Learning Parse Decisions From Examples With Rich Context
as submitted to ACL'96 on January 8, 1996

Ulf Hermjakob and Raymond J. Mooney
Dept. of Computer Sciences
University of Texas at Austin
Austin, TX 78712, USA
ulf@cs.utexas.edu  mooney@cs.utexas.edu

Abstract

We present a knowledge and context-based system for parsing natural language and evaluate it on sentences from the Wall Street Journal. Applying machine learning techniques, the system uses parse action examples acquired under supervision to generate a deterministic shift-reduce parser in the form of a decision structure. It relies heavily on context, as encoded in features which describe the morphological, syntactical, semantical and other aspects of a given parse state.

1. INTRODUCTION

The parsing of unrestricted text, with its enormous word and structural ambiguity, still poses a great challenge in natural language processing. While systems that attempted to capture parse grammars in absolute hand-coded rules just could not penetrate the complexity of ambiguity of unrestricted text, probabilistic approaches with often only very limited context sensitivity have hit a performance ceiling even when trained on very large corpora.

To cope with the complexity of unrestricted text, parse rules of any kind of formalism will have to consider a complex context with many different features of e.g. morphological, syntactic or semantic type. This can present a significant problem, because even linguistically trained natural language developers have great difficulties writing and even more so extending explicit parse grammars covering a wide range of natural language. On the other hand it is much easier for humans to decide how specific sentences should be analyzed, although for some tricky sentences even this can still require substantial linguistic expertise.

We therefore propose an approach to parsing based on example-based learning with a very strong emphasis on context, integrating morphological, syntactical, semantical and other aspects relevant to making good parse decisions, thereby also allowing the parsing to be deterministic. Applying machine learning techniques, the system uses parse action examples acquired under supervision to generate a deterministic shift-reduce type parser in the form of a decision structure. The generated parser transforms input sentences into a phrase-structure and case-frame tree, powerful enough to be fed into a transfer and a generation module to complete the full process of machine translation.

Relieving the NL-developer from the hard if not impossible task of writing an explicit grammar, the focus on relevant features at the same time requires a relatively modest number of training examples when compared to more statistically oriented approaches. The approach we take provides a high degree of manageability of the ‘grammar’ when increasing its coverage.
2. BASIC PARSING PARADIGM

As the basic mechanism for parsing text into a shallow semantic representation, we choose a shift-reduce type parser (Marcus, 1980). It breaks parsing into an ordered sequence of small and manageable parse actions such as shift and reduce. This ordered "left-to-right" parsing is much closer to how humans parse a sentence than e.g. chart oriented parsers; it allows a very transparent control structure and makes the parsing process relative intuitive for humans. This is very important, because during the training phase, the system is guided by a human supervisor for whom the flow of control needs to be as transparent and intuitive as possible.

The parsing does not have separate phases for part-of-speech selection and syntax and semantic processing, but rather integrates all of them into a single parsing phase. Since the system has all morphological, syntactical and semantical context information available at all times, the system can make well-based decisions very early, allowing a single path, i.e. deterministic parse, which eliminates wasting computation on 'dead end' alternatives.

Before the parsing itself starts, the input string is segmented into a list of words including punctuation marks, which then are sent through a morphological analyzer that, using a lexicon, produces primitive frames for the segmented words. There is one primitive frame for each word and part of speech that the word can be, so that several primitive frames for one word would reflect ambiguity of the word with respect to part of speech. (Morphological ambiguity is captured within a frame.)

![Diagram](image)

Figure 1: A typical parse action (simplified); boxes represent frames

The central data structure for the parser contains a parse stack and an input list. The parse stack and the input list contain trees of frames of words or phrases. Core slots of frames are surface and lexical form, syntactic and semantic category, subframes with syntactic and semantic roles, and form restrictions such as number, person, and tense. Other slots can include information like the numerical
value of number words, flags whether or not a (German) verb has a separable or inseparable prefix etc.

Initially, the parse stack is empty and the input list contains the primitive frames produced by the morphological analyzer. After initialization, the deterministic parser applies a sequence of parse actions to the parse structure. The most frequent parse actions are shift, which shifts a frame from the input list onto the parse stack or backwards, and reduce, which combines one or several frames on the parse stack into one new frame. The frames to be combined are typically, but not necessarily, next to each other at the top of the stack. As shown in figure 1, the action

\[(R \text{ 2 to VP as PRED (OBJ PAT))}\]

for example reduces the two top frames of the stack into a new frame that is marked as a verb phrase and contains the next-to-the-top frame as its predicate (or head) and the top frame of the stack as its object and patient. Other parse actions include ‘add-into’ which adds frames arbitrarily deep into an existing frame tree, ‘mark’ that marks some slot of some frame with some value and operations to introduce empty categories (traces and ‘PRO’, as in “She wanted PRO, to win.”). Parse actions can have numerous arguments, making the parse action language very powerful.

