of names and surnames syllables but not necessarily includes every possible Italian name or surname. The training list of C2 is composed of 500,000 Italian words plus the most common foreign words. Words were collected from dictionaries, books, and newspapers.

Each entry of the training lists was phonetically transcribed into the SAM Phonetic Alphabet and split into phonetic syllables. The rules to build syllables are the same as the Italian orthographic rules but with two slight differences. Double consonants are not split since their sound cannot, and single phoneme syllable are joined to the next syllable, since single phoneme syllable are difficult to recognize. As an example, "annesso" (*a nn e ss o*) is split into *anne-sso*. The total number of syllables is about 3,500 and 7,000, respectively. The two resulting syllable corpora were employed to train trigram LMs.

A development set and a test set were acquired to estimate the confusion matrices and test the speech recognizer, respectively. Recordings were carried out in an office environment. The test set consists of 2,000 name-surname utterances by 10 male and 10 female speakers. Before testing each speaker, the confusion matrix was estimated by using a fixed development set of 1,300 utterances plus the test material of the other speakers.

The recognition stage produces phoneme sequences which exactly match the reference sequence in the 57% (C1) and 37% (C2) of the cases. The phoneme accuracies are 91% and 84%, respectively. It is easy to see that wrong phonemes are few but well spread among the words, especially when the LM model is trained with the C2 corpus. The real-time-ratios required by this stage are 0.90 (C1) and 0.87 (C2).

Results of the first stage were used to test the lexicon search algorithm by letting the pruning parameters vary. Among the many possible combinations that were tested, three settings were selected that represent significant trade-offs between efficiency and accuracy. A summary of the performed experiments is shown in Table 3 and Table 2.

num. of entries	real time ratio	Correct recognition 1	1-5	1-10
20K	0.90	82.6	85.5	85.5
20K	0.93	83.9	87.5	87.9
20K	1.22	84.1	88.0	88.3
200K	0.93	77.4	83.0	83.6
200K	1.05	77.9	83.6	84.8
1000K	0.95	72.4	79.5	81.0
1000K	1.26	73.3	79.8	81.5

Table 2: Experimental results with a domain dependent LM and different settings of the pruning parameters.

Memory requirements of the search algorithm for the 20K, 200K and 1M entries were, respectively, 3 Mb, 12 Mb, and 40 Mb. The syllable speech recognizer requires only 8 Mb of memory. Percentages of correct recognized words are given for the first solution, for the first five, and for the first ten solutions. The real-time-ratio includes both recognition stages.

num. of entries	real time ratio	Correct recognition 1	1-5	1-10
20K	0.87	73.8	76.4	76.6
20K	0.97	78.1	83.2	83.4
200K	0.90	68.4	73.8	74.5
200K	1.29	69.2	75.8	76.7
1000K	0.97	63.0	70.0	71.1

Table 3: Experimental results with a domain independent LM and different settings of the pruning parameters.

5. CONCLUSIONS

An architecture for very large vocabulary speech recognition of isolated keywords has been presented. The proposed technique is suited for information retrieval applications on the network. The recognition problem is divided into two stages. First, a lightweight speech recognizer is employed to decode the speech input into a sequence of phonetic symbols. Second, a lexical search algorithm is used to select a list of entries that better match with the input sequence. The key issue of the approach is that the first component can be trained independently from the application vocabulary. In fact, the speech recognizer employs a syllable based language model, that is trained on a huge list of words. Experimental results are shown by considering two different training conditions of the syllable language model: weakly application dependent and completely application independent.

6. REFERENCES

[1] R. De Mori, D. Giuliani, and R. Gretter. Phone-based prefiltering for continuous speech recognition. In *Proc. of ICSLP*, pages 2203–2206, Yokohama, Japan, 1994.

[2] M. Federico, M. Cettolo, F. Brugnara, and G. Antoniol. Language modeling for efficient beam-search. *Computer Speech and Language*, 9:353–379, 1995.

[3] D. A. James and S. J. Young. A fast lattice-based approach to vocabulary independent word spotting. In *Proc. of ICASSP*, vol. 1, pages 377–380, Adelaide, Australia, 1994.

[4] M. D. Kernighan, K. W. Church, and W. A. Gale. A spelling correction program based on a noisy channel. In *Proc. of COLING*, pages 205–210, Helsinki, Finland, 1990.

[5] P. Laface, C. Vair, and L. Fissore. A fast segmental Viterbi algorithm for large vocabulary recognition. In *Proc. of ICASSP*, vol. 1, pages 560–563, Detroit, MI, 1995.

[6] N. J. Nilsson. *Principles of Artificial Intelligence*. Springer Verlag, Berlin, Germany, 1982.