Dependency Parsing with Second-Order Feature Maps and Annotated Semantic Information

Massimiliano Ciaramita
Yahoo! Research
Ocata 1, S-08003
Barcelona, Spain
massi@yahoo-inc.com

Giuseppe Attardi
Dipartimento di Informatica
Università di Pisa
L. B. Pontecorvo 3, I-56127
Pisa, Italy
attardi@di.unipi.it

Abstract

This paper investigates new design options for the feature space of a dependency parser. We focus on one of the simplest and most efficient architectures, based on a deterministic shift-reduce algorithm, trained with the perceptron. By adopting second-order feature maps, the primal form of the perceptron produces models with comparable accuracy to more complex architectures, with no need for approximations. Further gains in accuracy are obtained by designing features for parsing extracted from semantic annotations generated by a tagger. We provide experimental evaluations on the Penn Treebank.

1 Introduction

A dependency tree represents a sentence as a labeled directed graph encoding syntactic and semantic information. The labels on the arcs can represent basic grammatical relations such as "subject" and "object". Dependency trees capture grammatical structures that can be useful in several language processing tasks such as information extraction (Culotta & Sorensen, 2004) and machine translation (Ding & Palmer, 2005). Dependency treebanks are becoming available in many languages, and several approaches to dependency parsing on multiple languages have been evaluated in the CoNLL 2006 and 2007 shared tasks (Buchholz & Marsi, 2006; Nivre et al., 2007).

Dependency parsing is simpler than constituency parsing, since dependency trees do not have extra non-terminal nodes and there is no need for a grammar to generate them. Approaches to dependency parsing either generate such trees by considering all possible spanning trees (McDonald et al., 2005), or build a single tree by means of shift-reduce parsing actions (Yamada & Matsumoto, 2003). Deterministic dependency parsers which run in linear time have also been developed (Nivre & Scholz, 2004; Attardi, 2006). These parsers process the sentence sequentially, hence their efficiency makes them suitable for processing large amounts of text, as required, for example, in information retrieval applications.

Recent work on dependency parsing has highlighted the benefits of using rich feature sets and high-order modeling. Yamada and Matsumoto (2003) showed that learning an SVM model in the dual space with higher-degree polynomial kernel functions improves significantly the parser's accuracy. McDonald and Pereira (2006) have shown that incorporating second order features relating to adjacent edge pairs improves the accuracy of maximum spanning tree parsers (MST). In the SVM-based approach, if the training data is large, it is not feasible to train a single model. Rather, Yamada and Matsumoto (see also (Hall et al., 2006)) partition the training data in different sets, on the basis of Part-of-Speech, then train one dual SVM model per set. While this approach simplifies the learning task it makes the parser more sensitive to the error rate of the POS tagger. The second-order MST algorithm has cubic time complexity. For non-projective languages the algorithm is NP-hard and McDonald and Pereira (2006) introduce an approximate algorithm to handle such cases.

In this paper we extend shift reduce parsing with second-order feature maps which explicitly repre-

Proceedings of the 10th Conference on Parsing Technologies, pages 133–143,
Prague, Czech Republic, June 2007.

sent all feature pairs. Also the augmented feature sets impose additional computational costs. However, excellent efficiency/accuracy trade-off is achieved by using the perceptron algorithm, without the need to resort to approximations, producing high-accuracy classifiers based on a single model.

We also evaluate a novel set of features for parsing. Recently various forms of shallow semantic processing have been investigated such as named-entity recognition (NER), semantic role labeling (SRL) and relation extraction. Syntactic parsing can provide useful features for these tasks; e.g., Punyakanok et al. (2005) show that full parsing is effective for semantic role labeling (see also related approaches evaluated within the CoNNL 2005 shared task (Carreras et al., 2005)). However, no evidence has been provided so far that annotated semantic information can be leveraged for improving parser performance. We report experiments showing that adding features extracted by an entity tagger improves the accuracy of a dependency parser.

2 Dependency parsing

A dependency parser takes as input a sentence s and returns a dependency graph d. Figure 1 shows a dependency tree for the sentence "Last week CBS Inc. canceled 'The People Next Door'."[1]. Dependencies are represented as labeled arrows from the *head* of the relation to the *modifier* word; thus, in the example, "Inc." is the modifier of a dependency labeled "SUB" (subject) to the main verb, the head, "canceled".

In statistical syntactic parsing a generator (e.g., a PCFG) is used to produce a number of candidate trees (Collins, 2000) with associated probability scores. This approach has been used also for dependency parsing, generating spanning trees as candidates and computing the maximum spanning tree (MST) using discriminative learning algorithms (McDonald et al., 2005). Second-order MST dependency parsers currently represent the state of the art in terms of accuracy. Yamada and Matsumoto (2003) proposed a deterministic classifier-based parser. Instead of learning directly which tree to assign to a sentence, the parser learns which *Shift/Reduce* actions to use in building the tree. Parsing is cast as a classification problem: at each step the parser applies a classifier to the features representing its current state to predict which action to perform on the tree. Similar deterministic approaches to parsing have been investigated also in the context of constituent parsing (Wong & Wu, 1999; Kalt, 2004).

