Prior of the Lexical model in the Hidden Vector State Parser

Filip Jurčiček¹, Jiří Zahradil², and Luboš Šmidl³

¹Center of Applied Cybernetics
²Department of Cybernetics
University of West Bohemia in Pilsen
Pilsen, 306 14, Czech Republic
{filip,jzahradl,smidl}@kky.zcu.cz

Abstract

This paper describes an implementation of a statistical semantic parser for a closed domain with limited amount of training data. We implemented the hidden vector state model, which we present as a structure discrimination of a flat-concept model. The model was implemented in the graphical modeling toolkit. We introduced into the hidden vector state model a concept insertion penalty as a part of pattern recognition approach. In our model, the linear interpolation was used for both to deal with unseen words (unobserved input events) in training data and to smooth probabilities of the model. We evaluated the implementation of the concept insertion penalty in our model on a closed domain human-human train timetable dialogue corpus. We found that the concept insertion penalty was indispensable in our implementation of the hidden vector state model on the human-human train timetable dialogue corpus. Accuracy of the baseline system was increased from 33.7% to 55.4%.

1. Introduction

Statistical approach to spoken language understanding (SLU) gains on popularity. [1] encourage to use more statistical signal processing techniques; even though, their presented system use in some extend expert written context free grammars. They argue for using a mathematical framework that is similar to pattern recognition in the automatic speech recognition (ASR).

We adapted pure statistical approach of [2], who introduced the hidden vector state (HVS) model. In this work, we understand the HVS model as a structure discrimination [3] of a flat-concept model [4], which was developed for the CHRONUS system.

The graphical modeling toolkit (GMTK) [5] is a perfect candidate for implementation of the HVS model. First of all, GMTK has general language to describe dynamic Bayesian networks, which are suitable for implementing models like the HVS model. Furthermore, GMTK was intentionally developed for structure discrimination. Consequently, modifications of the HVS model are easy to perform in GMTK. For instance, [6] experimented with different models in the case of dialogue act tagging. Finally, GMTK has implemented EM estimation [7] so that we can concentrate on modeling instead of implementing EM estimation procedures as [2].

Both the HVS model and the flat-concept model were developed for evaluation in ATIS [8]. Therefore, they deal with utterances, which contain semantics that can be expressed with attribute/value pairs. We see necessity to work with all utterances in a dialogue. This is only way how to incorporate context into future semantic models; as a result, we process all utterances in a dialogue, both operator’s and user’s in our evaluation.

We implemented the HVS model with respect to [2] conclusion in which semantic models should be able to process not only single best hypothesis from a speech recognizer but also hypothesis encoded in a word lattice. The model is possible to transform to FSM; as a result, we can use standard decoding techniques for FSMs [9]. Additionally, this approach is able to produce semantic lattices. Furthermore, the semantic model constructed in the same way is possible to use to constrain ASR.

The paper is organized as follows. Section 2 describes semantic parsing as a statistical pattern recognition. Section 3 introduce HVS model as a structure discrimination of a flat-concept parser. Section 4 elaborates on details of implementation of the HVS model in GMTK. Section 5 evaluates the GMTK HVS model implementation. Finally, section 6 concludes the paper.

2. Statistical semantic parsing

Statistical semantic parsing is a search process to uncover the sequence of concepts $S = EMPT Y, c_1, c_2, . . . , c_T, EMPT Y$ that has the maximum a posteriori probability $P(S|W)$ for the given word observation $W = empty, w_1, w_2, . . . , w_T, empty$. The word/concept utterance boundaries are denoted by symbols empty/EMPTY. The search process can be described as

$$S^* = \arg\max_S P(S|W)$$

$$= \arg\max_S P(W|S)P(S)$$

(1)

where $P(S)$ is the semantic model and $P(W|S)$ is the lexicalization model [1]. The semantic model $P(S)$ provides probability for the semantics $S$. The lexicalization model $P(W|S)$ assigns probability to the underlying utterance (word/lexical sequence) $W$ given the semantics $S$.

