Advances in Discriminative Parsing

Joseph Turian and I. Dan Melamed
{lastname}@cs.nyu.edu
Computer Science Department
New York University
New York, New York 10003

Abstract

The present work advances the accuracy and training speed of discriminative parsing. Our discriminative parsing method has no generative component, yet surpasses a generative baseline on constituent parsing, and does so with minimal linguistic cleverness. Our model can incorporate arbitrary features of the input and parse state, and performs feature selection incrementally over an exponential feature space during training. We demonstrate the flexibility of our approach by testing it with several parsing strategies and various feature sets. Our implementation is freely available at: http://nlp.cs.nyu.edu/parser/.

1 Introduction

Discriminative machine learning methods have improved accuracy on many NLP tasks, including POS-tagging, shallow parsing, relation extraction, and machine translation. Some advances have also been made on full syntactic constituent parsing. Successful discriminative parsers have relied on generative models to reduce training time and raise accuracy above generative baselines (Collins & Roark, 2004; Henderson, 2004; Taskar et al., 2004). However, relying on information from a generative model might prevent these approaches from realizing the accuracy gains achieved by discriminative methods on other NLP tasks. Another problem is training speed: Discriminative parsers are notoriously slow to train.

In the present work, we make progress towards overcoming these obstacles. We propose a flexible, end-to-end discriminative method for training parsers, demonstrating techniques that might also be useful for other structured prediction problems. The proposed method does model selection without ad-hoc smoothing or frequency-based feature cutoffs. It requires no heuristics or human effort to optimize the single important hyper-parameter. The training regime can use all available information from the entire parse history. The learning algorithm projects the hand-provided features into a compound feature space and performs incremental feature selection over this large feature space. The resulting parser achieves higher accuracy than a generative baseline, despite not using a generative model as a feature.

Section 2 describes the parsing algorithm. Section 3 presents the learning method. Section 4 presents experiments with discriminative parsers built using these methods. Section 5 compares our approach to related work.

2 Parsing Algorithm

The following terms will help to explain our work. A **span** is a range over contiguous words in the input. Spans **cross** if they overlap but neither contains the other. An **item** is a (span, label) pair. A **state** is a partial parse, i.e. a set of items, none of whose spans may cross. A parse **inference** is a (state, item) pair, i.e. a state and an item to be added to it. The **frontier** of a state consists of the items with no parents yet. The **children** of a candidate inference are the frontier items below the item to be inferred, and the **head** of a candidate inference is the child item chosen by English head rules (Collins, 1999, pp. 238–240). A parse **path** is a sequence of parse inferences. For some input sentence and training parse tree, a state is **correct** if the parser can infer zero or more additional items to obtain the training parse tree, and an inference

Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL, pages 873–880,
Sydney, July 2006.

is correct if it leads to a correct state.

Given input sentence s, the parser searches for parse $\hat{p}$ out of the possible parses $P(s)$:

$$\hat{p} = \arg\min_{p \in P(s)} C_\Theta(p) \quad (1)$$

where $C_\Theta(p)$ is the **cost** of parse p under model Θ:

$$C_\Theta(p) = \sum_{i \in p} c_\Theta(i) \quad (2)$$

Section 3.1 describes how to compute $c_\Theta(i)$. Because $c_\Theta(i) \in \mathbb{R}^+$, the cost of a partial parse monotonically increases as we add items to it.

The parsing algorithm considers a succession of states. The initial state contains terminal items, whose labels are the POS tags given by the tagger of Ratnaparkhi (1996). Each time we pop a state from the agenda, c_Θ computes the costs for the candidate bottom-up inferences generated from that state. Each candidate inference results in a successor state to be placed on the agenda.

The cost function c_Θ can consider arbitrary properties of the input and parse state. We are not aware of any tractable solution to Equation 1, such as dynamic programming. Therefore, the parser finds $\hat{p}$ using a variant of uniform-cost search. The parser implements the search using an agenda that stores entire states instead of single items. Each time a state is popped from the agenda, the parser uses depth-first search starting from the state that was popped until it (greedily) finds a complete parse. In preliminary experiments, this search strategy was faster than standard uniform-cost search (Russell & Norvig, 1995).

3 Training Method

3.1 General Setting

Our training set I consists of candidate inferences from the parse trees in the training data. From each training inference $i \in I$ we generate the tuple $\langle X(i), y(i), b(i) \rangle$. $X(i)$ is a feature vector describing i, with each element in $\{0, 1\}$. We will use $X_f(i)$ to refer to the element of $X(i)$ that pertains to feature f. $y(i) = +1$ if i is correct, and $y(i) = -1$ if not. Some training examples might be more important than others, so each is given a **bias** $b(i) \in \mathbb{R}^+$, as detailed in Section 3.3.

