Special Workshop in Maui, Maui, Hawaii, 2004

A SEMANTICALLY STRUCTURED LANGUAGE MODEL

Alex Acero, Ye-Yi Wang and Kuansan Wang

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

ABSTRACT

In this paper we propose a semantically structured language (SSLM) model that significantly reduces the authoring load required over the traditional manually derived grammar when developing a spoken language system. At the same time, the SSLM results in an understanding error rate which is roughly half as large as that of the manually authored grammar. The proposed model combines the advantages of both statistical word n-grams and context-free grammars. When the SSLM directly acts as the recognizer's language model there's a significant reduction in understanding error rate over the case where it is applied only at the output of a recognizer driven by an word n-gram language model.

1. INTRODUCTION

In recent years, speech recognition systems have been used in interactive voice response systems, desktop dictation, car navigation and many other application domains. In these cases, the speech recognizer is simply one (albeit an important one) component within a spoken language system. In the last decade the availability of high powered affordable personal computers has fueled the progress of speech recognition technology. Despite this progress, state-of-the-art machine speech recognition still underperforms human recognition. While the mathematical models used in the recognition process are pretty powerful, they typically make assumptions that speech scientists know are too simplistic. Moreover, the loose coupling typically done between the recognition and understanding phases is also probably suboptimal. In this paper we propose a semantically structured language model that combines language modeling with understanding.

Language models for speech recognition typically belong to two categories: word ngrams and context free grammars (CFG). These two models differ in many areas: their application areas, generalization capabilities, coverage characteristics, authoring load, and semantic content. It would be desirable to come up with a model that combines the advantages of both. Prior work in such area includes a model that allows CFG fragments to behave like words in a statistical n-gram [7], a syntactically structured language model [1] and the Hidden Understanding Model [5] among others.

The paper organization is as follows. Section 2 reviews the language models used in speech recognition whereas Section 3 describes the main algorithm in the paper, the semantically structured language model. Experimental results are shown in Section 4.

2. REVIEW OF LANGUAGE MODELING FOR SPEECH RECOGNITION

Speech recognition is the problem of assigning a word sequence *W* to an acoustic signal *X*. Most often this is done by finding the word sequence $\hat{W}$ that maximizes the posterior probability:

$$\begin{aligned}\hat{W} &= \arg\max_W p(W \mid X) = \arg\max_W p(W)p(X \mid W) \\ &= \arg\max_W \left[\ln p(W) + \ln p(X \mid W)\right]\end{aligned} \tag{1}$$

where we have used Bayes rule to decompose the posterior into $p(X \mid W)$, the acoustic model, and $p(W)$, the language model. The main mission of the speech recognizer is to search all possible word sequences and select the one with highest probability.

The acoustic model is normally based on a generative model, typically using Hidden Markov Models (HMM). Because of the incorrect assumptions of this modeling technique, this acoustic probability value tends to be much less reliable than the language model probability. Because of that, Eq (1) is often modified to be

$$\hat{W} = \arg\max_W \left[\ln p(W) + \lambda \ln p(X \mid W)\right] \tag{2}$$

where $\lambda = 1/LW$ and LW, the language weight, is larger than 1 to reflect the lower weighting to the acoustic model.

The job of the language model is to assign a probability to a word sequence. There are two different types of language models used in speech recognition depending on the application: word n-grams and context free grammars (CFG).

2.1. N-grams

We can use conditional probabilities to express $P(W)$ as

$$\begin{aligned}P(W) &= p(w_1, w_2, \ldots, w_N) \\ &= p(w_1)p(w_2 \mid w_1)p(w_3 \mid w_1 w_2)p(w_N \mid w_1, w_2, \ldots, w_{N-1})\end{aligned} \tag{3}$$

The so-called word *n*-grams assume that the probability of a word depends exclusively on the past $n-1$ words. Word trigrams are the most likely used language model for speech to text applications:

$$p(w_n \mid w_1, w_2, \ldots, w_{n-1}) \simeq p(w_n \mid w_{n-2}, w_{n-1}) \tag{4}$$