2.1. Flat-concept parser

The flat-concept (FC) model is an example of a statistical approach to semantic parsing [4]. It is simple implementation of the equation (1) which is frequently called HMM tagging model. Its graphical model is in Figure 1: the prologue is the first, the epilogue is the last, and the chunk is the repeated dynamic Bayesian network frame unrolled to fit the entire observation length. The corresponding semantic model is

$$P(S) = \prod_{t=0}^{T+1} P(c_t|c_{t-1})$$

(2)
The lexicalization model is

\[
P(W|S) = \prod_{t=0}^{T+1} P(w_t|c_t) \quad (3)
\]

In this model, generation of utterance is modeled by a process in which the hidden states correspond to semantics concepts. The state outputs correspond to individual words. The name flat-concept model comes from its incapability to capture long distance dependencies and hierarchical structure.

3. Hidden Vector State parser

The ability of a semantic model to capture hierarchical structure is beneficial. [2] showed that the HVS model performs better than the FC model. The HVS model is both an extension of the FC model and a variation of a pushdown automaton. A state in the HVS model represents a stack of a pushdown automaton. The stack in this model is called a vector state. The vector state keeps information that spans over several words.

The semantic information corresponding to every word in an utterance is completely described by a sequence of concepts from a leaf to a root of an abstract semantic annotation (see Figure 2). If we place concepts along the way from the leaf to the root to a vector, then a derivation tree can be transformed to a sequence of these vectors. For example, the word Prague is described by the vector state [STATION, TO, DEPARTURE, EMPTY].

Note that the leftmost\(^1\) concept in a stack is always the most recent concept pushed on a stack. Stack concepts may be also indexed in which case the most recent stack concept has index 1 and the oldest concept has index 4.

[2] imposed a hard limit on the maximum depth of a stack\(^2\). We introduce the HVS model with the maximum depth of the stack 4. [2] demonstrated on a corpus in English that an increase in the maximum depth above 4 did not lead to an increase of performance. In addition, the maximum depth 4 is consistent with the hierarchical complexity the abstract semantic annotation of the Czech human-human train timetable dialogue corpus version 2 (HHTT), which we use in our experiments. The version 1 was described in [10]. According to our analysis, the deepest semantic tree has the depth 4.

Furthermore, operations with a stack are limited to further increase the robustness of the HVS model. Because the vector state is equivalent to a stack in a pushdown automaton, the transitions between the vector states are modeled by stack operations. The stack operations include:

- popping 0 to 3 concepts from a stack,
- pushing a new concept onto a stack\(^3\),
- generating a new word.

The first two operations are included in the semantic model. The lexicalization model performs the last operation, generation of a new word. The graphical model of the HVS model is in Figure 3. The solid edges represent probabilistic implementation of pushing a new concept onto a stack and generating a new word. The dotted edges depict determinism of popping 0 to 3 concepts from a stack. The corresponding semantic model is

\[
P(S) = \prod_{t=0}^{T+1} P(n_t|c_{t-1}[1,4]) \cdot P(c_t[1]|c_{t-1}[2,4]) \quad (4)
\]

where \(n_t\) is the vector state shift operation and takes values in range 0, \ldots, 4, and \(c_t\) at word position \(t\) is a vector state of 4 concepts, i.e. \(c_t = [c_t[1], c_t[2], c_t[2], c_t[4]]\), where \(c_t[1]\) is a preterminal concept dominating the word \(w_t\) and \(c_t[4]\) is the root concept. The variable \(n_t\) defines the number of concepts which will be popped of a stack. If \(n_t = 0\), it relates to growing a stack by one concept. If \(n_t = 1\), it relates to replacing preterminal concept \(c_t[1]\) by a new concept. If \(n_t > 1\), it relates to popping \(n_t\) concepts and pushing a new concept. For example, the transition from the vector state represented by the seventh block in Figure 2 is made by popping two concepts TO and STATION and pushing a new concept TIME (\(n_0 = 2\)).

The lexicalization model is

\[
P(W|S) = \prod_{t=0}^{T+1} P(w_t|c_t[1,4]) \quad (5)
\]

The word \(w_t\) is given by \(c_t[2,4]\).

3.1. Structure discrimination of the HVS model

Discriminative parameter training techniques [11, 12] already showed their importance. Represented by Maximum Mutual Information (MMI) training, these methods assume a fixed statistical modeling structure, and then optimize only the numerical parameters (such as means, variances). In contrast, the discriminative structure learning techniques modify the graphical model by adding and removing of nodes and edges [3].

