Parsin Syntactic and Semantic Dependencies with Two Single-Stage Maximum Entropy Models

Hai Zhao and Chunyu Kit
Department of Chinese, Translation and Linguistics
City University of Hong Kong
83 Tat Chee Avenue, Kowloon, Hong Kong, China
haizhao@cityu.edu.hk, ctckit@cityu.edu.hk

Abstract
This paper describes our system to carry out the joint parsing of syntactic and semantic dependencies for our participation in the shared task of CoNLL-2008. We illustrate that both syntactic parsing and semantic parsing can be transformed into a word-pair classification problem and implemented as a single-stage system with the aid of maximum entropy modeling. Our system ranks the fourth in the closed track for the task with the following performance on the WSJ+Brown test set: 81.44% labeled macro F1 for the overall task, 86.66% labeled attachment for syntactic dependencies, and 76.16% labeled F1 for semantic dependencies.

1 Introduction
The joint parsing of syntactic and semantic dependencies introduced by the shared task of CoNLL-2008 is more complicated than syntactic dependency parsing or semantic role labeling alone (Surdeanu et al., 2008). For semantic parsing, in particular, a dependency-based representation is given but the predicates involved are unknown, and we also have nominal predicates besides the verbal ones. All these bring about more difficulties for learning. This paper presents our research for participation in the CoNLL-2008 shared task, with a highlight on our strategy to select learning framework and features for maximum entropy learning.

2 System Description
For the sake of efficiency, we opt for the maximum entropy model with Gaussian prior as our learning model for both the syntactic and semantic dependency parsing. Our implementation of the model adopts L-BFGS algorithm for parameter optimization as usual (Liu and Nocedal, 1989). No additional feature selection techniques are applied.

Our system consists of three components to deal with syntactic and semantic dependency parsing and word sense determination, respectively. Both parsing is formulated as a single-stage word-pair classification problem, and the latter is carried out by a search through the NomBank (Meyers et al., 2004) or the PropBank (Palmer et al., 2005). These two dictionaries that we used are downloaded from CoNLL-2008 official website.

2.1 Syntactic Dependency Parsing
We use a shift-reduce scheme to implement syntactic dependency parsing as in (Nivre, 2003). It takes a step-wised, history- or transition-based approach. It is basically a word-by-word method with a projective constraint. In each step, the classifier checks a word pair, e.g., TOP, the top of a stack for processed words, and, NEXT, the first word in the unprocessed word sequence, in order to determine if a dependent label should be assigned to them. Besides two arc-building actions, a shift action and a reduce action are also defined to meet the projective constraint, as follows.

This study is supported by CERG grant 9040861 (CityU 1318/03H) and CityU Strategic Research Grant 7002037.

1These two dictionaries that we used are downloaded from CoNLL-2008 official website.
and pop the stack.

cannot eliminate all non-projective dependencies.

only if the above standard projectivization step

deprojectivizing the output during decoding. Sec-

tional dependent label encoding for the purpose of

replace the head of a non-projective dependency

projective. Thus, we use a simplified strategy to

corpus. The former counts a given split PoS occur-

We implement a left-to-right arc-eager parsing

model in a way that the parser scan through an in-

sequence, etc.

dependency. In practice, the above two-

original head will be chosen as the head of a non-

in a sequence, then the word with the shortest se-

and pop the stack.

(2) **Right-arc**: Add an arc from **TOP** to **NEXT**

and push **NEXT** onto the stack.

(3) **Reduce**: Pop **TOP** from the stack.

(4) **Shift**: Push **NEXT** onto the stack.