Nivre and Scholz (2004) proposed a variant of the model of Yamada and Matsumoto that reduces the complexity, from the worst case quadratic to linear. Attardi (2006) proposed a variant of the rules that handle non-projective relations while parsing deterministically in a single pass. Shift-reduce algorithms are simple and efficient, yet competitive in terms of accuracy: in the CoNLL-X shared task, for several languages, there was no statistically significant difference between second-order MST parsers and shift-reduce parsers.

[1]The figure also contains entity annotations which will be explained below in Section 4.1.

3 A shift-reduce parser

We build upon DeSR, the shift-reduce parser described in (Attardi, 2006). This and Nivre and Scholz's (2004) provide among the simplest and most efficient methods. This parser constructs dependency trees by scanning input sentences in a single left-to-right pass and performing shift/reduce parsing actions. The parsing algorithm is fully deterministic and has linear complexity. The parser's behavior can be described as repeatedly selecting and applying a parsing rule to transform its state, while advancing through the sentence. Each token is analyzed once and a decision is made locally concerning the action to take, that is, without considering global properties of the tree being built. Nivre (2004) investigated the issue of (strict) incrementality for this type of parsers; i.e., if at any point of the analysis the processed input forms one connected structure. Nivre found that strict incrementality is not guaranteed within this parsing framework, although for correctly parsed trees the property holds in almost 90% of the cases.

3.1 Parsing algorithm

The state of the parser is represented by a triple $\langle S, I, A\rangle$, where S is the stack, I is the list of input tokens that remain to be processed and A is the arc

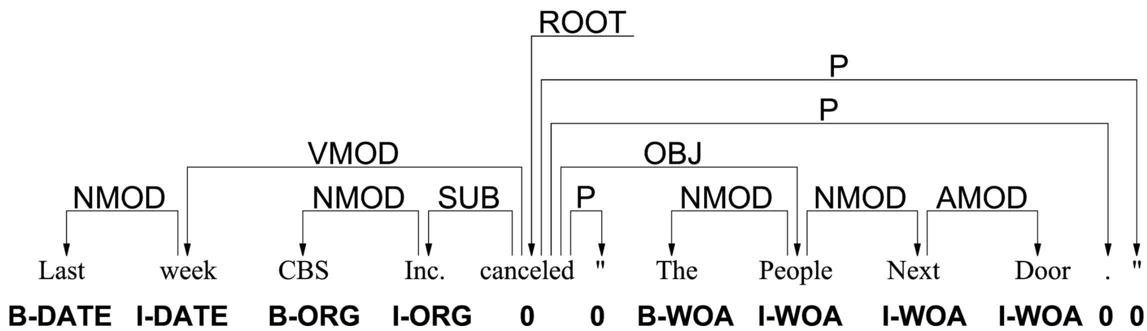

Figure 1. A dependency tree from the Penn Treebank, with additional entity annotation from the BBN corpus.

relation for the dependency graph, which consists of a set of labeled arcs (w_i, r, w_j), where $w_i, w_j \in W$ (the set of tokens), $d \in D$ (the set of dependencies). Given an input sentence s, the parser is initialized to $\langle \emptyset, s, \emptyset \rangle$, and terminates at configuration $\langle s, \emptyset, A \rangle$. There are three parsing schemata:

(1) Shift $\frac{\langle S,n|I,A \rangle}{\langle n|S,I,A \rangle}$

(2) $\mathrm{Right_r}$ $\frac{\langle s|S,n|I,A \rangle}{\langle S,n|I,A \cup \{(s,r,n)\} \rangle}$

(3) $\mathrm{Left_r}$ $\frac{\langle s|S,n|I,A \rangle}{\langle S,s|I,A \cup \{(n,r,s)\} \rangle}$

The Shift rule advances on the input; each $\mathrm{Left_r}$ and $\mathrm{Right_r}$ rule creates a link r between the next input token n and the top token on the stack s. For producing labeled dependencies the rules $\mathrm{Left_r}$ and $\mathrm{Right_r}$ are instantiated several times once for each dependency label.

Additional parsing actions (cf. (Attardi, 2006)) have been introduced for handling non-projective dependency trees: i.e., trees that cannot be drawn in the plane without crossing edges. However, they are not needed in the experiments reported here, because in the Penn Treebank used in our experiments dependencies are extracted without considering empty nodes and the resulting trees are all projective[2].

The pseudo code in Algorithm 1 reproduces schematically the parsing process.

The function $\mathrm{getContext}()$ extracts a vector of features $\mathbf{x}$ relative to the structure built up to that point from the context of the current token, i.e., from a subset of I, S and A. The step $\mathrm{estimateAction}()$ predicts a parsing action y, given a trained model α and $\mathbf{x}$. The final step $\mathrm{performAction}()$ updates the state according to the predicted parsing rule.

Algorithm 1: DeSR: Dependency Shift Reduce parser.

input: $s = w_1, w_2, ..., w_n$
begin
  $S \leftarrow \langle \rangle$
  $I \leftarrow \langle w_1, w_2, ..., w_n \rangle$
  $A \leftarrow \langle \rangle$
  while $I \neq \langle \rangle$ **do**
    $\mathbf{x} \leftarrow \mathrm{getContext}(S, I, A)$
    $y \leftarrow \mathrm{estimateAction}(\mathbf{x}, \alpha)$
    $\mathrm{performAction}(y, S, I, A)$
end

3.2 Features

The set of features used in this paper were chosen with a few simple experiments on the development data as a variant of a generic model. The only features of the tokens used are “Lemma”, “Pos” and “Dep”: “Lemma” refers to the morphologically simplified form of the token, “Pos” is the Part-of-Speech and “Dep” is the label on a dependency. “Child” refers to the child of a node (right or left): up to two furthest children of a node are considered. Table 1 lists which feature is extracted for which token: negative numbers refer to tokens on the stack, positive numbers refer to input tokens. As an example, POS(-1) is the Part-of-Speech of the token on the top of the stack, while Lemma(0) is the lemma of the next token in the input, PosLeftChild(-1) extracts the Part-of-Speech of the leftmost child of the token on the top of the stack, etc.