The goal during training is to induce a hypothesis $h_\Theta(i)$, which is a real-valued inference scoring function. In the present work, h_Θ is a linear model parameterized by a real vector Θ, which has one entry for each feature f:

$$h_\Theta(i) = \Theta \cdot X(i) = \sum_f \Theta_f \cdot X_f(i) \quad (3)$$

The sign of $h_\Theta(i)$ predicts the y-value of i and the magnitude gives the confidence in this prediction.

The training procedure optimizes Θ to minimize the expected risk R_Θ over training set I. R_Θ is the **objective** function, a combination of **loss** function L_Θ and **regularization** term Ω_Θ:

$$R_\Theta(I) = L_\Theta(I) + \Omega_\Theta \quad (4)$$

The loss of the inference set decomposes into the loss of individual inferences:

$$L_\Theta(I) = \sum_{i \in I} l_\Theta(i) \quad (5)$$

In principle, l_Θ can be any loss function, but in the present work we use the log-loss (Collins et al., 2002):

$$l_\Theta(i) = b(i) \cdot \ln(1 + \exp(-\mu_\Theta(i))) \quad (6)$$

and $\mu_\Theta(i)$ is the **margin** of inference i:

$$\mu_\Theta(i) = y(i) \cdot h_\Theta(i) \quad (7)$$

Inference cost $c_\Theta(i)$ in Equation 2 is $l_\Theta(i)$ computed using $y(i) = +1$ and $b(i) = 1$, i.e.:

$$c_\Theta(i) = \ln(1 + \exp(-h_\Theta(i))) \quad (8)$$

Ω_Θ in Equation 4 is a regularizer, which penalizes complex models to reduce overfitting and generalization error. We use the ℓ_1 penalty:

$$\Omega_\Theta = \sum_f \lambda \cdot |\Theta_f| \quad (9)$$

where λ is a parameter that controls the strength of the regularizer. This choice of objective R_Θ is motivated by Ng (2004), who suggests that, given a learning setting where the number of irrelevant features is exponential in the number of training examples, we can nonetheless learn effectively by building decision trees to minimize the ℓ_1-regularized log-loss. On the other hand, Ng (2004) suggests that most of the learning algorithms commonly used by discriminative parsers *will* overfit when exponentially many irrelevant features are present.[1]

Learning over an exponential feature space is the very setting we have in mind. *A priori*, we define only a set A of simple **atomic** features (given

[1] including the following learning algorithms:
- unregularized logistic regression
- logistic regression with an ℓ_2 penalty (i.e. a Gaussian prior)
- SVMs using most kernels
- multilayer neural nets trained by backpropagation
- the perceptron algorithm

in Section 4). The learner then induces **compound** features, each of which is a conjunction of possibly negated atomic features. Each atomic feature can have one of three values (yes/no/don't care), so the size of the compound feature space is $3^{|A|}$, exponential in the number of atomic features. It was also exponential in the number of training examples in our experiments ($|A| \approx |I|$).

3.2 Boosting ℓ_1-Regularized Decision Trees

We use an ensemble of confidence-rated decision trees (Schapire & Singer, 1999) to represent h_Θ.[2] The path from the root to each node n in a decision tree corresponds to some compound feature f, and we write $\varphi(n) = f$. To score an inference i using a decision tree, we percolate the inference's features $X(i)$ down to a leaf n and return confidence $\Theta_{\varphi(n)}$. An inference i percolates down to node n iff $X_{\varphi(n)} = 1$. Each leaf node n keeps track of the parameter value $\Theta_{\varphi(n)}$.[3] The score $h_\Theta(i)$ given to an inference i by the whole ensemble is the sum of the confidences returned by the trees in the ensemble.

Listing 1 Outline of training algorithm.

```
1:  procedure TRAIN(I)
2:      ensemble ← ∅
3:      λ ← ∞
4:      while dev set accuracy is increasing do
5:          t ← tree with one (root) node
6:          while the root node cannot be split do
7:              decay ℓ1 parameter λ
8:          while some leaf in t can be split do
9:              split the leaf to maximize gain
10:         percolate every i ∈ I to a leaf node
11:         for each leaf n in t do
12:             update Θ_φ(n) to minimize R_Θ
13:         append t to ensemble
```

Listing 1 presents our training algorithm. At the beginning of training, the ensemble is empty, $\Theta = \mathbf{0}$, and the ℓ_1 parameter λ is set to ∞ (Steps 1.2 and 1.3). We train until the objective cannot be further reduced for the current choice of λ. We then determine the accuracy of the parser on a held-out development set using the previous λ value (before it was decreased), and stop training when this accuracy reaches a plateau (Step 1.4). Otherwise, we relax the regularization penalty by decreasing λ (Steps 1.6 and 1.7) and continue training. In this way, instead of choosing the best λ heuristically, we can optimize it during a single training run (Turian & Melamed, 2005).