These probabilities can be estimated from a text corpus through maximum likelihood

$$p(w_n \mid w_{n-2}, w_{n-1}) = \frac{C(w_{n-2}, w_{n-1}, w_n)}{C(w_{n-2}, w_{n-1})} \tag{5}$$

where $C(w_{n-2}, w_{n-1}, w_n)$ represents the number of times the sequence (w_{n-2}, w_{n-1}, w_n) appeared in the corpus. For a typical vocabulary size of 100,000 this would require estimating 10^{15} values. Even on corpora with billions of words, many such trigrams would not be present and thus would be assigned a zero probability according to Eq. (5) and thus can never be hypothesized by the recognizer. It is also possible $C(w_{n-2}, w_{n-1}) = 0$ for some bigrams under which, Eq. (5) is not defined. Because of these reasons, smoothing techniques are needed that reserve some of the probability mass estimated by the ML estimate and distribute it across unseen n-grams to avoid a 0 value:

$$\bar{p}(w_n \mid w_{n-2}, w_{n-1}) = \begin{cases} p_d(w_n \mid w_{n-2}, w_{n-1}) & \text{if } C(w_{n-2}, w_{n-1}, w_n) > 0 \\ \gamma(w_{n-2}, w_{n-1})\bar{p}(w_n \mid w_{n-1}) & \text{if } C(w_{n-2}, w_{n-1}, w_n) = 0 \end{cases} \tag{6}$$

where $\gamma(w_{n-2}, w_{n-1})$ is the scaling factor to make the conditional distribution sum up to 1 and $p_d(w_n \mid w_{n-2}, w_{n-1})$ is a discounted version of Eq. (5) where the counts used are reduced by some amount to free probability mass for smoothing.

Word trigrams are the choice of language models for speech dictation systems. On the other hand, the use of word n-grams often results in errors that are ungrammatical, as proper grammar cannot be enforced by simply looking at the last two words: agreement, anaphora and all the basic relationships in natural language have dependencies that go beyond two words. Moreover, the use of word n-grams results in errors that are often nonsensical, since semantics also have a dependency that goes beyond the last two words. Finally, such system offer poor generalization. Shortly after a toddler is taught that "dove" is a bird, she has no problem in using the word dove properly in many contexts that she hasn't heard before; yet training n-gram models require seeing all those n-grams before. Despite all these deficiencies, they work reasonably well and remain the method of choice for most systems.

2.2. Context Free Grammars

Context-free grammars (CFG) are the language model of choice for most of today's interactive voice response applications through the telephone. CFGs are used in command-and-control (C&C) scenarios where the goal is to convert a speech signal into a particular command with possible slots or variables. Often the full power of CFGs is not needed, so Finite State Grammars (FSG) are used instead because of their computational advantages.

One simple example of a CFG used in an interactive voice response system (IVR) to capture an affirmative or negative response is as follows:

```
<Affirmative> -> Yes
<Affirmative> -> Yeah
<Affirmative> -> Sure
<Affirmative> -> Yes, please
<Negative>    -> No
<Negative>    -> No thanks
<Negative>    -> No way
<Negative>    -> Nope
```

In this case, the recognizer can convert the speech signal into text (any of the 8 text strings above can be recognized). In addition, the recognizer also returns the rule that resulted in the highest probability. Such rule contains semantic information that allows an application to reason or act upon it. In the example above it lets the application determine the next branch in the dialog.

CFGs also offer good generalization. For example, the following CFG

```
<ShowFlight> • Flghts from <city> to <city>
<city> • Seattle | Denver | New York
```

can be modified to show flights to many other cities by simply adding more city names into the second rule.

CFGs are built by hand, and thus are labor intensive. Although we have listed many possible ways of saying yes/no in the above grammar, we haven't covered all. Creating a grammar that covers most ways users' responses is a difficult problem. For a real application such grammars can be quite complicated.

Table 1 shows a comparison between CFGs and n-grams. The last column shows the desirability of a hybrid model that combines the advantages of both approaches. Such model, a semantically structured language model (SSLM) is the object of this paper.