[2]Instead, the version of the Penn Treebank used for the CoNLL 2007 shared task includes also non-projective representations.

FEATURES	TOKEN					
	Stack		Input			
Lemma	-2	-1	0	1	2	3
Pos	-2	-1	0	1	2	3
LemmaLeftChild		-1	0			
PosLeftChild		-1	0			
DepLeftChild		-1	0			
LemmaRightChild		-1	0			
PosRightChild		-1	0			
DepRightChild		-1				
LemmaPrev			0			
PosSucc		-1				

Table 1. Configuration of the feature parameters used in the experiments.

3.3 Learning a parsing model with the perceptron

The problem of learning a parsing model can be framed as a classification task where each class $y_i \in \mathcal{Y}$ represents one of k possible parsing actions. Each of such actions is associated with a weight vector $\alpha_k \in \mathbb{R}^d$. Given a datapoint $\mathbf{x} \in \mathcal{X}$, a d-dimensional vector of binary features in the input space $\mathcal{X}$, a parsing action is chosen with a winner-take-all discriminant function:

$$\text{estimateAction}(\mathbf{x}, \alpha) = \arg\max_k f(\mathbf{x}, \alpha_k) \quad (4)$$

when using a linear classifier, such as the perceptron or SVM, $f(\mathbf{u}, \mathbf{v}) = \langle \mathbf{u}, \mathbf{v} \rangle$ is the inner product between vectors $\mathbf{u}$ and $\mathbf{v}$.

We learn the parameters α from the training data with the perceptron (Rosemblatt, 1958), in the online multiclass formulation of the algorithm (Crammer & Singer, 2003) with uniform negative updates. The perceptron has been used in previous work on dependency parsing by Carreras et al. (2006), with a parser based on Eisner's algorithm (Eisner, 2000), and also on incremental constituent parsing (Collins & Roark, 2006). Also the MST parser of McDonald uses a variant of the perceptron algorithm (McDonald, 2006). The choice is motivated by the simplicity and performance of perceptrons, which have proved competitive on a number of tasks; e.g., in shallow parsing, where perceptron's performance is comparable to that of Conditional Random Field models (Sha & Pereira, 2003).

The only adjustable parameter of the model is the number of instances T to use for training. We fixed T using the development portion of the data. In our experiments, the best value is between 20 and 30 times the size of the training data. To regularize the model we take as the final model the average of all weight vectors posited during training (Collins, 2002). Algorithm 2 illustrates the perceptron learning procedure. The final average model can be computed efficiently during training without storing the individual α vectors (e.g., see (Ciaramita & Johnson, 2003)).

Algorithm 2: Average multiclass perceptron

input : $\mathcal{S} = (\mathbf{x}_i, y_i)^N; \alpha_k^0 = \vec{0},\ \forall k \in \mathcal{Y}$
for $t = 1$ **to** T **do**
  choose j
  $E^t = \{r \in \mathcal{Y} : \langle \mathbf{x}_j, \alpha_r^t \rangle \geq \langle \mathbf{x}_j, \alpha_{y_j}^t \rangle\}$
  if $|E^t| > 0$ **then**
    $\alpha_r^{t+1} = \alpha_r^t - \frac{\mathbf{x}_j}{|E^t|},\ \forall r \in E^t$
    $\alpha_{y_j}^{t+1} = \alpha_{y_j}^t + \mathbf{x}_j$
output: $\alpha_k = \frac{1}{T}\sum_t \alpha_k^t,\ \forall k \in \mathcal{Y}$

3.4 Higher-order feature spaces

Yamada and Matsumoto (2003) and McDonald and Pereira (2006) have shown that higher-order feature representations and modeling can improve parsing accuracy, although at significant computational costs. To make SVM training feasible in the dual model with polynomial kernels, Yamada and Matsumoto split the training data into several sets, based on POS tags, and train a parsing model for each set. McDonald and Pereira's second-order MST parser has $O(n^3)$ complexity, while for handling non-projective trees, otherwise an NP-hard problem, the parser resorts to an approximate algorithm. Here we discuss how the feature representation can be enriched to improve parsing while maintaining the simplicity of the shift-reduce architecture, and performing discriminative learning without partitioning the training data.

