Eurospeech 2003, pp. 2809-2812, ISCA, Geneva, Switzerland, 2003

Combination of CFG and N-gram Modeling in Semantic Grammar Learning

Ye-Yi Wang and Alex Acero

Speech Technology Group
Microsoft Research, One Microsoft Way, Redmond, WA 98052
yeyiwang@microsoft.com, alexac@microsoft.com

Abstract

SGStudio is a grammar authoring tool that eases semantic grammar development. It is capable of integrating different information sources and learning from annotated examples to induct CFG rules. In this paper, we investigate a modification to its underlying model by replacing CFG rules with n-gram statistical models. The new model is a composite of HMM and CFG. The advantages of the new model include its built-in robust feature and its scalability to an n-gram classifier when the understanding does not involve slot filling. We devised a decoder for the model. Preliminary results show that the new model achieved 32% error reduction in high resolution understanding.

1. Introduction

Semantic-based understanding has been successfully used in many research conversational systems [1, 2]. The technology often relies on the manual development of domain-specific grammars, a task that is time-consuming, error-prone and requires a significant amount of expertise. To facilitate the development of speech enabled applications, we introduced SGStudio, an example-based grammar authoring tool that greatly eases grammar development by taking advantage of different sources of prior knowledge [3, 4]. It allows a regular developer with little linguistic knowledge to build a semantic grammar for spoken language understanding.

While experiments have shown that SGStudio not only significantly reduces the effort in grammar development, but also improves the understanding accuracy across several different domains, the technology still has two limitations: it only works well for slot-rich high resolution understanding tasks; and the grammar it generates only works well with robust understanding technology.

In this paper we introduce a new model that combines semantic context free grammar and n-gram language models. The model accounts for the robustness problem itself instead of relaying it to the parser. The model also scales down to accurately handle low resolution understanding tasks.

The remaining part of this section introduces SGStudio. The following sections describe the new model, its training and decoding algorithms, and some experimental results.

1.1. Example-Based Grammar Authoring

SGStudio learns to create a domain-specific grammar from three types of inputs: a semantic schema that defines the semantics of the domain; a grammar library that contains CFG rules for either domain-independent concepts (such as Date and Time) or domain-specific semantic terminals (such as city names and airlines that can be obtained from the application database); and training data with their meaning annotated according to the schema. Below is a simplified example of a ***semantic class*** in a schema that defines the semantics for an appointment scheduling command. It states that to schedule a meeting, a user can (optionally) specify its attendee and start time (***slots*** *of the semantic class*). The slots can be specified with different language expressions that refer to a Person or Time (***types*** *of the slots*).

```
<command name="NewAppt">
    <slot type="Person" name="Attendee"/>
    <slot type="Time" name="StartTime"/>
</command>
```

SGStudio assumes that the syntactic structures are invariable for the type of language used in human-computer interaction. Therefore, it encodes the structural constraints into template rules, and treats the semantic constraints as the template variables. The following template rules are automatically generated by SGStudio for the semantic class NewAppt, where symbols inside braces are optional:

```
<C_NewAppt> ➔ <NewApptCmd> {<NewApptProperties>}                    (1)
<NewApptProperties> ➔ <NewApptProperty>
                          {<NewApptProperties>}                     (2)
<NewApptProperty> ➔ <NewApptAttendeeProperty> |
                    <NewApptStartTimeProperty>                      (3)
<NewApptAttendeeProperty> ➔
    {<PreAttendee>} <Person> {<PostAttendee>}                       (4)
<NewApptStartTimeProperty> ➔
    {<PreStartTime>} <Time> {<PostStartTime>}                       (5)
```

The template grammar models the language for appointment scheduling with a command part followed by properties (1). The properties part incorporates the slots in the schema (2, 3). It brackets each slot with a preamble and a post-amble (4, 5). The commands, preambles and post-ambles are not defined in the template grammar. The language expressions for them are learned from the semantic annotations. The annotation for the sentence "New meeting with Peter at five" is shown below as an example:

```
<NewAppt>                                                           (6)
    <Attendee type="Person">Peter</Attendee>
    <StartTime type="Time">five</StartTime>
</NewAppt>
```