	CFG	**n-gram**	**SSLM**
Application	C&C	Speech-text	*both*
Generalization	good	poor	*good*
Build process	manually	from corpus	*from corpus*
Weights	no	statistical	*statistical*
Semantic	yes	no	*yes*
Coverage	poor	good	*good*

Table 1. Comparison between CFG and n-gram language models, with the proposed SSLM.

3. SEMANTICALLY STRUCTURED LANGUAGE MODELS

In this section we define a semantically structured language model that not only provides a probability for a word string W as in Eq. (3), but also provides semantics S:

$$P(W,S) = P(S)p(W \mid S) \tag{7}$$

We propose to model semantics S as a set of labels (one per word) out of a discrete alphabet with M symbols:

$$S = (S_{i_1}, S_{i_2}, \ldots, S_{i_N}) \tag{8}$$

and model $P(S)$ as an n-gram[1]:

$$P(S) = P(S_1)\prod_{i=1}^{N-1} P(S_{i+1} \mid S_i) \tag{9}$$

Finally, we model word strings given semantics as a conditional ngram:

[1] We chose bigram for Eq. (9) and the rest of this work but a longer range n-gram is also possible.

$$P(W \mid S) = P(w_1, w_2, \ldots, w_n \mid S)$$
$$\simeq \prod_{\forall i_n} P(w_1 \mid S_{i_1}) P(w_2 \mid w_1, S_{i_2}) \ldots P(w_n \mid w_{n-2} w_{n-1}, S_{i_n}) \quad (10)$$

where we assumed that the word n-gram depends only on the semantic label of the current word:

$$P(w_n \mid w_{n-2} w_{n-1}, S) \simeq P(w_n \mid w_{n-2} w_{n-1}, S_{i_n}) \quad (11)$$

Next we show in more detail what the model looks like. First we define semantic schema, then explain the network topology and finally the learning algorithm.

3.1. Semantic Schema

Domain semantics are defined through a semantic schema. Below is a simplified example of a *semantic class* in a schema that defines the semantics for the air traffic information system (ATIS) domain [3].

```
<task name="ShowFlight">
   <slot type="City" name="ACity"/>
   <slot type="City" name="DCity"/>
</task>
<task name="GroundTransport">
   <slot type="City" name="City"/>
   <slot type="TransType" name="TType"/>
</task>
```

The schema simply states that the application supports two types of information queries: those for flight information (the ShowFlight task) and those for ground transportation information (the GroundTransport task). To evaluate a flight information query, both the departure city (DCity) and the arrival city (ACity) slots are needed. The type of a slot specifies the requirement for its "fillers". For both DCity and ACity slots, the filler must be an object of the type "City". In this case the semantic values for "City" would be defined by the application backend which, in the case of ATIS, would be the 3-letter airport code. As we can see, the semantic labels of Eq. (8) could be hierarchical as both DCity and ACity refer to another type.

3.2. Network Topology

The next step is to automatically generate the network topology from the semantic schema. The semantic constraints in the schema are incorporated in the understanding grammar with the HMM illustrated in Figure 1. The top level HMM has two branches to the ShowFlight and GroundTransport sub-networks. The transition weights on the branches are the probabilities for the two tasks.

The ShowFlight network at the bottom of Figure 1 models the linguistic expressions that users may use to issue a ShowFlight command. It starts with a command part (e.g., "Show me flights"), followed by the expressions for the slots. Each slot is bracketed by a preamble and a post-amble, which serve as the linguistic context for the slot. For example, the word "from" is a preamble for the DCity slot: it signals that the city following it is likely to be a departure city. The command, preambles and post-ambles in Figure 1 are modeled with word n-grams as shown by Eq. (11).

The slots are inter-connected and the connections are weighted by a slot bigram probability, as in Eq. (9). Such bigrams, if properly estimated, can allow us to determine that "2pm" is an arrival time in "arriving in Seattle at 2pm" and a departure time in "departing Seattle at 2pm" even though the preamble "at" is the same in both cases.