The linear classifier (see Equation 4) learned with the perceptron is inherently limited in the types of solutions it can learn. As originally pointed out by Minsky and Papert (1969), there are problems which require non-linear solutions that cannot be learned by such models. A simple workaround this limitation relies on feature maps $\mathbf{\Phi} : \mathbb{R}^d \rightarrow \mathbb{R}^h$ that map the input vectors $\mathbf{x} \in \mathcal{X}$ into some higher h-dimensional representation $\mathbf{\Phi}(\mathcal{X}) \subset \mathbb{R}^h$, the feature space. The feature space can represent, for example, all combinations of individual features in the input space. We define a feature map which extracts all second order features of the form $x_i x_j$; i.e., $\mathbf{\Phi}(\mathbf{x}) = (x_i, x_j | i = 1, ..., d, j = i, ..., d)$. The linear perceptron working in $\mathbf{\Phi}(\mathcal{X})$ effectively implements a non-linear classifier in the original input space $\mathcal{X}$. One shortcoming of this approach is that it inflates considerably the feature representation and might not scale. In general, the number of features of degree g over an input space of dimension d is $\binom{d+g-1}{g}$. In practice, a second-order feature map can be handled with reasonable efficiency by the perceptron. We call this the 2nd-order model, which uses a modified scoring function:

$$g(\mathbf{x}, \alpha_k) = f(\Phi(\mathbf{x}), \alpha_k) \tag{5}$$

where also α_k is h-dimensional. The proposed feature map is equivalent to a polynomial kernel function of degree two. Yamada and Matsumoto (2003) have shown that the degree two polynomial kernel has superior accuracy than the linear model and polynomial kernels of higher degrees. However, using the dual model is not always practical for dependency parsing. The discriminant function of the dual model is defined as:

$$f'(\mathbf{x}, \alpha) = \arg\max_k \sum_{i=1}^{N} \alpha_{k,i} \langle \mathbf{x}, \mathbf{x}_i \rangle^g \tag{6}$$

where the weights α are associated with class-instance pairs rather than class-feature pairs. With respect to the discriminant function of equation (4) there is an additional summation. In principle, the inner products can be cached in a Kernel matrix to speed up training.

There are two shortcomings to using such a model in dependency parsing. First, if the amount of training data is large it might not be feasible to store the Kernel matrix; which for a dataset of size N requires $O(N^3)$ computations and $O(N^2)$ space. As an example, the number of training instances N in the Penn Treebank is over 1.8 million, caching the Kernel matrix would require several Terabytes of space. The second shortcoming is independent of training. In predicting a tree for unseen sentences the model will have to recompute the inner products between the observation and all the support vectors; i.e., all class-instance pairs with $\alpha_{k,i} > 0$. The second-order feature map with the perceptron is more efficient and allows faster training and prediction. Training a single parsing model avoids a potential loss of accuracy that occurs when using the technique of partitioning the training data according to the POS. Inaccurate predictions of the POS can affect significantly the accuracy of the actions predicted, while the single model is more robust, since the POS is just one of the many features used in prediction.

4 Semantic features

Semantic information is used implicitly in parsing. For example, conditioning on lexical heads provides a source of semantic information. There have been a few attempts at using semantic information more explicitly. Charniak's 1997 parser (1997), defined probability estimates backed off to word clusters. Collins and Koo (Collins & Koo, 2005) introduced an improved reranking model for parsing which includes a hidden layer of semantic features. Yi and Palmer (2005) retrained a constituent parser in which phrases were annotated with argument information to improve SRL, however this didn't improve over the output of the basic parser.

In recent years there has been a significant amount of work on semantic annotation tasks such as named-entity recognition, semantic role labeling and relation extraction. There is evidence that dependency and constituent parsing can be helpful in these and other tasks; e.g., by means of tree kernels in question classification and semantic role labeling (Zhang & Lee, 2003; Moschitti, 2006).

It is natural to ask if also the opposite holds: whether semantic annotations can be used to improve parsing. In particular, it would be interesting to know if entity-like tags can be used for this purpose. One reason for this is that entity tagging is efficient and does not seem to need parsing for achieving top performance. Beyond improving traditional parsing, independently learned semantic tags might be helpful in adapting a parser to a new domain. To the best of our knowledge, no evidence has been produced yet that annotated semantic information can improve parsing. In the following we investigate adding entity tags as features of our parser.

4.1 BBN Entity corpus

The BBN corpus (BBN, 2005) supplements the Wall Street Journal Penn Treebank with annotation of a large set of entity types. The corpus includes annotation of 12 named entity types (Person, Facility, Organization, GPE, Location, Nationality, Product, Event, Work of Art, Law, Language, and Contact-Info), nine nominal entity types (Person, Facility, Organization, GPE, Product, Plant, Animal, Substance, Disease and Game), and seven numeric types (Date, Time, Percent, Money, Quantity, Ordinal and Cardinal). Several of these types are further divided into subtypes[3]. This corpus provides adequate support for experimenting semantic features for parsing.

Figure 1 illustrates the annotation layer provided by the BBN corpus[4]. It is interesting to notice one apparent property of the combination of semantic tags and dependencies. When we consider segments composed of several words there is exactly one dependency connecting a token outside the segment with a token inside the segment; e.g., "CBS Inc." is connected outside only through the token "Inc.", the subject of the main verb. With respect to the rest of the tree, segments tend to form units, with their own internal structure. Intuitively, this information seems relevant for parsing. This locally-structured patterns could help particularly simple algorithms like ours, which have limited knowledge of the global structure being built.

Table 2 lists the 40 most frequent categories in sections 2 to 21 of the BBN corpus, and the percentage of all entities they represent – together more than 97%. Sections 2-21 are comprised of 949,853 tokens, 23.5% of the tokens have a non-null BBN entity tag, on average there is one tagged token every four. The total number of entities is 139,029, 70.5% of which are named entities and nominal concepts, 17% are numerical types and the remaining 12.5% describe time entities.