\(^1\)Topmost in Figure 2.
\(^2\)We can benefit from the possibility to transform a pushdown automaton to finite automaton if we limit the maximum depth of a stack of such pushdown automaton.
\(^3\)In case when we have to empty a stack, we pop 3 concepts and push the concept \(c_{empty}\).
The HVS model can be understood as structure discrimination of the flat-concept model. First of all, [2] replaced the nodes of hidden state \( c \) by for 4 nodes for hidden vector state composed of states \( c[1], c[2], c[3], c[4] \). Secondly, they added nodes for hidden variable \( n \). Finally, they added edges among nodes \( c[1], \ldots, c[4] \), and \( n \). In brief, they optimized the structure of the network. Nonetheless, they needed more sophisticated annotation scheme in contrast to [4].

### 3.2. Training data for HVS model

The HVS model is possible to train using semantics that is simple and easy to obtain. In Figure 2, you can see an utterance with an abstract semantic annotation [2], which is DEPARTURE( TO(STATION), TIME).

[10] extended the semantics to cover not only users queries but also the rest of the dialogue. You can find a sample dialogue in [10]. Their definition of abstract semantic annotation covers even utterances that do not have attribute/value pairs. For example, sentences expressing agreement or hesitation.

The abstract semantic annotation is moderate to acquire. We do not need fully annotated treebank data. Dialogue annotators have to define the semantics that represents each training utterance, but they need not provide a full parse tree. A full parse tree define not only the three structure but also the alignment of words to leafs of the tree. In Figure 2, you can see a full parse tree of the previous example. Because abstract semantic annotation is fairly simple, transcribers do not have to have a prior linguistic knowledge (see a sample dialogue in Table 1).

### 4. GMTK HVS model implementation

We implemented our model in GMTK, a DBN system for speech and language.

We divided training of HVS model into three parts: 1) estimation of semantic and lexicalization models, 2) application of a concept insertion penalty, 3) smoothing of semantic and lexicalization models and incorporating the unseen word.

To train the HVS model, we have to use EM algorithm to estimate semantic and lexicalization model because HHTT corpus do not provide fully annotated treebank data in comparison to training data for Hidden Understanding Model (HUM) [13]. Because GMTK provides EM parameter estimation, we do not have to implement EM estimation described in [2]. We use 3 EM iterations to estimate model’s probabilities.

Our implementation of the HVS model differ from the original implementation of the HVS model in application of the concept insertion penalty and in incorporation of the unseen word. In next two sections, we discuss the concept insertion penalty and the smoothing incorporating the unseen word into the model.

#### 4.1. Concept insertion penalty

Weights in combination of the lexicalization model and the semantic model are important. The equation (1) combine the lexicalization model probability \( P(W|S) \) and the semantic model probability \( P(S) \) through simple multiplication. However, there is a difficulty with under estimating the lexicalization model to semantic model [14]. Therefore, two weights are defined to balance both probabilities, a semantic model weight (SW) and a concept insertion penalty (CIP).

\[
S^* = \arg\max_S P(W|S)P(S)^{SW}CIP^{N(S)} \tag{6}
\]

where \( N(W) \) is the number of concepts in the semantics \( S \).

SW change a weight of the semantic model. The concept insertion penalty CIP controls the length of output semantics from a decoder. If CIP is large, the decoder prefers semantics with fewer concepts. If CIP is small, decoder prefers semantics with more concepts. In addition, the purpose of CIP is also to control the side effect of varying SW, which function as an insertion penalty as well [14].

In this paper, we concentrate on CIP. As a result, the search process is given by following formula

\[
S^* = \arg\max_S P(W|S)P(S)^{CIP^{N(S)}} \tag{7}
\]

By modification of probability \( P(n_t|c_{t-1}[1,4]) \), we control the number of inserted concepts. We increase the probability of the stack operation \( n_t = 1 \). When \( n_t = 1 \), a stack content \( c_{t-1}[2,4] \) is copied to \( c_t[2,4] \); as a result, we limit changes of a stack; less concepts are inserted. We applied the insertion penalty according the following formulas

\[
x(n,c) = \begin{cases} 
CIP \cdot P(n|c) & \text{if } n = 1, \\
P(n|c) & \text{otherwise} 
\end{cases}
\]

\[
P_{CIP}(n_t|c_{t-1}[1,4]) = \frac{x(n_t,c_{t-1}[1,4])}{\sum_{n \in \{0, \ldots, 3\}} x(n,c_{t-1}[1,4])} \tag{8}
\]
where \( P_{CI}(n_t|c_t[1,4]) \) is the probability of a stack operation \( n_t \) given a stack \( c_{t-1}[1,4] \) with applied insertion penalty.

