Input Parameterization of the HVS Semantic Parser

Jan Švec[1], Filip Jurčíček[1], and Luděk Müller[2]

[1] Center of Applied Cybernetics, University of West Bohemia,
Pilsen, 306 14, Czech Republic
{honzas,filip}@kky.zcu.cz

[2] Department of Cybernetics, University of West Bohemia,
Pilsen, 306 14, Czech Republic
muller@kky.zcu.cz

Abstract. The aim of this paper is to present an extension of the hidden vector state semantic parser. First, we describe the statistical semantic parsing and its decomposition into the semantic and the lexical model. Subsequently, we present the original hidden vector state parser. Then, we modify its lexical model so that it supports the use of the input sequence of feature vectors instead of the sequence of words. We compose the feature vector from the automatically generated linguistic features (lemma form and morphological tag of the original word). We also examine the effect of including the original word into the feature vector. Finally, we evaluate the modified semantic parser on the Czech Human-Human train timetable corpus. We found that the performance of the semantic parser improved significantly compared with the baseline hidden vector state parser.

1 Introduction

This article concerns statistical semantic parsing which we interpret as a search process of the sequence of semantic tags $C = (c_1, c_2, \ldots, c_t)$ that maximizes the a posteriori probability $P(C|W)$ given the input sequence of words $W = (w_1, w_2, \ldots, w_t)$. This process can be described as:

$$C^* = \arg\max_C P(C|W) = \arg\max_C P(W|C)P(C) \tag{1}$$

where the probability $P(C)$ is called the semantic model and the probability $P(W|C)$ is called the lexical model. The simplest implementation of this process is Pieraccini's semantic finite state tagger (FST) which was used for the ATIS task [1]. The main disadvantage of this tagger is its inability to capture either long distance dependencies or a hierarchical structure of the processed utterance.

Several attempts has been proposed to overcome these disadvantages. In [2] He and Young described a HMM based parser with a hidden state variable implemented as a stack and where the state transitions are modeled using pushdown operations on this stack. The stack (called also vector state) stores the information about the hierarchical structure of a processed sequence of words. They called their parser the hidden vector state (HVS) parser [2].

The HVS parser is able to decode the hierarchical structure of the input sequence because it approximates the pushdown automaton. To estimate the parser parameters one needs a training data set provided with a structured semantic annotation. This structured semantic annotation forms a semantic tree. The nodes of the semantic tree are labeled with semantic concepts, which are considered to be basic units of particular meaning. The annotation must define an ordering relation between the nodes of the semantic tree. The order of nodes must correspond to the order of words in the underlying sentence. As a result, any semantic tree can be expressed as a sequence of vectors containing semantic concepts.

To alleviate the annotation stage the training data could be only weakly annotated by so-called abstract semantic trees. The abstract semantic tree does not provide the explicit word alignment between the nodes of the tree and the words of the sentence. Due to this merit the abstract annotation represents a robust annotation scheme to the intent to obtain a high inter-annotator agreement. We write the abstract semantic tree in the parenthesized form, which express both the parent/child relationships and the ordering of nodes in the semantic tree. For example, the sentence *Jede nějaký spěšný vlak do Prahy kolem čtvrté odpoledne?*[3] could be annotated with the semantic tree showed in Fig. 1. The corresponding abstract semantic tree is DEPARTURE(TO(STATION), TIME). Both the semantic and the abstract semantic trees can be converted into a sequence of vectors as it is shown at the bottom of the figure.

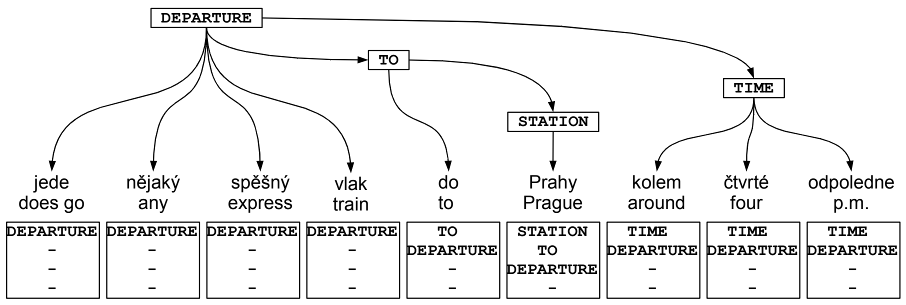

Fig. 1. An example of a full semantic parse tree with the corresponding stack sequence.