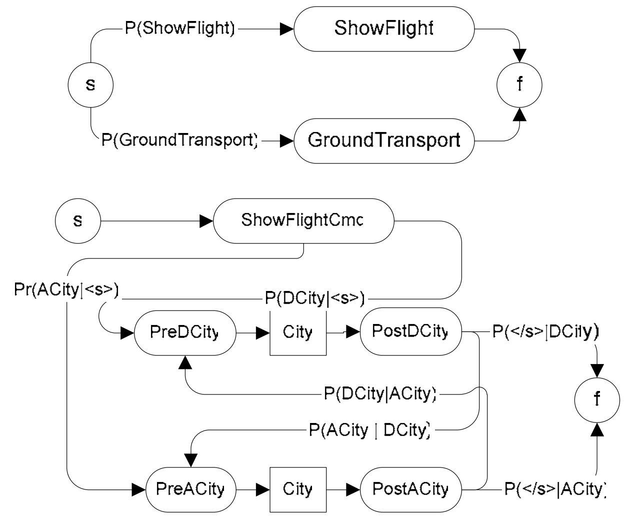

Figure 1. The HMM structure generated automatically from the semantic schema. The upper network is the top level grammar that has two sub-networks. The lower network shows the details of the ShowFlight network. The rectangular blocks are modeled with CFG rules; the rounded rectangular blocks are modeled with n-grams.

3.3. Semantic Entity Grammars and User Modeling

The network corresponding to semantic entities could be modeled by a unigram on the possible realizations of each entity with an additional output symbol[1]. For example, to have the grammar accept "Seattle", we need to add this to the unigram and add an output symbol on that arc with the value SEA, its airport code. We may need to add more than one arc per semantic value (LAX could be referred to both as "Los Angeles" or "LA"). Although a unigram suffices for many entities, some entities require a more sophisticated network. We could choose uniform probabilities or we could use a unigram that reflects the general population or some personalized unigram that matches a user best.

Probabilities for the unigram can be learned from usage data. It's likely a given user is not going to inquire about many cities. In [13] we studied the use of personalization in an email dictation application and observed a very significant reduction in perplexity when these unigram probabilities were learned from the user's sent and received emails. Learning parameters of a grammar from non-text data can be very advantageous as it can complement unavailable text/speech data. Aging [13], using an exponential distribution, was also found to be useful in tracking time-varying distributions.

Date and time are two such examples where, due to the complexity of the grammar, we model them as true CFGs. Since

[1] An alternate representation that retains all the required information that avoids the output symbol is possible by simply preserving the complete parse tree. But the idea of output symbol has already been standardized by the W3C [10] so we will use that in our work.

they are fairly general purpose, they are provided as a library grammar.

3.4. Learning Algorithm

Sections 3.1, 3.2 and 3.3 completely specify the form of the semantically structured language model and in this section we cover the training process.

The model parameters are trained with partially labeled training data as is illustrated below.

```
<ShowFlight text="Show flights from Seattle to
Boston">
    <DCity text="Seattle"/>
    <ACity text="Boston"/>
</ShowFlight>
```

where we note that this labeling does not require a parse tree nor ordering information.

Since the composite model is essentially an HMM (or an HMM with CFGs in some arcs), no closed form solution is available for estimating the model parameters and an iterative algorithm, the EM algorithm [9] is used.

The ngrams in command, preambles and post-ambles are initialized to be the same distribution, in our case a uniform distribution over the vocabulary size. We could have chosen to use a standard dictation language model, or a model created from example sentences from the domain if they are available.

During the E-step we find all possible alignments of words to nodes in the network with their corresponding posterior probabilities, so the alignments are treated as hidden variables.

In the M-step we update all the corresponding (fractional) counts to obtain the updated n-grams $P(w_n \mid w_{n-2}w_{n-1}, S_i)$. Slot bigrams in Eq. (9) are updated similarly. In doing the M-step we need to observe that the typical problems of data sparseness described in Section 2.1 are even more prominent now because the data is fragmented into several n-grams and because in practice the amount of data available to train these models is much smaller than for general dictation. Standard techniques modified to deal with fractional counts [9] can be used in this case to smooth such estimates.