We designed three new features which extract simple properties of entities from the semantic annotation information:

[3] BBN Corpus documentation.

[4] The full label for "ORG" is "ORG:Corporation", and "WOA" stands for "WorkOfArt:Other".

	TOKEN				
FEATURES	Stack		Input		
AS-0 = EOS+BIO+TAG			0		
AS-1 = EOS+BIO+TAG		-1	0	1	
AS-2 = EOS+BIO+TAG	-2	-1	0	1	2
EOS	-2	-1	0	1	2
BIO	-2	-1	0	1	2
TAG	-2	-1	0	1	2

Table 3. Additional configurations for the models with BBN entity features.

- EOS: Distance to the end of the segment; e.g., EOS("Last") = 1, EOS("canceled") = 0;
- BIO: The first character of the BBN label for a token; e.g., BIO("CBS") = "B", and BIO("canceled") = 0;
- TAG: Full BBN tag for the token; e.g., TAG("CBS") = "B-ORG:Corporation", TAG("week") = "I-DATE".

The feature EOS provides information about the relative position of the token within a segment with respect to the end of the segment. The feature BIO discriminates tokens with no semantic annotation associated, from tokens within a segment and token which start a segment. Finally the feature TAG identifies the full semantic tag associated with the token. With respect to the former two features this bears the most fine-grained semantics. Table 3 summarizes six additional models we implemented. The first three use all additional features together, applied to different sets of tokens, while the last three apply only one feature, on top of the base model, relative to the next token in the input, the following two tokens in the input, and the previous two tokens on the stack.

4.2 Corpus pre-processing

The original BBN corpus has its own tokenization which often does not reflect the Penn Treebank tokenization; e.g., when an entity intersects an hyphenated compound, thus "third-highest" becomes "third$_{ORDINAL}$ - highest". This is problematic for combining entity annotation and dependency trees. Since our main focus is parsing we re-aligned the BBN Corpus with the Treebank tokenization. Thus, for example, when an entity splits a Treebank token we extend the entity boundary to contain

WSJ-BBN Corpus Categories							
Tag	%	Tag	%	Tag	%	Tag	%
PER_DESC	15.5	ORG:CORP	13.7	DATE:DATE	9.2	ORG_DESC:CORP	8.9
PERSON	8.13	MONEY	6.5	CARDINAL	6.0	PERCENT	3.5
GPE:CITY	3.12	GPE:COUNTRY	2.9	ORG:GOV	2.6	NORP:NATION-TY	1.9
DATE:DURATION	1.8	GPE:PROVINCE	1.5	ORG_DESC:GOV	1.4	FAC_DESC:BLDG	1.1
ORG:OTHER	0.7	PROD_DESC:VEHICLE	0.7	ORG_DESC:OTHER	0.6	ORDINAL	0.6
TIME	0.5	GPE_DESC:COUNTRY	0.5	SUBST:OTHER	0.5	SUBST:FOOD	0.5
DATE:OTHER	0.4	NORP:POLITICAL	0.4	DATE:AGE	0.4	LOC:REGION	0.3
SUBST:CHEM	0.3	WOA:OTHER	0.3	FAC_DESC:OTHER	0.3	SUBST:DRUG	0.3
ANIMAL	0.3	GPE_DESC:PROVINCE	0.2	PROD:VEHICLE	0.2	GPE_DESC:CITY	0.2
PRODUCT:OTHER	0.2	LAW	0.2	ORG:POLITICAL	0.2	ORG:EDU	0.2

Table 2. The 40 most frequent labels in sections 2 to 21 of the Wall Street Journal BBN Corpus and the percentage of tags occurrences.

the whole original Treebank token, thus obtaining "third-highest$_{ORDINAL}$" in the example above.

4.3 Semantic tagger

We treated semantic tags as POS tags. A tagger was trained on the BBN gold standard annotation and used it to annotate development and evaluation data. We briefly describe the tagger (see (Ciaramita & Altun, 2006) for more details), a Hidden Markov Model trained with the perceptron algorithm introduced in (Collins, 2002). The tagger uses Viterbi decoding. Label to label dependencies are limited to the previous tag (first order HMM). A generic feature set for NER based on words, lemmas, POS tags, and word shape features was used.

The tagger is trained on sections 2-21 of the BBN corpus. As before, section 22 of the BBN corpus is used for choosing the perceptron's parameter T. The tagger's model is regularized as described for Algorithm 2. The full BBN tagset is comprised of 105 classes organized hierarchically, we ignored the hierarchical organization and treated each tag as an independent class in the standard BIO encoding. The tagger evaluated on section 23 achieves an F-score of 86.8%. The part of speech for the evaluation/development sections was produced with TreeTagger. As a final remark we notice that the tagger's complexity, linear in the length of the sentence, preserves the parser's complexity.

5 Parsing experiments

5.1 Data and setup

We used the standard partitions of the Wall Street Journal Penn Treebank (Marcus et al., 1993); i.e., sections 2-21 for training, section 22 for development and section 23 for evaluation. The constituent trees were transformed into dependency trees by means of a program created by Joakim Nivre that implements the rules proposed by Yamada and Matsumoto, which in turn are based on the head rules of Collins' parser (Collins, 1999)[5]. The lemma for each token was produced using the "morph" function of the WordNet (Fellbaum, 1998) library[6]. The data in the WSJ sections 22 and 23, both for the parser and for the semantic tagger, was POS-tagged using TreeTagger[7], which has an accuracy of 97.0% on section 23.