Later in this paper, we present an extension of the original HVS parser which we call the *input parameterization of the HVS parser.* The input of the original HVS parser is a pure sequence of words W. However, if the parser has some

[3] Lit. translation: *Does any express train go to Prague around four p.m.?*

additional information, it can take advantage of it and use a sequence of feature vectors $S = (s_1, s_2, \dots, s_T)$ where for every input word w_t one feature vector $s_t = (s_t[1], \dots, s_t[N])$ is computed. For example, the feature vector can be composed of some prosodic and linguistic features computed for the word w_t. In our experiments, we used the combination of the input word, its lemma, and morphological tag generated by an automatic morphology analyzer.

This article is organized in the following manner: in Section 2 we shortly describe the original HVS parser. In Section 3 we propose a novel method for the input parameterization of the HVS parser. Section 4 provides experimental results and finally, Section 5 concludes this paper.

2 The HVS parser

The HVS parser is an approximation of a pushdown automaton. This is mainly due to the limited stack depth and a reduced set of allowed stack operations. In other words, the HVS parser is the generalization of the finite state tagger. The HVS parser has a larger state space in comparison with Pieraccini's parser and state transitions are modeled using pushdown operations. The HVS parser is able to better capture the hierarchical structure typical of natural language.

The original HVS parser proposed by He and Young uses two stack operations: popping zero to four concepts off the stack and pushing exactly one new concept onto the stack. These operations are implemented in the semantic model which is given by:

$$P(C) = \prod_{t=1}^{T} P(pop_t \mid c_{t-1}) P(c_t[1] \mid c_t[2], c_t[3], c_t[4]) \tag{2}$$

where the hidden variable pop_t is the stack shift operation and takes values in the range $0, \dots, 4$ and the hidden variable $c_t = (c_t[1], \dots, c_t[4])$ is the vector state (the stack) of the HVS model. The depth of the stack is chosen to be the maximal depth of semantic trees found in the training data. We observed that the stack of at most four concepts is quite sufficient in all experiments described in Section 4. The concept $c_t[1]$ is the preterminal concept of the word w_t and the concept $c_t[4]$ is the root of semantic tree. The value of pop_t represents the count of concepts to be popped off the stack at time t. The value $pop_t = 0$ means that no concept is popped off the stack so the stack c_t grows by one new concept $c_t[1]$. Values greater than zero lead to popping pop_t concepts off the stack and to pushing one new concept $c_t[1]$ onto the stack.

The lexical model imposes an additional constraint on the stack sequence by allowing only such stack sequences that correspond to the input word sequence. The original HVS parser can process one word w_t at time t. The lexical model is given by:

$$P(W|C) = \prod_{t=1}^{T} P(w_t | c_t[1, \dots 4]) \tag{3}$$

Starting with the definition of the HVS parser, we extend the lexical model so that it accepts a sequence of feature vectors (parameterized input) on its input.

3 Input parameterization

The input parameterization extends the HVS parser into a more general HVS parser with the input feature vector (HVS-IFV parser). This parser works on a sequence of feature vectors $S = (s_1, \ldots, s_T)$ instead of a sequence of words W. The feature vector is defined as $s_t = (s_t[1], s_t[2], \ldots, s_t[N])$. To every word w_t, we assign the fixed number N of features. If we use the feature vector s_t instead of the word w_t in Eq. 3, the lexical model changes as follows:

$$\begin{aligned} P(S|C) &= \prod_{t=1}^{T} P(s_t \mid c_t) \\ &= \prod_{t=1}^{T} P(s_t[1], s_t[2], \ldots s_t[N] \mid c_t) \end{aligned} \tag{4}$$

Using the chain rule, we can rewrite this equation into the form:

$$P(S|C) = \prod_{t=1}^{T} \prod_{i=1}^{N} P(s_t[i] \mid s_t[1, \ldots i-1], c_t) \tag{5}$$

To minimize the data sparsity problem, we used the conditional independence assumption between the features $s_t[i]$ and $s_t[j]$, $i \neq j$ given the concept c_t. This kind of assumption is also used for example in the naive Bayes classifier to avoid the curse of dimensionality problem. The lexical model of the HVS-IFV parser is then given by:

$$P(S|C) = \prod_{t=1}^{T} \prod_{i=1}^{N} P(s_t[i] \mid c_t) \tag{6}$$

Because the conditional independence assumption is hardly expected to be always true, we need to modify the search process defined in Section 1. Let's assume that for example we have the sequence of the feature vectors $S = (s_t[1], s_t[2])_{t=1}^{T}$ where $s_t[1]$ is equal to $s_t[2]$ for every t. The search process is given by:

$$\begin{aligned} C^* &= \arg\max_C \prod_{t=1}^{T} \left[\prod_{i=1}^{2} P(s_t[i] \mid c_t) \right] P(pop_t \mid c_{t-1}) P(c_t[1] \mid c_t[2], c_t[3], c_t[4]) \\ &= \arg\max_C \prod_{t=1}^{T} \left[P(s_t[1] \mid c_t) \right]^2 P(pop_t \mid c_{t-1}) P(c_t[1] \mid c_t[2], c_t[3], c_t[4]) \end{aligned}$$