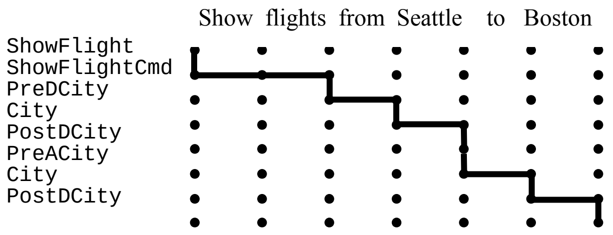

Figure 2. Dynamic Programming Trellis showing the Viterbi interpretation as a solid path. Other paths with lower probability are not shown, yet they are also used in the EM algorithm.

At runtime we used the Viterbi path to find the most likely semantic labels. The Viterbi path for the input "Show flights from Seattle to Boston" is shown in Figure 2. 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 minus the maximum score in the same column. This pruning results in no measurable drop in accuracy but speed the process considerably.

3.5. Semantic Synchronous Understanding

Semantic Synchronous Understanding [6] is a process under which we run the SSLM directly in the speech recognizer and update the semantic values being popped up during the search using the sequential probability ratio (SPR) test:

$$SPR = \frac{P(S \mid x_{1,\dots,}x_t)}{P(S' \mid x_{1,\dots,}x_t)} : \begin{cases} > A & \text{accept} \\ < B & \text{reject} \\ \text{otherwise} & \text{defer} \end{cases} \qquad (12)$$

where the thresholds A and B are chosen given an operating point in the ROC curve determined by the probabilities of false detection P_F and false rejection P_M:

$$A \le \frac{1-P_M}{P_F}, \quad B \ge \frac{P_M}{1-P_F} \qquad (13)$$

where we have to note that some choices of operating points would result in larger latencies (possibly having to wait until the end of utterance) to avoid semantic pruning errors. In practice, a compromise between a small increase in P_F and P_M can be tolerated for faster system's response where the system can spot semantic labels even before the user has finished speaking. This results in a highly interactive feeling.

A side benefit of semantic synchronous understanding is its robustness: it allows us to have an automatic *garbage model* without any explicit models. If the posterior probability of the best semantic label is not high enough, it could be ignored. This means extraneous lexical terms, as well as noises, can be rejected.

Once we combine the effects of the language weight, the sequential probability ratio is given by:

$$L(S) = \frac{\max\limits_{W} \left[p(x_1,\dots,x_t \mid W,S) \right]^{\lambda} P(W \mid S) P(S)}{\sum\limits_{S' \ne S} \max\limits_{W} \left[p(x_1,\dots,x_t \mid W,S') \right]^{\lambda} P(W \mid S') P(S')} \qquad (14)$$

4. EXPERIMENTAL RESULTS

We conducted experiments with the ATIS3 data [3]. We constructed the semantic schema for ATIS by abstracting the CMU Phoenix grammar for ATIS [10]. We used 1700 sentences for training and the ATIS3 1993 set A test data for testing Both training and test sentences were annotated according to the schema. Since some tasks in the test data do not get any training samples we augmented the training set with nine additional sentences fabricated to give sufficient coverage.

For convenience we chose to use a simpler evaluation metric than the one used in [10], which required principles of interpretation and a SQL database. We studied the topic classification (henceforth Task ID) and slot identification (henceforth Slot ID) performance. Task ID accuracy 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 be counted as errors. The total insertion-deletion-substitution error rates are reported for slot ID. We chose the CMU system as a reference because it was the best performing system in the ATIS evaluations.

4.1. Text evaluations

Table 2 compares the error rates between the CFG and the combined models. For reference, we also list the error rates of the Naïve Bayes n-gram (n=1, 2) classifiers for task ID [2]. We can observe that task classification with statistical classifiers, both the Naïve Bayes and the SSLM, is more accurate than with the manually derived CFG. But Naïve Bayes classifiers can only do task classification and cannot extract slots. The SSLM with bigrams can essentially cut the slot error rate in half over the manually derived grammar. The fact that the bigram SSLM had lower task error rate than the unigram version suggests that correctly identifying slot also helps improve task classification.