Training a parsing model on the Wall Street Journal requires a set of 22 classes: 10 of the 11 labels in the dependency corpus generated from the Penn Treebank (e.g., subj, obj, sbar, vmod, nmod, root, etc.) are paired with both a Left and Right actions. In addition, there is in one rule for the "root" label and one for the Shift action. The total number of features found in training ranges from two hundred thousand for the 1st-order model to approximately 20 million of the 2nd-order models.

We evaluated several models, each trained with 1st-order and 2nd-order features. The base model (BASE) only uses the traditional set of features (cf. Table 1). Models EOS, BIO and TAG each use only one type of semantic feature with the configuration described in Table 3. Models AS-0, AS-1, and AS-2 use all three semantic features for the token on the stack in AS-0, plus the previous token on the stack and the new token in the input in AS-1, plus an addi-

[5] The script is available from http://w3.msi.vxu.se/%7enivre/research/Penn2Malt.html

[6] http://wordnet.princeton.edu

[7] TreeTagger is available from http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/

DeSR MODEL	1st-order scores LAS	UAS	Imp	LAC	2nd-order scores LAS	UAS	Imp	LAC
BASE	84.01	85.56	-	88.24	89.20	90.55	-	92.22
EOS	84.89	86.37	+5.6	88.94	89.36	90.64	+1.0	92.37
BIO	84.95	86.37	+6.6	89.06	89.63	90.89	+3.6	92.55
TAG	84.76	86.26	+4.8	88.80	89.54	90.81	+2.8	92.55
AS-0	84.40	85.95	+2.7	88.38	89.41	90.72	+1.8	92.38
AS-1	85.13	86.52	+6.6	89.11	89.57	90.77	+2.3	92.49
AS-2	85.32	86.71	+8.0	89.25	**89.87**	**91.10**	+5.8	**92.68**

Table 4. Results of the different models on WSJ section 23 using the CoNLL scores Labeled attachment score (LAS), Unlabeled attachment score (UAS), and Label accuracy score (LAC). The column labeled "Imp" reports the improvement in terms of relative error reduction with respect to the BASE model for the UAS score. In bold the best results.

tional token from the stack and an additional token from the input for AS-2 (cf. Table 3).

5.2 Results of 2nd-order models

Table 4 summarizes the results of all experiments. We report the following scores, obtained with the CoNLL-X scoring script: labeled attachment score (LAS), unlabeled attachment score (UAS) and label accuracy score (LAC). For the UAS score, the most frequently reported, we include the improvement in relative error reduction.

The 2nd-order base model improves on all measures over the 1st-order model by approximately 5%. The UAS score is 90.55%, with an improvement of 4.9%. The magnitude of the improvement is remarkable and reflects the 4.6% improvement that Yamada and Matsumoto (Yamada & Matsumoto, 2003) report going from the linear SVM to the polynomial of degree two. Our base model's accuracy (90.55% UAS) compares well with the accuracy of the parsers based on the polynomial kernel trained with SVM of Yamada and Matsumoto (UAS 90.3%), and Hall et al. (2006) (UAS 89.4%). We notice in particular that, given the lack of non-projective cases/rules, the parser of Hall et al. (2006) is almost identical to our parser, hence the difference in accuracy (+1.1%) might effectively be due to a better classifier. Yamada & Matsumoto's parser is slightly more complex than our parser, and has quadratic worst-case complexity. Overall, the accuracy of the 2nd-order parser is comparable to that of the 1st-order MST parser (90.7%).

There is no direct evidence that our perceptron produces better classifiers than SVM. Rather, the pattern of results produced by the perceptron seems comparable to that of SVM (Yamada & Matsumoto, 2003). This is a useful finding in itself, given that the former is more efficient: perceptron's update is linear while SVM solves a quadratic problem at each update. However, one major difference between the two approaches lies in the fact that learning with the primal model does not require splitting the model by Part-of-Speech, or other means. As a consequence, beyond the greater simplicity, our method might benefit from not depending so strongly on the quality of POS tagging. POS information is encoded as a feature and contributes its weight to the selection of the parsing action, together with all additionally available information. In the SVM-trained methods the model that makes the prediction for the parsing rule is essentially chosen by an oracle, the prediction of the POS tagger. Furthermore, it might be argued that learning a single model makes a better use of the training data by exploiting the correlations between all datapoints, while in the dual split-training case the interaction is limited to datapoints in the same partition. In any case, second-order feature maps could be used also with SVM or other classifiers. The advantage of using the perceptron lies in the unchallenged accuracy/efficiency trade-off. Finally, we recall that training in the primal model can be performed fully on-line without affecting the resulting model nor the complexity of the algorithm.