Since the mapping from schema to template rules is one-to-one, a semantic annotation can be mapped to a CFG parse tree. The annotated slots serve as the anchor points in the mapping, and the rest of the words in the input can be aligned to the pre-terminals in the parse tree according to their positions relative

to those anchor points. For example, given annotation (6), the word "at" has to align to the pre-terminals <PostAttendee> or <PreStartTime>, the only two pre-terminals that can appear between the two slots (<Attendee> "Peter" and <StartTime> "five"). Since "at" is a preposition and has to go with the phrase after it, SGStudio inducts the rule <PreStartTime> → at. Similarly, the words "new meeting with" have to align to the pre-terminals in front of "<Attendee> Peter", i.e., <NewApptCmd> and <PreAttendee>. It is hard to automatically determine from this single example the segmentation point that separates the words for <NewApptCmd> from those for <PreAttendee>. However, when multiple examples of pre-terminal sequence and word sequence pairs are available, the EM algorithm we introduced in [5] can learn from those examples to find the most probable segmentations. Once a segmentation point is found, the corresponding lexical rules are added into the grammar. In the previous example, the algorithm finds the position between "meeting" and "with" is the best segmentation point, therefore it adds the following two rules:

<NewApptCmd> → new meeting
<PreAttendee> → with

2. CFG and N-gram Combined Model

SGStudio has been used to develop the MiPad [2] and ATIS grammars. In both cases, it not only significantly reduced the human involvement, but also achieved better understanding accuracy. However, two limitations exist:

1. It only works well with slot-rich high resolution tasks. SGStudio depends on the annotated slots serving as the anchor points. It uses the anchor points to divide a training sentence into smaller units, and localize the segmentation ambiguities in a smaller context. This effectively reduces the requirement for data in EM segmentation; and makes it generalize well since it models with smaller units. When no slots are available, for example, when NewAppt does not have the slots Attendee and StartTime, it simply remembers the entire sentence by introducing rules like "NewApptCmd → new meetings with Peter at five." It does not generalize at all.

2. The grammar does not model the language robustly. When training data are limited, it depends on robust parser at runtime to get good coverage. This restricts the grammar from being used as the language model for speech recognition. Moreover, the robust parser relaxes the constraints of the grammar (model). As a side effect, it introduces ambiguities and accepts invalid inputs. Therefore there must be a trade-off between robustness and precision.

We propose to replace CFG rules with an n-gram to model each of the commands, preambles and post-ambles in the template grammar, and to use n-gram to model the slot transitions. The resulting model is a composite of an HMM and a CFG. The HMM models the template rules and the n-gram pre-terminals; the CFG models the library grammar, as illustrated by Figure 1.

In this model, the meaning of an input s can be obtained by finding the Viterbi semantic class c and the state sequence σ that satisfy

$$\begin{aligned}(c,\sigma) &= \arg\max_{(c,\sigma)} P(c,\sigma \mid s) = \arg\max_{(c,\sigma)} P(c,\sigma,s) \\ &= \arg\max_{(c,\sigma)} P(c) \times P(s,\sigma \mid c)\end{aligned} \tag{7}$$

The new model overcomes both of the limitations of the CFG model. For low resolution understanding (task classification), no property pre-terminals will be introduced into the template grammar. Therefore all training data are used to train/smooth the n-gram for the command pre-terminals. The model scales down nicely to an n-gram classifier below, which achieved comparable accuracy as other statistical classifiers [6]:

$$\begin{aligned}\hat{c} &= \arg\max_{c} P(c)P(s \mid c) \\ &= \arg\max_{c} P(c)\prod_i P(w_i \mid, w_{i-1}, w_{i-2}, \ldots, w_1;\ c\text{Cmd})\end{aligned} \tag{8}$$

The n-gram model does not require exact rule match. Instead of making binary decisions about rule applicability, it compares the probability that the observed word sequence is generated from a state (pre-terminal) sequence to find the most likely interpretation. Therefore, the model itself is robust, and there is no need for robust parser any more.