	Task ID	Slot ID
Unigram Classifier	3.7%	---
Bigram Classifier	3.2%	---
Manual CFG	5.5%	9.9%
SSLM/Unigrams	3.7%	7.5%
SSLM/Bigrams	2.3%	5.1%

Table 2. Task classification and slot ID error rates for text input for the manual CFG, Naïve Bayes classifiers (both unigram and bigram) [2] and the SSLM (both with unigrams and bigrams). Slot error rates are not available with Naïve Bayes classifiers. The Manual CFG was the one used in the CMU system [10].

We also investigated the effect of the amount of training data on the understanding accuracy (Figure 3 and Figure 4). The error rate drops very rapidly as training data increases.

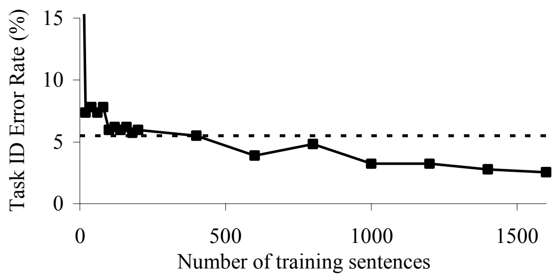

Figure 3. Task ID error vs. amount of training data. The dashed curve represents the learned CFG; the solid curve represents the SSLM. The horizontal line represents manually authored grammar.

SSLM has also been used to develop grammars for MiPad [8], a multimodal personal information manager running on a PocketPC. Both for ATIS and for MiPAD, SSLM not only significantly reduced the human involvement, but also achieved better understanding accuracy.

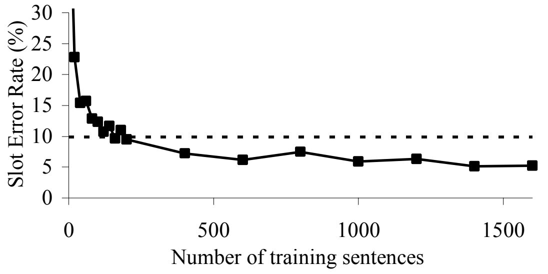

Figure 4. Slot error rate (ins-del-sub) vs. amount of training data. The dashed curve represents the learned CFG; the solid curve represents the SSLM. The horizontal line represents manually authored grammar.

The topology of the new SSLM is a generalization of that of the n-gram classifier for low resolution understanding (task classification), when no property pre-terminals will be introduced. Also, the SSLM runs considerably faster than the manually derived CFG and faster than a robust parser [8] for the same domain.

4.2. Speech Evaluations

Two SSLM models were trained. In the first model the bigrams were not smoothed. In the second model, the bigrams were smoothed with the uniform distribution with deleted interpolation [4]. The test data perplexity of the trigram model is 15.4. For the smoothed SSLM model, the test data perplexity is 16.2 when the likelihood of a sentence is summed over all possible paths in the network. We call it the *Baum-Welch perplexity*. When the likelihood of a sentence is only calculated over the Viterbi path, the resulting *Viterbi perplexity* is 24.1.

The third language model was a standard word trigram built from 5800 sentences from the class A (utterances that can be understood without context) of the ATIS2 and ATIS3 training data. The vocabulary of the model contained 780 words.

We used the three language models for speech recognition. The recognizer was built using HTK [12]. When we used the standard word trigram to drive the recognizer, the one-best result was then fed to the SSLM. When using the SSLM only one-pass recognition and understanding was done. Table 3 shows the results.

	n-gram	unsmoothed SSLM	Smoothed SSLM	Transcription
WER	6.0%	9.2%	7.6%	---
Task ID	6.8%	4.9%	3.8%	2.3%
Slot ID	9.0%	10.3%	8.8%	5.1%

Table 3. Recognition word error rate, task classification error rate and slot identification error rate of the trigram model, the unsmoothed SSLM model and the smoothed SSLM model. The results were obtained with the HTK decoder

Even though the word error rate is over 25% higher than the trigram model, the SSLM model achieved a task classification error rate that is less than half than that of the trigram model, but only a slightly lower slot identification error rate. We noticed that the understanding error rate reduction was even bigger as the word error rates for all the three models became higher when a larger vocabulary was used.