The parse action examples needed for training the system are acquired interactively. For each training sentence, the system and the supervisor parse the sentence step by step with the supervisor entering the next parse action command, e.g. \[(R \text{ 2 to VP as PRED (OBJ PAT))}\], and the system executing it, repeating this sequence until the sentence is fully parsed. At least for the very first sentence, the supervisor actually has to type in the full parse action commands. With a growing number of parse action examples available, the system, as described below in more detail, can be trained on those previous examples. In such a partially trained system, the parse actions are then proposed by the system using a parse decision structure which “classifies” the current context. The proper classification is the specific action or sequence of actions that (the system believes) should be performed next. During further training, the supervisor then enters parse action commands by either confirming what the system proposes or overruling it by providing the proper action. As the corpus of parse example grows and the system gets trained on more and more data, the system becomes more refined, so that the supervisor has to overrule the system with decreasing frequency. The sequence of correct parse actions for a sentence is then recorded in a log file.

3. FEATURES

To make good parse decisions, a wide range of features at various degrees of abstraction have to be considered. To express such a wide range of features, we defined a feature language. Given a particular parse state and a feature, the system can interpret the feature and compute its value for the given parse state, often using additional knowledge resources such as

1. a general knowledge base (KB), which currently consists of a directed acyclic graph of concepts, with currently 3608 is-a-relationship links,
   e.g., “CAR NOUN-CONCEPT is-a VEHICLE NOUN-CONCEPT”, and

2. subcategorization tables that describe the syntactic and semantic role structure(s) for verbs and nouns with currently close to a total of 200 entries.

The following examples, for easier understanding rendered in English and not in feature language syntax, illustrate the expressiveness of the feature language:

- the general syntactic class of the top element of the stack (e.g. adjective, noun phrase),
• the specific finite tense of the second stack element (e.g. present tense, past tense),
• whether or not some element could be a nominal degree adverb,
• whether or not some phrase already contains a subject,
• the semantic role of some noun phrase with respect to some verb phrase (e.g. agent, time; this involves pattern matching with corresponding entries in the verb subcategorization table),
• whether or not some noun and verb phrase agree.

Features can in principal refer to any element on the parse stack or input list, and any of their subelements, at any depth. Since all of the currently 151 features are supposed to bear some linguistic relevance, none of them actually refer to anything too far removed from the current focus of a parse state. The set of features is globally used for all examples and can easily be extended when the need arises.

4. LEARNING DECISION STRUCTURES

Traditional statistical techniques also use features, but often have to sharply limit their number (for some trigram approaches to three fairly simple features) to avoid overwhelming computational complexity.

In parsing, only a very small number of features are crucial over a wide range of examples, while most features are critical in only a few examples, being used to ‘fine-tune’ the decision structure for special cases. So in order to overcome the antagonism between the importance of having a large number of features and the need to control the number of examples required for learning, particularly when acquiring examples under supervision, we choose a decision-tree-type learning algorithm, which recursively selects the most discriminating feature of the corresponding subset of training examples, eventually ignoring all locally irrelevant features, thereby tailoring the size of the final decision structure to the complexity of the training data.

Given a set of features, the system can feed parse actions from the log file through the parse engine, and automatically augment each parse action with its corresponding feature-value vector. While parse actions might be complex for the action interpreter, they are atomic with respect to the decision structure learner; e.g. “(R 2 TO VP AS PRED (OBJ PAT))” would be such an atomic classification. A set of examples consisting of a feature-value vector and a parse action as a classification is then fed into an ID3-based learning routine that generates a decision structure, which can then ‘classify’ any given parse state (which implicitly assigns a values to every feature) by proposing what parse action to perform next. As an extension to the standard ID3 model (Quinlan, 1986), our system uses a flexible decision structure that currently consists of a decision list of hierarchical decision trees.

5. WALL STREET JOURNAL EXPERIMENTS

We now present intermediate results on training and testing a prototype implementation of the system with sentences from the Wall Street Journal, a prominent corpus of ‘real’ text, as collected on the ACL-CD.

In order to limit the size of the required lexicon, we work on a reduced corpus that includes all those sentences that are fully covered by the 3000 most frequently occurring words (ignoring numbers etc.) in the entire corpus. The first 144 sentences used in this experiment vary in length from 4 to 45 words, averaging at 17.0 words. One of these sentence is “Canadian manufacturers’ new orders fell
to $20.80 billion (Canadian) in January, down 4% from December’s $21.67 billion on a seasonally adjusted basis, Statistics Canada, a federal agency, said.”.

For the following test series, the corpus of 144 sentences that currently have parse action logs associated with them is divided into 9 blocks of 16 sentences. Each of these 9 blocks is then consecutively used for testing. For each of the 9 sub-tests, a varying number of sentences from the other blocks is used for training the parse decision structure, so that within a sub-test, none of the training sentences are ever used as a test sentence. The results of the 9 sub-tests of each series are then averaged.