5.3 Results of models with semantic features

All models based on semantic features improve over the base model on all measures. The best configura-

Parser	UAS
Hall et al. '06	89.4
Yamada & Matsumoto '03	90.3
DeSR	90.55
McDonald & Pereira 1st-order MST	90.7
DeSR AS-2	91.1
McDonald & Pereira 2nd-order MST	91.5
Sagae & Lavie '06	92.7

Table 5. Comparison of main results on the Penn Treebank dataset.

tion is that of model AS-2 which extracts all semantic features from the widest context. In the 1st-order AS-2 model the improvement, 86.71% UAS (+8% relative error reduction) is more marked than in the 2nd-order AS-2 model, 91.1% UAS (+5.8% error reduction). A possible simple exaplanation is that some information captured by the semantic features is correlated with other higher-order features which do not occur in the 1st-order encoding. Overall the accuracy of the DeSR parser with semantic information is slightly inferior to that of the second-order MST parser (McDonald & Pereira, 2006) (91.5% UAS). The best result on this dataset to date (92.7% UAS) is that of Sagae and Lavie (Sagae & Lavie, 2006) who use a parser which combines the predictions of several pre-existing parsers, including McDonald's and Nivre's parsers. Table 5 lists the main results to date on the version of the Penn Treebank for dependency parsing task used in this paper.

In Table 4 we also evaluate the gain obtained by adding one semantic feature type at a time (cf. rows EOS/BIO/TAG). These results show that all semantic features provide some improvement (with the dubious case of EOS in the 2nd-order model). The BIO encoding seems to produce the most accurate features. This could be promising because it suggests that the benefit does not depend only on the specific tags, but that the segmentation in itself is important. Hence tagging could improve the adaptation of parsers to new domains even if only generic tagging methods are available.

5.4 Remarks on efficiency

All experiments were performed on a 2.4GHz AMD Opteron CPU machine with 32GB RAM. The 2nd-order parser uses almost 3GB of memory. While

	Parsing time/sec	
Parser	English	Chinese
MST 2n-order	97.52	59.05
MST 1st-order	76.62	49.13
DeSR	36.90	21.22

Table 6. Parsing times for the CoNNL 2007 English and Chinese datasets for MST and DeSR.

it is several times slower and larger than the 1st-order model[8] the 2nd-order model performance is still competitive. It takes 3 minutes (user time) to parse section 23, POS tagging included. In training, the model takes about 1 hour to process the full dataset once. As a comparison, Hall et al. (2006) reports 1.5 hours for training the partitioned SVM model and 10 minutes for parsing the evaluation set on the same Penn Treebank data. We also compared directly the parsing time of our parser with that of the MST parser using the version 0.4.3 of MST-Parser[9]. For these experiments we used two datasets from the CoNLL 2007 shared task for English and Chinese. Table 6 reports the times, in seconds, to parse the test sets for these languages on a 3.3GHz Xeon machine with 4 GB Ram, of the MST 1st and 2nd-order parser and DeSR parser (without semantic features).

The architecture of the model presented here offers several options for optimization. For example, implementing the α models with full vectors rather than hash tables speeds up parsing by a factor of three, at the expense of memory. Alternatively, memory load in training can be reduced, at the expense of time, by using on-line training. However, the most valuable option for space need reduction might be to filter out low-frequency second-order features. Since the frequency of such features seems to follow a power law distribution, this reduces significantly the feature space size even for low thresholds at small accuracy expense. In this paper however we focused on the full model, no approximations were required to run the experiments.

[8] The 1st-order parser takes 7 seconds (user time) to process Section 23.

[9] Available from sourceforge.net.

6 Conclusion

We explored the design space of a dependency parser by modeling and extending the feature representation, while adopting one of the simplest parsing architecture: a single-pass deterministic shift-reduce algorithm trained with a regularized multiclass perceptron. We showed that with the perceptron it is possible to adopt higher-order feature maps equivalent to polynomial kernels without need of approximating the model (although this remains an option for optimization). The resulting models achieve accuracies comparable (or better) to more complex architectures based on dual SVM training, and faster parsing on unseen data. With respect to learning, it is possible that more sophisticated formulations of the perceptron (e.g. MIRA (Crammer & Singer, 2003)) could provide further gains in accuracy, as shown with the MST parser (McDonald et al., 2005).

We also experimented with novel types of semantic features, extracted from the annotations produced by an entity tagger trained on the BBN corpus. This model further improves over the standard model yielding an additional 5.8% relative error reduction. Although the magnitude of the improvement is not striking, to the best of our knowledge this is the first encouraging evidence that annotated semantic information can improve parsing and suggests several options for further research. For example, this finding might indicate that this type of approach, which combines semantic tagging and parsing, is viable for the adaptation of parsing to new domains for which semantic taggers exist. Semantic features could be also easily included in other types of dependency parsing algorithms, e.g., MST, and in current methods for constituent parse reranking (Collins, 2000; Charniak & Johnson, 2005).

For future research several issues concerning the semantic features could be tackled. We notice that more complex semantic features can be designed and evaluated. For example, it might be useful to guess the "head" of segments with simple heuristics, i.e., the guess the node which is more likely to connect the segment with the rest of the tree, which all internal components of the entity depend upon. It would be also interesting to extract semantic features from taggers trained on different datasets and based on different tagsets.

Acknowledgments

The first author would like to thank Thomas Hofmann for useful inputs concerning the presentation of the issue of higher-order feature representations of Section 3.4. We would also like to thank Brian Roark and the anonymous reviewers for useful comments and pointers to related work.