We realize that this application of CIP is not perfect. For example, even though the GMTK decoder chooses \( n_t = 1 \) at time \( t \), it does not mean that the whole stack does not change. If \( n_t = 1 \), only \( c_{t-1}[2,4] \) is copied to a new stack \( c_t \) at time \( t \). A new concept \( c_1[1] \) is introduced because the new concept is generated by the probability \( P(c_1[1]|c_2[2,4]) \). The concept \( c_1[1] \) is independent of \( n_t \); as a result, the concept \( c_t[1] \) is not influenced by a value of CIP.

### 4.2. Smoothing

We implemented linear interpolation [7] smoothing into our model. We smooth all three probabilities \( P(n_t|c_{t-1}[1,4]), P(c_t[1]|c_t[2,4]), \) and \( P(w_t|c_t[1,4]) \). For instance, in the case of the probability \( P(w_t|c_t[1,4]) \), the smoothed probability is following

\[
P_{\text{smooth}}(w_t|c_t[1,4]) = \lambda_0 P(w_t|c_t[1,4]) + \\
+ \lambda_1 P(w_t|c_t[1,3]) + \lambda_2 P(w_t|c_t[1,2]) + \\
+ \lambda_3 P(w_t|c_t[1]) + \lambda_4 P(w_t) + \lambda_5 f(w_t)
\]  

where \( f(w) \) is a function that evaluates to 1 if \( w = \text{unseen} \), otherwise 0.

The linear interpolation is suitable for integration of the unseen word into the model. Because we want to simplify work with unseen words, we replace all unseen words in the held-out data and the test data by the word unseen. Obviously, there is no unseen word in the training data. Consequently, we are not able to estimate probability \( P(\text{unseen}|c_t[1,4]) \) on training data. However during smoothing, we are able to estimate \( \lambda_5 \) on the held-out data. The coefficient \( \lambda_5 \) determines what weight the word unseen should have. The purpose of function \( f(w_t) \) is to enable the probability (9) to assign non-zero value to words that were not seen during training.

Another way of seeing the equation (9) is that there exists a hidden variable \( s_{un} \), which is used to switch among different lexicalization models. The graphical model in Figure 4 shows that variable \( s_{un} \) switches the parents of \( w_t \) among \( c_{t-1}[1,4], c_{t-1}[1,3], c_{t-1}[1,2] \), or \( c_{t-1}[1] \). The equation (9) can be there-
5. Evaluation

We use the string edit distance [15] to measure distance between two semantics. First, we transform both reference and hypothesis semantics into strings. We extract a list of all concepts and parentheses from semantics. Every concept and parenthesis is treated as a symbol. The parentheses in semantics are treated as concepts because represent structural information, which we want to evaluate. For example, the semantics \text{DEPARTURE(TO, STATION, TIME)} is transformed to the sequence \text{DEPARTURE, (, TO, (, STATION, ), TIME, )}. Secondly, we count substitutions, insertions, and deletions in the reference and hypothesis strings. In addition, we count number of symbols the reference string. For instance, if we measure string edit distance between the previous semantic sequence and the sequence \text{DEPARTURE, (, FROM, (, STATION, ), AMOUNT, TIME, )}, we get one substitution (TO - FROM) and one insertion (AMOUNT).

The accuracy of a hypothesis is defined as

$$\text{Acc} = \frac{N - S - I - D}{N} \cdot 100\%$$

(11)

where \(N\) is the number of symbols in the reference semantics, \(S\) is the number of substitution errors, \(I\) is the number of insertion errors, and \(D\) is the number of deletion. Accuracy is equal to the percentage of the concepts that were correctly recognized and is penalized for insertion errors. So \(\text{Acc} = \frac{5 - 1 - 1 - 2}{5} = 75.0\%\). We measured the accuracy by the NIST scilite tollkit as well as statistical significance (\(p\)-value).

We can not compare our best result with the result of [2] because HHTT corpus does not have annotated attribute/value pairs; nevertheless, this evaluation would be desirable. If we compare the amount of data available for [2] parser and our parser, we find that we have approximately 5 times smaller training corpus.