The recognition errors for the SSLM typically occur in the command, preamble and post-amble parts. The sparseness of the training data for a pre-terminal makes the recognition of words underneath the pre-terminal less accurate. Perhaps the choice of a different smoothing (such as backing off to a word trigram) would have alleviated this problem and will be explored in the future. However, since the understanding model is robust, a word error inside this pre-terminal doesn't matter too much as long as it will not flip to another pre-terminal. An example of this is given below:

Reference	find me a flight that flies from Memphis to Tacoma
Trigram	find me a flight that flies from Memphis to Tacoma
SSLM	find me a flight the flights from Memphis to Tacoma

Here although "that flies" was misrecognized as "the flights" with the SSLM model, it did not change its status as the preamble of a flight slot. The meaning was not affected at all.

On the other hand, the trigram model lacks the stricter constraints imposed by the rules in CFG library, therefore the content of a slot can often get recognized incorrectly. This will cause slot ID errors. Since the task ID also depends on the correct slot information, this may adversely affect the task ID accuracy too. Below is the example of this case.

Reference	list the originating cities for Alaska airlines
Trigram	list the originating is the cities for last the airlines
SSLM	list the originating fit cities for Alaska airlines

The optimal language model weight for the SSLM model is 26, which is much higher than that for the trigram model (16). This is because the language model probability mass is split and distributed over multiple ambiguous paths in the SSLM state space, while with the trigram model a word sequence corresponds to a single language model state sequence. Therefore the language model score in a path in the SSLM state space needs to be boosted.

The decoder using the n-gram model ran much faster than the SSLM language model. We are currently optimizing the model structure to make it work faster with speech decoders. We believe that the proper optimization, together with the advances in CFG decoding technology and the continuing growth of computing power, will make this model ready for practical use.

5. DISCUSSION

Our proposed SSLM model resembles the Hidden Understanding Model (HUM) [5] as the "indicators" in HUM functions similarly as our preambles and both were modeled with n-grams. The major difference is that SSLM does not try to learn everything from data and rather SSLM takes advantage of grammar library and thus the semantic structure exposed to the developer 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, SSLM requires much less training data to get satisfactory accuracy. The grammar libraries used in ATIS were finite state but our SSLM will accommodate CFGs and thus is 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 the EM algorithm with are used to estimate the n-gram parameters, while HUM uses direct ML estimation without any hidden variables.

In conclusion, the proposed SSLM drastically reduces the authoring load of creating a speech understanding system while at the same time results in a more accurate, usable system.

6. REFERENCES

[1] Chelba C. "Exploiting Syntactic Structure for Natural Language Modeling". PhD Thesis. Johns Hopkins University. 2000.

[2] Chelba C. et al. "Speech Utterance Classification", in Proc. of the Int. Conf. on Acoustics, Speech, and Signal Processing. Hong Kong, Apr, 2003.

[3] Dahl D. et. al., "Expanding the scope of the ATIS Task: the ATIS-3 Corpus," Human Language Technology Workshop, 1994.

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

[5] 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.

[6] Wang. K. "Semantic Synchronous Understanding for Robust Spoken Language Applications". IEEE Workshop on Automatic Speech Recognition and Understanding. St. Thomas, Dec 2003.

[7] Wang Y. et al. "A Unified Context-Free Grammar and N-Gram Model for Spoken Language Processing", in Proc. of the Int. Conf. on Acoustics, Speech, and Signal Processing. Istanbul, Turkey, June, 2000.

[8] Wang, Y. "Robust Spoken Language Understanding in MiPad". Eurospeech. 2001. Aalborg, Denmark.

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

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

[11] W3C Voice Browser working group. "Semantic Interpretation for Speech Recognition: working draft". http://www.w3.org/TR/semantic-interpretation/. 2004.

[12] Young S. et al. The HTK Book for HTK Version 2.1. Cambridge University, 1997

[13] Yu. D. et al. "Name Recognition with User Modeling". Proc. Eurospeech Conf. Geneva, Sep. 2003.