References

G. Attardi. 2006. Experiments with a Multilanguage Non-Projective Dependency Parser. In *Proceedings of CoNNL-X 2006*.

BBN. 2005. BBN Pronoun Coreference and Entity Type Corpus. Linguistic Data Consortium (LDC) catalog number LDC2005T33.

S. Buchholz and E. Marsi. 2006. Introduction to CoNNL-X Shared Task on Multilingual Dependency Parsing. In *Proceedings of CoNNL-X 2006*.

X. Carreras and L. Màrquez. 2005. Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling. In *Proceedings of CoNLL 2005*.

X. Carreras, M. Surdeanu, and L. Màrquez. 2006 Projective Dependency Parsing with Perceptron. In *Proceedings of CoNLL-X*.

E. Charniak. 1997. Statistical Parsing with a Context-Free Grammar and Word Statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence AAAI*.

E. Charniak and M. Johnson. 2005. Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking. In *Proceedings of ACL 2005*.

M. Ciaramita and Y. Altun. 2006. Broad-Coverage Sense Disambiguation and Information Extraction with a Supersense Sequence Tagger. In *Proceedings of EMNLP 2006*.

M. Ciaramita and M. Johnson. 2003. Supersense Tagging of Unknown Nouns in WordNet. In *Proceedings of EMNLP 2003*.

M. Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. Thesis, University of Pennsylvania.

M. Collins. 2000. Discriminative Reranking for Natural Language Parsing. In *Proceedings of ICML 2000.*

M. Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of EMNLP 2002.*

M. Collins and T. Koo. 2005. Hidden-Variable Models for Discriminative Reranking. In *P*roceedings of EMNLP 2005.

M. Collins and B. Roark. 2004. Incremental Parsing with the Perceptron Algorithm. In *Proceedings of ACL 2004*.

K. Crammer and Y. Singer. 2003. *Ultraconservative Online Algorithms for Multiclass Problems.* Journal of Machine Learning Research 3: pp.951-991.

A. Culotta and J. Sorensen. 2004. Dependency Tree Kernels for Relation Extraction. In *Proceedings of ACL 2004*.

Y. Ding and M. Palmer. 2005. Machine Translation using Probabilistic Synchronous Dependency Insertion Grammars. In *Proceedings of ACL 2005*.

J. Eisner. 2000. Bilexical Grammars and their Cubic-Time Parsing Algorithms. In H.C. Bunt and A. Nijholt, eds. *New Developments in Natural Language Parsing*, pp. 29-62. Kluwer Academic Publishers.

C. Fellbaum. 1998. *WordNet: An Electronic Lexical Database* MIT Press, Cambridge, MA. 1969.

J. Hall, J. Nivre and J. Nilsson. 2006. Discriminative Classifiers for Deterministic Dependency Parsing. In *P*roceedings of the COLING/ACL 2006.

T. Kalt. 2004. Induction of Greedy Controllers for Deterministic Treebank Parsers. In *Proceedings of EMNLP 2004*.

M. Marcus, B. Santorini and M. Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2): pp. 313-330.

R. McDonald. 2006. *Discriminative Training and Spanning Tree Algorithms for Dependency Parsing*. Ph.D. Thesis, University of Pennsylvania.

R. McDonald, F. Pereira, K. Ribarov and J. Hajič. 2005. Non-projective Dependency Parsing using Spanning Tree Algorithms. In *Proceedings of HLT-EMNLP 2005*.

R. McDonald and F. Pereira. 2006. Online Learning of Approximate Dependency Parsing Algorithms. In *Proceedings of EACL 2006*.

M.L. Minsky and S.A. Papert. 1969. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, MA. 1969.

A. Moschitti. 2006. Efficient Convolution Kernels for Dependency and Constituent Syntactic Trees. In *Proceedings of ECML 2006*.

J. Nivre. 2004. Incrementality in Deterministic Dependency Parsing. In *Incremental Parsing: Bringing Engineering and Cognition Together. Workshop at ACL-2004*.Spain, 50-57.

J. Nivre, J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel and D. Yuret. 2007. The CoNLL 2007 Shared Task on Dependency Parsing, In Proceedings of EMNLP-CoNLL 2007.

J. Nivre and M. Scholz. 2004. Deterministic Dependency Parsing of English Text. In *Proceedings of COLING 2004*.

V. Punyakanok, D. Roth, and W. Yih. 2005. The Necessity of Syntactic Parsing for Semantic Role Labeling. In *Proceedings of IJCAI 2005*.

F. Rosemblatt. 1958. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psych. Rev.*, 68: pp. 386-407.

K. Sagae and A. Lavie. 2005. Parser Combination by Reparsing. In *Proceedings of HLT-NAACL 2006*.

F. Sha and F. Pereira. 2003. Shallow Parsing with Conditional Random Fields. In *Proceedings of HLT-NAACL 2003*.

H. Yamada and Y. Matsumoto. 2003. Statistical Dependency Analysis with Support Vector Machines. In *Proceedings of the Eighth International Workshop on Parsing Technologies. Nancy, France*.

S. Yi and M. Palmer. 2005. The Integration of Syntactic Parsing and Semantic Role Labeling. In *Proceedings of CoNLL 2005*.

A. Wong and D. Wu. 1999. Learning a Lightweight Deterministic Parser. In *Proceedings of EUROSPEECH 1999*.

D. Zhang and W.S. Less. 2003. Question Classification using Support Vector Machines. In *Proceedings of SIGIR 2003*.