We implement a left-to-right arc-eager parsing

model in a way that the parser scan through an in-

sequence from left to right and the right de-

pends are attached to their heads as soon as possible (Hall et al., 2007). To construct a single-

stage system, we extend the left-/right-arc actions

to their correspondent multi-label actions as neces-

Table 1: Feature Notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>Clique in the top of stack</td>
</tr>
<tr>
<td>s−1,...</td>
<td>The first clique below the top of stack, etc.</td>
</tr>
<tr>
<td>i,...,i+1</td>
<td>The first (second) clique in the unprocessed sequence, etc.</td>
</tr>
<tr>
<td>dprel</td>
<td>Dependent label</td>
</tr>
<tr>
<td>h</td>
<td>Head</td>
</tr>
<tr>
<td>lns</td>
<td>Leftmost child</td>
</tr>
<tr>
<td>rns</td>
<td>Rightmost child</td>
</tr>
<tr>
<td>rmns</td>
<td>Right nearest child</td>
</tr>
<tr>
<td>form</td>
<td>Word form</td>
</tr>
<tr>
<td>lemma</td>
<td>Word lemma</td>
</tr>
<tr>
<td>pos</td>
<td>Predicted PoS tag</td>
</tr>
<tr>
<td>sp,Y</td>
<td>Split Y, which may be form, lemma or pos.</td>
</tr>
<tr>
<td>x</td>
<td>of the clique in the top of stack</td>
</tr>
<tr>
<td>l</td>
<td>Feature combination, i.e. <code>s.pos/i.pos</code></td>
</tr>
<tr>
<td>p</td>
<td>The current predicate candidate</td>
</tr>
<tr>
<td>a</td>
<td>The current argument candidate</td>
</tr>
</tbody>
</table>

Table 2: Features for Syntactic Parsing

<table>
<thead>
<tr>
<th>Basic Feature</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>x.sp,Y</td>
<td>itself, its previous two and next two Y’s, and all bigrams within the five-clique window, ((x is s or i, and Y is form, lemma or pos.))</td>
</tr>
<tr>
<td>x,Y</td>
<td>((x is s or i, and Y is form, lemma or pos.))</td>
</tr>
<tr>
<td>x.Y/i,Y</td>
<td>((x is s or i, and Y is pos, sp.lemma or sp.pos))</td>
</tr>
</tbody>
</table>

}\(x, y, z, i, j, k,\) | Y, \(s, \) \(i, \) \(j, \) \(k, \) \(\) | Y, \(s, \) \(i, \) \(j, \) \(k, \) \(\) | Y, \(s, \) \(i, \) \(j, \) \(k, \) \(\) | Y, \(s, \) \(i, \) \(j, \) \(k, \) \(\) | Y, \(s, \) \(i, \) \(j, \) \(k, \) \(\) |
| s.k.sp.form   | - | - | - | - | - |
| s.dprel      | - | - | - | - | - |
| s.lm.dprel   | - | - | - | - | - |
| s.rn.dprel   | - | - | - | - | - |
| s.lm.sp.lemma| - | - | - | - | - |
| s.dprel      | - | - | - | - | - |
| s.lm.sp.pos  | - | - | - | - | - |
| s.rn.sp.pos  | - | - | - | - | - |
| x.sp.pos     | - | - | - | - | - |
| rootscore    | - | - | - | - | - |
| pairscore    | - | - | - | - | - |
| curroot      | - | - | - | - | - |

Table 2: Features for Syntactic Parsing

in a sequence, then the word with the shortest se-

sequence (rather than dependent tree) distance to the

original head will be chosen as the head of a non-

projective dependency. In practice, the above two-

step projectivization procedure can eliminate all

non-projective dependencies in all sequences. Our

purpose here is to provide as much data as possible

for training, and only projective sequences are

input for training and output for decoding.

While memory-based and margin-based learn-

ing approaches such as support vector machines

are popularly applied to shift-reduce parsing, our

work provides evidence that the maximum en-

ropy model can achieve a comparative perfor-

mance with the aid of a suitable feature set. With

feature notations in Table 1, we use a feature set

as shown in Table 2 for syntactic parsing.