<table>
<thead>
<tr>
<th>Number of training sentences</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>84.6%</td>
<td>84.6%</td>
<td>87.3%</td>
<td>89.9%</td>
</tr>
<tr>
<td>Recall</td>
<td>80.4%</td>
<td>83.0%</td>
<td>86.3%</td>
<td>89.4%</td>
</tr>
<tr>
<td>Labelled precision</td>
<td>81.0%</td>
<td>81.6%</td>
<td>85.1%</td>
<td>88.3%</td>
</tr>
<tr>
<td>Labelled recall</td>
<td>77.4%</td>
<td>80.0%</td>
<td>83.8%</td>
<td>87.6%</td>
</tr>
<tr>
<td>Tagging accuracy</td>
<td>97.2%</td>
<td>97.3%</td>
<td>97.5%</td>
<td>97.9%</td>
</tr>
<tr>
<td>Crossings per sentence</td>
<td>2.6</td>
<td>2.5</td>
<td>2.0</td>
<td>1.8</td>
</tr>
<tr>
<td>Sent. with 0 crossings</td>
<td>27.8%</td>
<td>30.6%</td>
<td>39.6%</td>
<td>42.4%</td>
</tr>
<tr>
<td>Sent. with up to 1 crossing</td>
<td>46.5%</td>
<td>48.6%</td>
<td>58.3%</td>
<td>61.8%</td>
</tr>
<tr>
<td>Sent. with up to 2 crossings</td>
<td>59.7%</td>
<td>64.4%</td>
<td>69.4%</td>
<td>76.4%</td>
</tr>
<tr>
<td>Sent. with up to 3 crossings</td>
<td>73.6%</td>
<td>75.6%</td>
<td>81.3%</td>
<td>84.7%</td>
</tr>
<tr>
<td>Sent. with up to 4 crossings</td>
<td>79.2%</td>
<td>81.3%</td>
<td>88.2%</td>
<td>87.5%</td>
</tr>
<tr>
<td>Correct operations</td>
<td>78.8%</td>
<td>82.4%</td>
<td>86.1%</td>
<td>89.3%</td>
</tr>
<tr>
<td>Sent. with correct OpSequence</td>
<td>2</td>
<td>5</td>
<td>7</td>
<td>19</td>
</tr>
<tr>
<td>Sent. with correct Struct&amp;Label</td>
<td>5</td>
<td>8</td>
<td>18</td>
<td>28</td>
</tr>
<tr>
<td>Sentences with endless loop</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: Evaluation results using a total of 144 test sentences for each of the four test series

**Precision:** \( \frac{\text{number of correct constituents in system parse}}{\text{number of correct constituents in system parse}} \)

**Recall:** \( \frac{\text{number of correct constituents in system parse}}{\text{number of correct constituents in logged parse}} \)

**Crossing brackets:** number of constituents which violate constituent boundaries with a constituent in the logged parse.

**Labelled precision/recall** measures not only structural correctness, but also the correctness of the syntactic label. **Correct operations** measures the number of correct operations during a parse continuously corrected based on the logged sequence. A sentence has a correct operating sequence, **OpSequence**, if the system fully predicts the logged parse action sequence, and a correct structure and labeling, **Struct&Label**, if the structure and syntactic labeling of the final system parse of a sentence is 100% correct, regardless of the operations leading to it.

The current set of 151 features was sufficient to always discriminate examples with different parse actions, thereby producing 100% test-accuracy on sentences that the system had previously been trained on. While that percentage is certainly less important than the accuracy figures for unseen sentences, it nevertheless represents an upper ceiling, which for many statistical systems lies significantly below 100%.

Many of the mistakes are due to encountering constructions that just have not been seen before at all, typically causing several erroneous parse decisions in a row. This observation also supports our hope that with more training sentences, the testing accuracy for unseen sentences will still rise significantly.
6. RELATED WORK

Our basic parsing and interactive training paradigm is based on (Simmons and Yu, 1992). We have extended their work by significantly increasing the expressiveness of the parse action and feature languages, in particular by moving far beyond a few simple features, limited to syntax only, by adding several knowledge resources and by introducing a sophisticated machine learning component.

(Magerman, 1995) uses a decision tree model similar to ours, training his system SPATTER with parse action sequences for 40,000 Wall Street Journal sentences derived from the Penn Treebank. Questioning the traditional n-grams, Magerman already advocates a heavier reliance on contextual information. Going beyond Magerman's still relatively rigid set of 36 features, we propose a yet richer, basically unlimited feature language and set. Our parse action sequences are too complex to be derived from a treebank like Penn's. While this necessitates the involvement of a supervisor for training, we are able to perform deterministic parsing and get already very good test results for only 128 training sentences.

7. CONCLUSION

We try to bridge the gap between the typically hard-to-scale, context-rich rule approach and the typically large-scale, context-poor probabilistic approach for unrestricted text parsing. Using rich and unified context and a complex parse action language, we can demonstrate good intermediate results for single-pass deterministic parsing.

While many limited-context probabilistic approaches have already reached a performance ceiling, we still expect to significantly improve our results when increasing our training base beyond the currently 128 sentences. Even then the training size will compare favorably with the huge number of training sentences necessary for many probabilistic systems.

REFERENCES