As we can see, the lexical model probability $P(S|C)$ is exponentially scaled with the factor 2 and it causes imbalance between the lexical and the semantic model depending on the dimension N of the feature vector. Therefore, we use the scaling factor λ to compensate the imbalance caused by the use of feature vectors:

$$C^* = \arg\max_C P(S|C)P^{\lambda}(C) \tag{7}$$

The HVS-IFV parser is defined by equations 2, 6, and 7. To find the optimal value of λ, we use a grid search over the finite set of values to maximize the concept accuracy measure defined in Section 4.2. The grid search is performed on the development data.

4 Experiments

The semantic parsers evaluated in this article were trained and tested on the Czech human-human train timetable (HHTT) dialogue corpus [3]. The HHTT corpus consists of 1,109 dialogues completely annotated with the abstract semantic annotations. Both operators and users have been annotated. The corpus comprises 17,900 utterances in total. The vocabulary size of the whole corpus is 2,872 words. There are 35 semantic concepts in the HHTT corpus. The dialogues were divided into training data (798 dialogues - 12,972 dialogue acts, 72%), development data (88 dialogues - 1,418 dialogue acts, 8%), and test data (223 dialogues - 3,510 dialogue acts, 20%). The development data were used for finding the optimal value of the semantic model scaling factor.

The training of the HVS parser is divided into three parts: 1) initialization of the semantic and lexical models, 2) estimation of parameters of the semantic and lexical models, 3) smoothing the probabilities of the semantic and lexical models. All probabilities are initialized uniformly. To estimate the parameters of the semantic and the lexical model, it is necessary to use the expectation-maximization (EM) algorithm because abstract semantic annotations does not provide fully annotated parse trees. There are no explicit word level annotations. After training the parameters we smooth all three probabilities using the back-off model.

We evaluated our experiments using two measures: semantic accuracy ($SAcc$) and concept accuracy ($CAcc$). We could not use the PARSEVAL measures [4] because they rely on availability of full parse trees of the test data. As we already mentioned above, the HHTT corpus has no explicit word level annotation.

4.1 Semantics accuracy

Two semantic annotations are considered equal only if they exactly match each other. Exact match is very tough standard because under the exact match the difference between semantics ARRIVAL(TIME, FROM(STATION)) and ARRIVAL(TIME, TO(STATION)) is equal to the difference between semantics ARRIVAL(TIME, FROM(STATION)) and DEPARTURE(TRAIN_TYPE). The semantic accuracy of a hypothesis is defined as

$$SAcc = \frac{E}{N} \cdot 100\% \tag{8}$$

where N is the number of evaluated semantic annotations and E is the number of hypothesis semantic annotations which exactly match the reference.

4.2 Concept accuracy

Similarity scores between the reference and the hypothesis semantics can be computed also by the tree edit distance algorithm [5]. The tree edit distance measures the similarity between two trees by comparing subtrees of both the reference and the hypothesis annotations.

The tree edit distance algorithm computes the minimum number of substitutions, deletions, and insertions required to transform one tree into another one and uses the dynamic programing. The operations act on the tree nodes and modify the tree by changing the parent/child relationships of the tree. The tree edit distance is convenient for measuring similarity between two abstract semantic annotations because it does not rely on the alignment of the annotation and the underlying word sequence. We define the concept accuracy as

$$CAcc = \frac{N - S - D - I}{N} \cdot 100\% \tag{9}$$

where N is the number of concepts in the reference semantic annotation and S, D, and I are the numbers of substitutions, deletions, and insertions, respectively, in the minimum-cost alignment of the reference and the hypothesis semantic annotation.

4.3 Morphological analysis

Every utterance of the HHTT training corpus was automatically processed using the morphological analyser and the tagger from Prague Dependency Treebank (PDT) [6]. The morphological analysis for every input word generated a lemma and a morphological tag which were then used as features in the parameterized input of the HVS parser. The morphological tag is represented as a string of several symbols, each for one morphological category: part-of-speech, detailed part-of-speech, gender, number, case, possessor's gender, possessor's number, person, tense, degree of comparison, negation, and voice. We have also done some experiments with reduced morphological tags (we removed some morphological categories from the morphological tag) but these experiments bring no improvement in concept accuracy. Because Czech has very rich inflection, the lemmatization of input sentence reduced the vocabulary. The vocabulary size for every input feature is shown in Table 1.