Here, we explain ‘rootscore’, ‘pairscore’ and

curroot in Table 2. Both rootscore and pairscore

return the log frequency for an event in the training

corpus. The former counts a given split PoS occur-

ning as ROOT, and the latter two split PoS’s com-

bination associated with a dependency label. The

feature curroot returns the root of a partial parsing

tree that includes a specified node.

2.2 Semantic Dependency Parsing

Assuming no predicates overtly known, we keep

using a word-pair classifier to perform semantic

parsing through a single-stage processing. Specif-

ically, we specify the first word in a word pair as

a predicate candidate (i.e., a semantic head, and

noted as p in our feature representation) and the
next as an argument candidate (i.e., a semantic dependent, and noted as $a$). We do not differentiate between nominal and verbal predicates and our system handles them in in exactly the same way. If decoding outputs show that no arguments can be found for a predicate candidate in the decoding output, then this candidate will be naturally discarded from the output predicate list.

When no constraint available, however, all word pairs in the input sequence must be considered, leading to very poor efficiency in computation for no gain in effectiveness. Thus, the training sample needs to be pruned properly.

For predicate, only nouns and verbs are considered possible candidates. That is, all words without a split PoS in these two categories are filtered out. Many prepositions are also marked as predicate in the training corpus, but their arguments' roles are 'SU', which are not counted the official evaluation.

For argument, a dependency version of the pruning algorithm in (Xue and Palmer, 2004) is used to find, in an iterative way, the current syntactic head and its siblings in a parse tree in a constituent-based representation. In this representation, the head of a phrase governs all its sisters in the tree, as illustrated in the conversion of constituents to dependencies in (Lin, 1995). In our implementation, the following equivalent algorithm is applied to select argument candidates from a syntactic dependency parse tree.

Initialization: Set the given predicate candidate as the current node;

(1) The current node and all of its syntactic children are selected as argument candidates.

(2) Reset the current node to its syntactic head and repeat step (1) until the root is reached.

This algorithm can cover 98.5% arguments while reducing about 60% of the training samples, according to our statistics. However, this is achieved at the price of including a syntactic parse tree as part of the input for semantic parsing.

The feature set listed in Table 3 is adopted for our semantic parsing, some of which are borrowed from (Hacioglu, 2004). Among them, dpTreeRelation returns the relationship of $a$ and $p$ in a syntactic parse tree. Its possible values include parent, sibling, child, uncle, grand parent etc. Note that there is always a path to the ROOT in the syntactic parse tree for either $a$ or $p$. Along the common part of these two paths, SharedDprelPath returns the sequence of dependent labels collected from each node, and SharedPosPath returns the corresponding sequence of PoS tags. $x$.dprelPath and $x$.posPath return the PoS tag sequence from $x$ to the beginnings of SharedDprelPath and SharedPosPath, respectively. $a/p$.dprelPath returns the concatenation of $a$.dprelPath and $p$.dprelPath.

We may have an example to show how the feature bankAdvice works. Firstly, the current processed semantic role labels and argument candidate direction are checked. Specifically, they are

<table>
<thead>
<tr>
<th>Basic</th>
<th>Extension</th>
</tr>
</thead>
</table>
| $x.sp.Y$ | itself, its previous and next two clauses, and all bigrams within the three-clique window. ($Y$ is form or lemma)
| $x.sp.pos$ | itself, its previous and next two clauses, and all bigrams within the five-clique window.
| $x.Y$ | ($Y$ is form, lemma or pos.)
| $p.Y/i.Y$ | ($Y$ is sp.lemma or sp.pos.)