We tested our model on abstract semantic annotation from HHTT corpus. Currently, the corpus consists of 862 dialogues completely annotated by abstract semantic annotation. Both operator and users are annotated. The corpus has 13769 utterances in total. The vocabulary size is 2667 words including the word empty. There are 38 semantic concepts including the concept \text{EMPTY} in the corpus. In our experiments, the dialogues were randomly divided into training data (619 dialogues - 9928 utterances, 72%), validation data (69 dialogues - 1108 utterances, 8%), and test data (174 dialogues - 2733 utterances, 20%).

5.1. Baseline system

We performed an experiment with the original HVS model on the test data. No concept insertion penalty was applied. We obtained following results:

<table>
<thead>
<tr>
<th>Del</th>
<th>Subs</th>
<th>Ins</th>
<th>Acc (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>586</td>
<td>768</td>
<td>4950</td>
<td>33.7</td>
</tr>
</tbody>
</table>

Table 2: Results for the baseline system.

5.2. Concept insertion penalty

The insertion penalty was determined automatically on the validation set. We started with CIP equal to five, and in every step we increased the value of CIP by five.

We report 5 results for different CIP measured on the validation data. The results are reported in Table 3. Numbers of deletions (\(D\)), substitutions (\(S\)), insertions (\(I\)), and accuracy (\(\text{Acc}\)) are given.

<table>
<thead>
<tr>
<th>CIP</th>
<th>D</th>
<th>S</th>
<th>I</th>
<th>Acc (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>514</td>
<td>276</td>
<td>824</td>
<td>57.0</td>
</tr>
<tr>
<td>10</td>
<td>665</td>
<td>274</td>
<td>654</td>
<td>57.4</td>
</tr>
<tr>
<td>15</td>
<td>737</td>
<td>280</td>
<td>579</td>
<td>57.5</td>
</tr>
<tr>
<td>20</td>
<td>802</td>
<td>283</td>
<td>533</td>
<td>57.0</td>
</tr>
<tr>
<td>25</td>
<td>836</td>
<td>291</td>
<td>534</td>
<td>55.8</td>
</tr>
</tbody>
</table>

Table 3: Results for the HVS model with different values of CIP on the validation data.

With increasing CIP, we see that the number of insertions is declining. At the same time, the accuracy is increasing until CIP is equal to 15. Surprisingly, we did not see large degradation in accuracy with increasing CIP value above 15. This is probably caused by implementation of CIP. Because the variable \(c_t[1]\) is not affected by CIP, we are not able to completely limit change of \(c_t[1, 4]\).

By analysis of the results, we found that the utterances with the length of semantics equal to one substantially worsen the total accuracy. We conducted experiment when we removed utterances with the length of semantics equal to one from the test set. On the altered test set including 337 utterances, the GGTK HVS model achieved 61.3% in accuracy.

If we evaluate the HVS model with CIP equal to 15 on the test data, we obtain following results:

<table>
<thead>
<tr>
<th>CIP</th>
<th>D</th>
<th>S</th>
<th>I</th>
<th>Acc (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>1807</td>
<td>806</td>
<td>1626</td>
<td>55.4</td>
</tr>
</tbody>
</table>

Table 4: Result for the HVS model with CIP equal to 15 on the test data.

We performed Matched Pairs Sentence-Segment Word Error test, using NIST scilite, of the baseline system result and the new system result; the \(p\)-value was < 0.001. In short, the accuracy (\(\text{Acc} = 55.4\%\)) of the new system significantly differ from the accuracy of the baseline system (\(\text{Acc} = 33.7\%\)). The implementation of CIP into the HVS model contributes to significantly better results.

6. Conclusion

This paper has presented the implementation of the HVS model in GGTK. The HVS model was introduced as structure discrimination of the flat-concept model. Our experiments provided evidence that CIP significantly betters results in our GGTK HVS model. We increased the accuracy of the baseline system from 33.7% to 55.4%.

In future work, we will integrate our model with a speech recognizer into one pass decoder. Our model is implemented as a dynamic Bayes network. As a result, we are able easily to transform our model to FSM. We will use the resulting FSM.
semantic model in the speech recognizer AT&T DRECOG\textsuperscript{TM}, [16]. This integration will allow us not only to create one pass decoder but also to generate semantic lattices or semantic N-best lists.

7. Acknowledgment

This work was supported by the Ministry of Education of the Czech Republic under project 1M0567. The authors would also like to thank to Luděk Müller for useful comments.

8. References