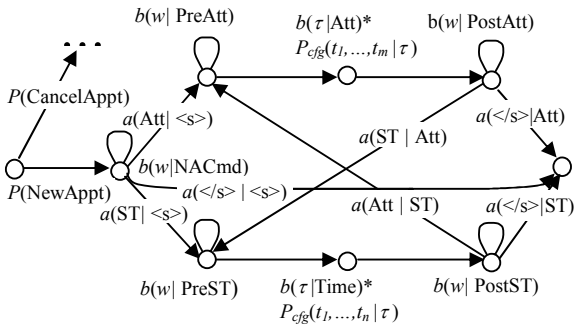

Figure 1. HMM representation of the template grammar. Att *abbreviates for* Attendee*, and* ST *abbreviates for* StartTime*. The emission probabilities b are pre-terminal-dependent unigrams, and the transition probabilities a are the slot transition bigrams. The emissions τ from a slot node are library CFG non-terminals. Words will be generated from them according to the CFG model P_{cfg}.*

3. Model Training

The training algorithm for the new model is an extension of the EM algorithm used by SGStudio for automatic string segmentation [5]. In section 1.1, we described that pairs of pre-terminal (command, preamble or post-amble) sequence and word sequence can be harvested from the annotated data. The EM algorithm then automatically segments the word sequence, aligns each segment α to the corresponding pre-terminal *NT* in the pre-terminal sequence of the same pair. It builds a model $P(NT \to \alpha)$ that assigns probability for generating word string α from *NT*, and parameterizes it with an initial uniform distribution. It then iteratively refines the parameterization. In each iteration, it computes the expected count for the rule $NT \to \alpha$ according to the parameterization of the model in the previous iteration (E-step), and then re-estimates the probability $P(NT \to \alpha)$ by normalizing the expected counts (M-Step). To train the new model that

models the pre-terminals with n-grams, we simply use the expected counts to train the n-grams in the M-step. This results in the following algorithm:

Initialize the model λ with uniform parameterization
do {
  foreach $NT \rightarrow \alpha$ in λ
    Compute the expected count $C(NT \rightarrow \alpha)$ with DP described in [5]
  foreach *NT,* set its n-gram parameters in the new model λ':
    Partition all the rules for NT into training and held-out sets;
    For the rules $NT \rightarrow \alpha$ in the training set, train the n-gram model for NT using α with the expected count $C(NT \rightarrow \alpha)$
    Estimate the model smoothing parameters with the held-out counts via deleted interpolation [7].
} **while** (Perplexity(Sample | λ) – Perplexity(Sample | λ') > threshold)

In our experiments, we set threshold = 0.01.

4. Decoding Algorithm

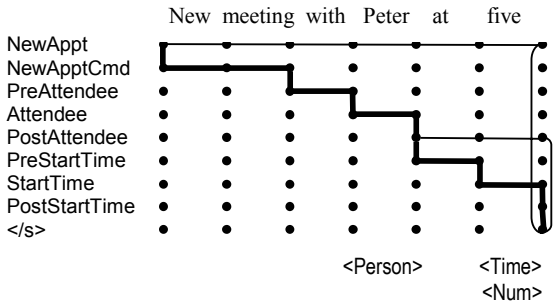

Figure 2. DP Trellis. Below the trellis are the non-terminal identified by the chart parser. The thick path is the Viterbi interpretation. The higher thin path identifies the correct task but neither of the slots. The lower thin path (sharing parts of the Viterbi path) identifies the Attendee *but not the* StartTime *slot. It treats "at five" as the post-amble for* Attendee.

To find the Viterbi path in Equation 7 for an input, we devised a dynamic programming (DP) decoder. Figure 2 illustrates the DP trellis structure for the input "New meeting with Peter at five". Upon receiving the input, the decoder first uses a bottom-up chart parser to find the library grammar non-terminals that cover some input spans. In this example, it identifies "Peter" as <Person> and "five" as either <Time> or <Num>. The decoder then searches through the trellis, starting from the semantic class nodes at the first column (the example only shows the semantic class NewAppt). At each node, it makes transitions to other nodes in the same column (switching to a different non-terminal) or to the next node in the same row (consuming an input word by the non-terminal.) The search continues from left to right until it reaches the rightmost column. When it makes a transition, a score is obtained by adding an appropriate log probability to the score of the starting node. The score is then compared with that of the destination node, and replaces it if the new score in higher. The log probabilities for each of the first 9 transitions are listed below for the Viterbi path in Figure 2.