$a$ is the same as $p$

- $a.h.sp.form$
- $x.dprel$
- $x.lm.dprel$
- $p.rm.dprel$
- $p.lm.sp.pos$
- $a.lm.dprel/a.dprel$
- $a.lm.sp.pos/a.sp.pos$
- $a.sp/Y.a.dprel$ ($Y$ is lemma or pos.)
- $x.sp.lemma/x.h.sp.form$
- $p.sp.lemma/p.h.sp.pos$
- $p.sp.pos/p.dprel$
- $a.preddir$
- $p.voice/a.preddir$
- $x.posSeq$
- $x.dprelSeq$
- $a/dpTreeLevel$
- $a/p/dpRelation$
- $a/p/SharedPosPath$
- $a/p/SharedDprelPath$
- $a/p/x.posPath$
- $a/p/x.dprelPath$
- $a/p/dpRelPath$

*a* $x$ is $p$ or $a$ throughout the whole table.

*b* This and the following features are all concerned with a known syntactic dependency tree.

*preddir*: the direction to the current predicate candidate.

*voice*: if the syntactic head of $p$ is be and $p$ is not ended with -ing, then $p$ is passive.

*posSeq*: PoS tag sequence of all syntactic children

*dprelSeq*: syntactic dependent label sequence of all syntactic children

*dpTreeLevel*: the level in the syntactic parse tree, counted from the leaf node.

Table 3: Features for Semantic Parsing
Table 4: The Results of Syntactic Parsing (%)

<table>
<thead>
<tr>
<th>Data</th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development</td>
<td>88.78</td>
<td>85.85</td>
<td>91.14</td>
</tr>
<tr>
<td>WSJ</td>
<td>89.86</td>
<td>87.52</td>
<td>92.47</td>
</tr>
<tr>
<td>Brown</td>
<td>85.03</td>
<td>79.83</td>
<td>86.71</td>
</tr>
<tr>
<td>WSJ+Brown</td>
<td>89.32</td>
<td>86.66</td>
<td>91.83</td>
</tr>
</tbody>
</table>

Table 5: The Results of Semantic Parsing (%)

<table>
<thead>
<tr>
<th>Data</th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development</td>
<td>79.76</td>
<td>72.25</td>
<td>75.82</td>
</tr>
<tr>
<td>WSJ</td>
<td>80.57</td>
<td>74.97</td>
<td>77.67</td>
</tr>
<tr>
<td>Brown</td>
<td>66.28</td>
<td>61.29</td>
<td>63.69</td>
</tr>
<tr>
<td>WSJ+Brown</td>
<td>79.03</td>
<td>73.49</td>
<td>76.16</td>
</tr>
<tr>
<td>Label.</td>
<td>84.05</td>
<td>81.25</td>
<td>82.62</td>
</tr>
<tr>
<td>Macro</td>
<td>73.05</td>
<td>70.56</td>
<td>71.78</td>
</tr>
<tr>
<td>WSJ+Brown</td>
<td>82.85</td>
<td>80.08</td>
<td>81.44</td>
</tr>
</tbody>
</table>

Table 6: Overall Scores (%)

<table>
<thead>
<tr>
<th>Data</th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development</td>
<td>82.80</td>
<td>79.05</td>
<td>80.88</td>
</tr>
<tr>
<td>Label.</td>
<td>89.18</td>
<td>84.97</td>
<td>87.02</td>
</tr>
<tr>
<td>Macro</td>
<td>84.08</td>
<td>80.96</td>
<td>82.49</td>
</tr>
<tr>
<td>WSJ+Brown</td>
<td>89.06</td>
<td>85.94</td>
<td>87.47</td>
</tr>
</tbody>
</table>

Table 4: The Results of Syntactic Parsing (%)

<table>
<thead>
<tr>
<th>Data</th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development</td>
<td>82.85</td>
<td>80.08</td>
<td>81.44</td>
</tr>
<tr>
<td>Label.</td>
<td>89.18</td>
<td>84.97</td>
<td>87.02</td>
</tr>
<tr>
<td>Macro</td>
<td>84.08</td>
<td>80.96</td>
<td>82.49</td>
</tr>
<tr>
<td>WSJ+Brown</td>
<td>89.06</td>
<td>85.94</td>
<td>87.47</td>
</tr>
</tbody>
</table>

The arguments \(A_0\) and \(A_1\) that have been marked before the predicate candidate \(p\) and the current argument identification direction after \(p\). Secondly, each example\(^2\) of \(p\) in NomBank or PropBank that depends on the split PoS tag of \(p\) is checked if it partially matches the current processed role labels. If a unique example exists in this form, e.g., \texttt{Before: A0-A1; After: A3}, then this feature returns \(A_3\) as feature value. If no matched or multiple matched examples exist, then this feature returns a default value.

