COMBINING SYNTACTICAL AND STATISTICAL LANGUAGE
CONSTRAINTS IN CONTEXT-DEPENDENT LANGUAGE
MODELS FOR INTERACTIVE SPEECH APPLICATIONS

Ute Kilian, Fritz Class
DaimlerChrysler AG, Research and Technology
Wilhelm-Runge-Str. 11, D-89081 Ulm, Germany
ute.kilian@daimlerchrysler.com

ABSTRACT
In interactive speech applications the expected vocabulary and the expected user utterances change from one dialogue step to the next one. The use of several context dependent language models results in a better system performance than the use of a single model. In this paper we present a new approach combining syntactical and statistical language constraints to a single language model. Recognition results on a database of spelled city names are presented. Furthermore a match against the list of all possible city names is performed.

Keywords: language models, spelling applications, subdialogues, interactive speech applications

1. INTRODUCTION
The use of context dependent language models results in a better system performance than the use of a single model (see also [1], [2]), especially if the size of the vocabulary is quite different (e.g. 42000 city names in one dialogue step and the 10 digits for entering the postcode in further step.

In [3] we presented LDS (lexicon development system) – a tool for defining context dependent language models based on sentence templates. This approach is used for new applications if there is no training set available for language model generation. The expected user utterances are defined for each context. Based on that definitions lexicon and language model for a continuous speech recognizer are generated automatically. That language model holds all the syntactical restrictions given by the sentence templates and is transformed to a special bigram language model called SynBi (syntactical bigram, see [4]).

Problems arising with the use of a full coverage grammar like bad formed phrases are well known. Therefore we favored a new approach combining syntactical and statistical language constraints to a single language model: the SynBiTri (syntactical bigram with trigram) where the trigram represents the statistical component.

The aim of this approach is to combine the advantages of both approaches:
- if syntactical language constraints are available, they are used in the language model (e.g. date or time)
- statistical language constraints are used for generalization

2. THE SYNBITRI TECHNIQUE
The principle of the SynBiTri technique is described based on an example of spelled city names. The lexicon of the recognizer consists of 32 letters, and a set of German city names were chosen for language model training. One part of the syntactical constraints given by the set of city names is used explicitly, with the remaining parts being transformed into statistical N-grams.

Example:

A A C H E N
A A L E N
A M B E R G

Suppose these three city names have to be transformed into a SynBiTri, where the syntactical constraints of the first 3 letters will be used explicitly, the remaining parts of the city names will be modeled by a trigram. I.e. the SynBi component of the SynBiTri will consist of

A A C
A A L
A M B
and will result in the subgraph shown in Figure 1
(the nodes AC, AL and MB hold the ‘history’ for
the trigram, i.e. in the case of AC the actual letter is
C with predecessor A).

Figure 1: SynBi component of the SynBiTri

The trigram component will be modeled based on
the letter combinations

\[(A \ C) \ H \ E \ N\]
\[(A \ L) \ E \ N\]
\[(M \ B) \ E \ R \ G\]

the parts in parantheses refering to parts of the city
names that were already modeled by a SynBi, but
they are used as ‘history’ for the trigram.

In order to store the SynBiTri language model as a

Figure 2: Trigram component of the SynBiTri

special kind of bigram, the trigram is transformed

into bigram notation (e.g. AC -> CH refers to the
transition from letter C with predecessor A to the
letter H with predecessor C, see Figure 2).

Combining the SynBi and the Trigram components
of Figure 1 and Figure 2, the result is the SynBiTri
language model shown in Figure 3. This language
model is stored as special kind of bigram language

Figure 3: SynBiTri

model, similar to the SynBi language model
described in [4].

3. EXPERIMENTS AND RESULTS

One important application of this technique is the
spelling mode of a speech recognizer. We
performed experiments on a database consisting of
spelled city names. The complete list of all possible
(German) city names contains about 42000 entries.
The lexicon of the recognizer consists of 32 letters
and some special words like ‘double’ (e.g. the city
name ‘Aachen’ has two correct spellings: ‘A A C H
E N’ and ‘double A C H E N’) or ‘hyphen’ (e.g. in
Bad-Reichenhall).

If we would try to model all of the city names by a
SynBi (what would be theoretically possible), the
syntactical restrictions would be too many to be
stored completely in a language model. Therefore
we decided to use only parts of the syntactical
constraints (e.g. the first 2 to 5 letters) explicitly, with the remaining parts being transformed into statistical N-grams.

The list of all city names was used as basis for language model training. Due to the fact that each city name might have several correct spellings, the list of all city names was transformed into the list of all correct city name pronunciations (allCities).

Example:

City name: Allmannsdorf
Correct spellings:
- ALLMANNSDORF
- A double L MANNSDORF
- ALLMA double NSDORF
- A double L M A double NSDORF

The training set of the speech recognizer consisted of 38000 spelled city names, partially recorded in the car (noisy environment). For testing two different sets of spelled city names were chosen:
- test_2k: 2000 files recorded in the car (noisy environment), not included in the training set
- test_800: 800 files, low noise, included in the training set of the speech recognizer

We will compare our language model experiments with two baseline results (extreme points): the first one is the recognition rate without a language model (noLM), and the second one is the recognition rate with a technique we call LexLM. Using this technique the lexicon of the recognizer consists explicitly of city names, each city name being a sequence of letters. LexLM will cause very good results, the disadvantage, however, is the very large recognizer lexicon.