1. log *P*(NewAppt) // Class Prior
2. log *b*(New | <s>; NewApptCmd) // Word bigram
3. log *b*(meeting | new; NewApptCmd) // Word bigram
4. log *b*(</s> | meeting; NewApptCmd) + // Word bigram
   log *a*(Attendee | <s>; NewAppt) // Slot bigram
5. log *b*(with | <s>; PreAttendee) // Word bigram
6. log *b*(</s> | with; PreAttendee) // Word bigram
7. log P_{cfg}(Peter| <Person>) // PCFG
8. 0
9. log *b*(</s> | <s>; PostAttendee) + // Word bigram
   log *a*(StartTime | Attendee; NewAppt) // Slot bigram

A simple pruning mechanism was used such that at each column of the trellis, no transition would be made out of a node if its score is smaller than a threshold (5.0) less the maximum score in the same column. In other words, a path is not extended if it is 10^5 times less likely than another that leads to a node in the same column. The decoder runs an order of magnitude faster than the robust parser after pruning.

5. Experimental Results

We conducted experiments with the ATIS3 set A data. We used the ATIS3 1993 set A test data for testing. Since some tasks in the test data do not get any training samples, we followed the practice in [4] to augment the training set with nine sentences.

We constructed the semantic schema for ATIS by abstracting the CMU Phoenix grammar for ATIS. Training and test sentences were annotated according to the schema. The resulting canonical meaning representations were used as the gold standard in SLU evaluation. We used our robust parser [2] and the SGStudio derived CFG as the baseline system, and compared its accuracy with the CFG/n-gram models.

We studied the topic classification (henceforth Task ID) and slot identification (henceforth Slot ID) performance. Task ID performance was measured by comparing the parser/DP decoder found top level semantic class with the corresponding manual label. In slot ID evaluation, slots were extracted from a semantic parse tree by listing all the paths from the root to the pre-terminals, and the resulting list was compared with that of the manual annotation. Hence a task ID error makes all the slots in the parse tree incorrect. The total insertion-deletion-substitution error rates are reported for slot ID.

	Task ID	Slot ID
Unigram Classifier	3.68	---
Bigram Classifier	3.22	---
SGStudio CFG	2.07	7.67
Combined/Unigram	3.68	7.50
Combined/Bigram	2.30	5.14

Table 1. Task classification and slot ID error rates.

Table 1 compares the error rates between the CFG and the combined models. For reference, we also list the error rates of the n-gram (n=1, 2) classifiers for task ID. For the model that uses bigrams for the pre-terminals, the task ID error rate is about the same as the CFG/robust parser --- the new model is one sentence worse, which is not statistically significant according to the sign test. The CFG/bigram model reduced slot ID error rate by 32%. Compared to the n-gram classifiers that use words as input features, the CFG/unigram model achieved the same task ID error rate as the unigram classifier, while the CFG/bigram model reduced the classification error by 26% over the bigram classifier. This suggests that correctly identifying slot helps improve task classification.

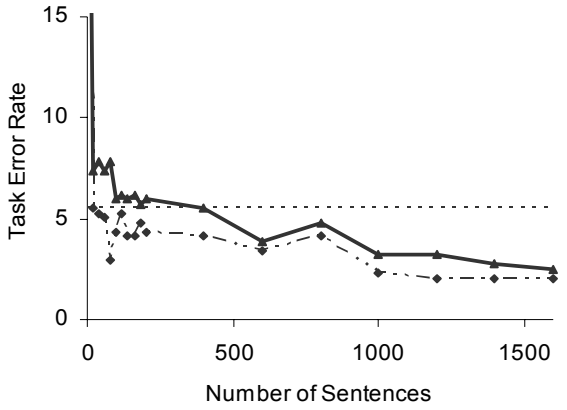

Figure 3. Task ID error vs. amount of training data. The dashed curve represents the SGStudio trained CFG; the solid curve represents the CFG/n-gram combined model. The horizontal line represents manually authored grammar.