2.3 Word Sense Determination

The shared task of CoNLL-2008 for word sense disambiguation task is to determine the sense of an output predicate. Our system carries out this task by searching for a right example in the given NomBank or PropBank. The semantic role set scheme of each example for an output predicate is checked. If a scheme is found to match the output semantic role set of a predicate, then the corresponding sense for the first match is chosen; otherwise the system outputs ‘01’ as the default sense.

3 Evaluation Results

Our evaluation is carried out on a 64-bit ubuntu Linux installed server with double dual-core AMD Opteron processors of 2.8GHz and 8GB memory. The full training set for CoNLL-2008 is used to train the maximum entropy model. The training for the syntactic parser costs about 200 hours and 4.1GB memory and that for the semantic parser costs about 170 hours and 4.9GB memory. The running time in each case is the sum of all running time for all threads involved. When a parallel optimization technique is applied to speedup the training, the time can be reduced to about 1/3.5 of the above.

The official evaluation results for our system are presented in Tables 4, 5 and 6. Following the official guideline of CoNLL-2008, we use unlabeled attachment score (UAS), labeled attachment score (LAS) and label accuracy to assess the performance of syntactic dependency parsing. For semantic parsing, the unlabeled scores metric the identification performance and the labeled scores the overall performance of semantic labeling.

4 To Do

Although we are unable to follow our plan to do more than what we have done for this shared task, because of the inadequate computational resource and limited time, we have a number of techniques in our anticipation to bring in further performance improvement.

While expecting to accomplish the joint inference of syntactic and semantic parsing, we only have time to complete a system with the former to enhance the latter. But we did have experiments in the early stage of our work to show that a syntactic dependency parser can make use of avail-
able semantic dependency information to enhance its performance by 0.5-1%.

Most errors in our syntactic parsing are related to the dependencies of comma and prepositions. We need to take care of them, for PP attachment is also crucial to the success of semantic parsing. Extra effort is paid, as illustrated in previous work such as (Xue and Palmer, 2004), to handle such cases, especially when a PP is involved. We find in our data that about 1% arguments occur as a grandchild of a predicate through PP attachment.

Syntactic parsing contributes crucially to the overall performance of the joint parsing by providing a solid basis for further semantic parsing. Thus there is reason to believe that improvement of syntactic dependency parsing can be more influential than that of semantic parsing to the overall improvement. Only one model was used for syntactic parsing in our system, in contrast to the existing work using an ensemble technique for further performance enhancement, e.g., (Hall et al., 2007). Again, the latter means much more computational cost should be taken.

Though it was not done before submission deadline, we also tried to enhance the semantic parsing with some more sophisticated inputs from the syntactic parsing. One is predicted syntactic parsed tree input that may be created by cross-validation rather than the gold-standard syntactic input that our submitted semantic parser was actually trained on. Another is the $n$-best outputs of the syntactic parser. However, only the single-best output of the syntactic parser was actually used.

5 Conclusion

As presented in the above sections, our system to participate in the CoNLL-2008 shared task is implemented as two single-stage maximum entropy learning. We have tackled both syntactic and semantic parsing as a word-pair classification problem. Despite the simplicity of this approach, our system has produced promising results.

Acknowledgements

We wish to thank Dr. Wenliang Chen of NICT, Japan for helpful discussions on dependency parsing, and two anonymous reviewers for their valuable comments.

References