4.4 Results

Table 1 shows the performance of the original HVS parser described in Section 2 with words, lemmas, and morphological tags on its input. The original HVS parser with words on its input was chosen as the baseline. To measure the statistical significance, we used the paired t-test (NIST MAPSSWE test) and the difference was taken as significant if p-value of this test was < 0.01.

We combined these features and passed their combination to the input of HVS-IFV parser described in Section 3. The performance on the development data set is shown in Table 2. We can see, that the HVS-IFV parser whose feature vector contains besides words also their lemmas yields better performance. Table 3 reports the performance comparison of the original HVS parser and the best HVS-IFV parser evaluated both on the development and the test data sets.

Table 1. *The performance and the vocabulary size of the HVS parser with different inputs evaluated on the development data.*

Input feature	*Vocab. size*	*CAcc*	*SAcc*	*p-value*
words (baseline)	696	67.0	52.8	
lemmas	551	68.3	53.4	< 0.01
morph. tags	225	42.2	30.5	< 0.01

Table 2. *The performance of the HVS-IFV parser with different inputs evaluated on the development data.*

Input features	*SAcc*	*CAcc*	*p-value*
words, m. tags	69.3	55.3	< 0.01
lemmas, m. tags	70.3	56.2	< 0.01
words, lemmas	73.1	58.2	< 0.01
words, lemmas, m. tags	72.2	58.1	< 0.01

Table 3. *The performance of the baseline HVS and developed HVS-IFV parsers. The parsers were evaluated on the test and the development data.*

	Test data			Development data		
Parser Type	*CAcc*	*SAcc*	*p-value*	*CAcc*	*SAcc*	*p-value*
HVS (baseline)	64.9	50.4		67.0	52.8	
HVS-IFV (words, lemmas)	69.4	57.0	< 0.01	73.1	58.2	< 0.01

5 Conclusion

In this work, we presented a modification of the HVS parser. The proposed HVS-IFV parser is able to parse the sequence of feature vectors. We started with the original lexical model of the HVS parser and we used the assumption of conditional independence of the input features to simplify the parser's lexical model. We showed that the conditional independence assumption affects the search process. We used the exponential scaling of the semantic model probability to correct the imbalance between the lexical and the semantic model of the HVS-IFV parser.

We processed the training corpus with the automatic morphology analyzer and tagger. We used original words, their lemmas, and morphological tags as the input features of the HVS-IFV parser. The best performance was achieved with the feature vector composed of words and lemmas. This approach significantly increases the performance of the semantic parser. We believe that the improvements come mainly from the ability of lemmas to cluster the original word vocabulary into classes with the same meaning and in this way make the model more robust. However, we found that in some cases it is still useful to include the words into the feature vector because the words help to distinguish the ambiguous cases where two words with the different meaning has the same lemma. In total, $SAcc$ was significantly increased from 50.4% to 57.0% and $CAcc$ from 64.9% to 69.4% measured on the test data.

6 Acknowledgments

This work was supported by the Ministry of Education of the Czech Republic under project No. 1M0567 (CAK).

References

1. Hemphill, C.T., Godfrey, J.J., Doddington, G.R.: The ATIS spoken language systems pilot corpus. In: Proceedings of DARPA Speech and Natural Language Workshop, Hidden Valley, PA, USA (1990) 96–101
2. He, Y., Young, S.: Hidden vector state model for hierarchical semantic parsing. In: Proceedings of ICASSP, Hong Kong (2003)
3. Jurčíček, F., Zahradil, J., Jelinek, L.: A Human-Human Train Timetable Dialogue Corpus. In: Proceedings of EUROSPEECH, Lisboa, Portugal (2005)
4. Black, E., Abney, S., Flickinger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M., Roukos, S., Strzalkowski, S.T.: A procedure for quantitatively comparing the syntactic coverage of english grammars. In: Proceedings of the 1990 DARPA Speech and Natural Language Workshop, Pacific Grove, CA (1991) 306–311
5. Klein, P.: Computing the edit-distance between unrooted ordered trees. In Bilardi, G., Italiano, G.F., Pietracaprina, A., eds.: Proceedings of the6 th Annual European Symposium. Number 1461, Venice, Italy, Springer-Verlag, Berlin (1998) 91–102
6. Hajič, J.: Disambiguation of Rich Inflection (Computational Morphology of Czech). Karolinum Press, Charles University -in prep., Prague (2001)