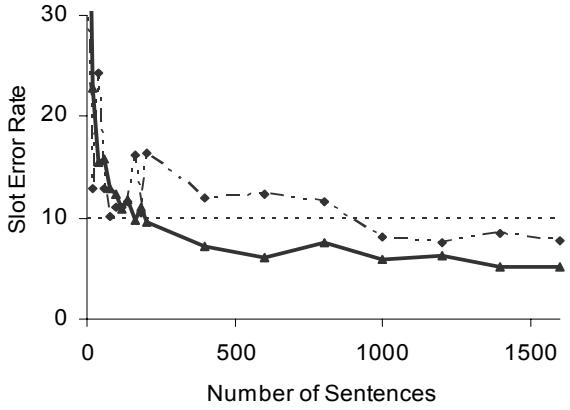

Figure 4. Slot error rate (ins-del-sub) vs. amount of training data. The dashed curve represents the SGStudio trained CFG; the solid curve represents the CFG/n-gram model. The horizontal line represents manually authored grammar.

We also investigated the effect of the amount of training data on the understanding accuracy (Figure 3 and Figure 4). Both models have error rate dropped significantly at the early stage of learning. It is interesting to note that the curves for the new model are smoother than the curves for the CFG/robust parser. This is due to the fact that the new decoder directly uses the statistical model and abides by its constraints, therefore the more the training data are, the better the accuracy is. The robust parser, on the other hand, does not do exactly what the model expects due to model relaxation. Therefore the correlation of the training data and the accuracy is not as strong as the new model.

6. Discussions and Conclusions

HMM has been used for SLU in conversational systems before [8, 9]. Our model resembles the Hidden Understanding Model (HUM) [9]. The "indicators" in HUM functions similarly as our preambles. Both were modeled with n-grams. The major difference is that our model does not try to learn everything from data. Instead we take advantage of grammar library. Because of that, the semantic structure exposed to the user is much simpler. For example, it is up to the library grammar to figure out what type of Date the word "Friday" is, while the HUM requires developers explicitly annotate it as DayOfWeek. For the same reason, our model requires much less data to get satisfactory accuracy. On the other hand, since there is no guarantee that a third party library grammar is finite state, our model has to be a composite of HMM and CFG; while HUM is purely an HMM. The inclusion of post-ambles in our model makes it more precise --- a preamble-only model will not account for the words appearing after the last slot. Modeling in finer granularity also makes the model generalize better. The introduction of post-ambles in our model results in segmentation ambiguities. So we have to use the EM algorithm to estimate the n-gram parameters, while HUM uses direct ML estimation.

In conclusion, the combined CFG/N-gram model overcomes the robustness and the scalability problem of the semantic grammar model used in SGStudio. It improves the understanding accuracy. The dynamic programming decoder runs an order of magnitude faster than the robust parser.

7. Acknowledgements

The authors would like to thank Ciprian Chelba for providing the n-gram training code, Asela Gunawardana for providing the FST toolkit used in an early experiment, and the members of the Microsoft speech group for the feedback.

8. References

[1] Ward, W. *Recent Improvements in the CMU Spoken Language Understanding System*. *Human Language Technology Workshop*. 1994. Plainsboro, New Jersey.

[2] Wang, Y.-Y. *Robust Spoken Language Understanding in MiPad*. *Eurospeech*. 2001. Aalborg, Denmark.

[3] Wang, Y.-Y. and A. Acero. *Grammar Learning for Spoken Language Understanding*. *IEEE workshop on Automatic Speech Recognition and Understanding*. 2001. Madonna di Campiglio, Italy.

[4] Wang, Y.-Y. and A. Acero. *Evaluation of Spoken Language Grammar Learning in ATIS Domain*. *ICASSP*. 2002. Orlando, Florida.

[5] Wang, Y.-Y. and A. Acero. *Concept Acquisition in Example-Based Grammar Authoring*. *ICASSP*. 2003. Hong Kong, China.

[6] Wang, Y.-Y., et al. *Combination of Statistical and Rule-Based Approaches for Spoken Language Understanding. ICSLP*. 2002. Denver, Colorado.

[7] Jelinek, F. and E.L. Mercer, *Interpolated Estimation of Markov Source Parameters from Sparse Data*, *Pattern Recognition in Practice*, D. Gelsema and L. Kanal, Editors. 1980, North-Holland.

[8] Pieraccini, R. and E. Levin, *Stochastic Representation of Semantic Structure for Speech Understanding.* Speech Communication, 1992. **11**.

[9] Miller, S., et al. *Hidden Understanding Models of Natural Language. The 31st Annual Meeting of the Association for Computational Linguistics*. 1994. New Mexico State University.