We carried out the following experiments:
- LexLM: explicit letter sequences of city names
- noLM: no language model was used
- SynBiTri: a combination of a SynBi (first two letters) and a trigram (remaining letters)
  1. trained on allCities
  2. trained on cities_1400, a subset of 1400 cities out of allCities
- CMU_LM: statistical language model, see [5]

The experiments have been done with a speaker-independent continuous speech recognizer (for details see [6]). The results we present refer to the ‘best sentence’ recognition rate (i.e. the completely spelled city name was correctly recognized).

<table>
<thead>
<tr>
<th></th>
<th>test_2k</th>
<th>test_800</th>
</tr>
</thead>
<tbody>
<tr>
<td>LexLM</td>
<td>88.9%</td>
<td>91.2%</td>
</tr>
<tr>
<td>noLM</td>
<td>24.5%</td>
<td>25.1%</td>
</tr>
<tr>
<td>SynBiTri (allCities)</td>
<td>31.2%</td>
<td>33.6%</td>
</tr>
<tr>
<td>SynBiTri (cities_1400)</td>
<td>44.8%</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Recognition results

Without any language model the recognition rate on test_2k was 24.5%. With our SynBiTri trained on the list of all cities, the recognition rate on test_2k was 31.2%. If the amount of expected city names can be reduced (e.g. by a preselection of the region), the training set (cities_1400) is more restrictive causing a much better recognition rate (44.8% on test_2k), see Table 1.

Based on the N-best recognition results we performed a match against the list of all city names (match_42k) and against the subset of 1400 cities (match_1.4k). The matching procedure is based on the Viterbi Algorithm (Dynamic Time Warping) with Levenshtein distance and additionally confusion probabilities of letters. The results are shown in Table 2.

<table>
<thead>
<tr>
<th></th>
<th>test_2k</th>
<th>match_42k</th>
<th>match_1.4k</th>
</tr>
</thead>
<tbody>
<tr>
<td>noLM</td>
<td>24.5%</td>
<td>88.0% (1)</td>
<td>97.6% (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>93.0% (2)</td>
<td>98.8% (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>94.4% (3)</td>
<td>99.2% (3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>95.0% (4)</td>
<td>99.3% (4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>95.7% (5)</td>
<td>99.5% (5)</td>
</tr>
<tr>
<td>SynBiTri (allCities)</td>
<td>31.2%</td>
<td>85.5% (1)</td>
<td>95.0% (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>90.2% (2)</td>
<td>96.6% (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>91.2% (3)</td>
<td>97.1% (3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>92.0% (4)</td>
<td>97.3% (4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>92.5% (5)</td>
<td>97.5% (5)</td>
</tr>
<tr>
<td>CMU_LM</td>
<td>33.3%</td>
<td>81.0% (1)</td>
<td>92.2% (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>86.8% (2)</td>
<td>95.3% (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>88.8% (3)</td>
<td>96.4% (3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>89.9% (4)</td>
<td>97.1% (4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>90.7% (5)</td>
<td>97.3% (5)</td>
</tr>
</tbody>
</table>

Table 2: Recognition results and the corresponding matching results

1 No value is given since the cities of test_800 are not contained in cities_1400
For each match different results are given:
- the best result of the match is correct (1)
- the correct result is among the 2 best alternatives (2)
- the correct result is among the 3 best alternatives (3)
etc.

A surprising result can be seen in Table 2: although the recognition results enhance (no_LM: 24.5%, SynBiTri: 31.2%, CMU_LM: 33.3%) the matching results become worse (no_LM, match_42k: 88.0% (1), SynBiTri, match_42k: 85.5% (1), CMU_LM, match_42k: 81.0% (1)). The same effect can be seen regarding match_1.4k. Our interpretation is the following: the matching algorithm knows that the city name is completely spelled. This information together with the trained confusion probabilities of the letters is better suited to find the correct city name (working on the complete recognition result) than any preceding language model (working only on parts of the city name) and the matching algorithm on the ‘better’ recognition results (with language model).

The result of LEX_LM on test_2k (88.9%, see Table 1) is very close to match_42k with no_LM (88.0%, see Table 2). These two results represent the optimum we gained – both results under the condition that the city names were completely spelled.

Nevertheless, for real life applications it is not realistic to let the users spell city names like NEUNKIRCHEN – SEELSCHEID completely. There will be further information involved in the dialogue (e.g. the region or the area code) to restrict the number of city names in the actual subdialogue. And the spelling of e.g. 5 letters could cause the system to make propositions concerning alternative city names (including all information given so far, the area code and the 5 spelled letters e.g.). In the next subdialogue the user might choose between these propositions or refine the information if it is not yet sufficient.

4. CONCLUSION

We presented an approach called SynBiTri combining syntactical and statistical language constraints in context-dependent language models. The implementation and computational amount of the SynBiTri approach is much lower than that of a statistical trigram. In those cases where no ‘real’ statistics are available (either new applications or in this paper a set of city names without statistics), the SynBiTri and a statistical language model give similar results.

SynBiTri is not restricted to spelling applications. We also use this approach for different interactive speech applications. At the beginning there is no training set available for the language model. We define the first training set using syntactical and statistical constraints. As soon as the application is running, more and more user utterances become available and are used for language model refinement.

5. REFERENCES