Each training iteration (Steps 1.5–1.13) has several steps. First, we choose some compound features that have high magnitude gradient with respect to the objective function. We do this by building a new decision tree, whose leaves represent the chosen compound features (Steps 1.5–1.9). Second, we confidence-rate each leaf to minimize the objective over the examples that percolate down to that leaf (Steps 1.10–1.12). Finally, we append the decision tree to the ensemble and update parameter vector Θ accordingly (Step 1.13). In this manner, compound feature selection is performed incrementally *during* training, as opposed to *a priori*.

Our strategy minimizing the objective $R_\Theta(I)$ (Equation 4) is a variant of steepest descent (Perkins et al., 2003). To compute the gradient of the unpenalized loss L_Θ with respect to the parameter Θ_f of feature f, we have:

$$\frac{\partial L_\Theta(I)}{\partial \Theta_f} = \sum_{i \in I} \frac{\partial l_\Theta(i)}{\partial \mu_\Theta(i)} \cdot \frac{\partial \mu_\Theta(i)}{\partial \Theta_f} \tag{10}$$

where:

$$\frac{\partial \mu_\Theta(i)}{\partial \Theta_f} = y(i) \cdot X_f(i) \tag{11}$$

Using Equation 6, we define the **weight** of an example i under the current model as the rate at which loss decreases as the margin of i increases:

$$w_\Theta(i) = -\frac{\partial l_\Theta(i)}{\partial \mu_\Theta(i)} = b(i) \cdot \frac{1}{1 + \exp(\mu_\Theta(i))} \tag{12}$$

Recall that $X_f(i)$ is either 0 or 1. Combining Equations 10–12 gives:

$$\frac{\partial L_\Theta(I)}{\partial \Theta_f} = -\sum_{\substack{i \in I \\ X_f(i)=1}} y(i) \cdot w_\Theta(i) \tag{13}$$

We define the **gain** of feature f as:

$$G_\Theta(I; f) = \max\left(0, \left|\frac{\partial L_\Theta(I)}{\partial \Theta_f}\right| - \lambda\right) \tag{14}$$

Equation 14 has this form because the gradient of the penalty term is undefined at $\Theta_f = 0$. This discontinuity is why ℓ_1 regularization tends to produce sparse models. If $G_\Theta(I; f) = 0$, then the objective $R_\Theta(I)$ is at its minimum with respect to parameter Θ_f. Otherwise, $G_\Theta(I; f)$ is the magnitude

[2] Turian and Melamed (2005) reported that decision trees applied to parsing have higher accuracy and training speed than decision stumps, so we build full decision trees rather than stumps.

[3] Any given compound feature can appear in more than one tree, but each leaf node has a distinct confidence value. For simplicity, we ignore this possibility in our discussion.

of the gradient of the objective as we adjust Θ_f in the appropriate direction.

To build each decision tree, we begin with a root node. The root node corresponds to a dummy "always true" feature. We recursively split nodes by choosing a splitting feature that will allow us to increase the gain. Node n with corresponding compound feature $\varphi(n) = f$ can be split by atomic feature a if:

$$G_\Theta(I; f \wedge a) + G_\Theta(I; f \wedge \neg a) > G_\Theta(I; f) \quad (15)$$

If no atomic feature satisfies the splitting criterion in Equation 15, then n becomes a leaf node of the decision tree and $\Theta_{\varphi(n)}$ becomes one of the values to be optimized during the parameter update step. Otherwise, we choose atomic feature $\hat{a}$ to split node n:

$$\hat{a} = \arg\max_{a \in A} \left(G_\Theta(I; f \wedge a) + G_\Theta(I; f \wedge \neg a)\right) \quad (16)$$

This split creates child nodes n_1 and n_2, with $\varphi(n_1) = f \wedge \hat{a}$ and $\varphi(n_2) = f \wedge \neg\hat{a}$.

Parameter update is done sequentially on only the most recently added compound features, which correspond to the leaves of the new decision tree. After the entire tree is built, we percolate examples down to their appropriate leaf nodes. We then choose for each leaf node n the parameter $\Theta_{\varphi(n)}$ that minimizes the objective over the examples in that leaf. A convenient property of decision trees is that the leaves' compound features are mutually exclusive. Their parameters can be directly optimized independently of each other using a line search over the objective.

3.3 The Training Set

We choose a single correct path from each training parse tree, and the training examples correspond to all candidate inferences considered in every state along this path.[4] In the deterministic setting there is only one correct path, so example generation is identical to that of Sagae and Lavie (2005). If parsing proceeds non-deterministically then there might be multiple paths that lead to the same final parse, so we choose one randomly. This method of generating training examples does not require a working parser and can be run prior to any training. The disadvantage of this approach is that it minimizes the error of the parser at *correct* states only. It does not account for compounded error or teach the parser to recover from mistakes gracefully.

Turian and Melamed (2005) observed that uniform example biases $b(i)$ produced lower accuracy as training progressed, because the induced classifiers minimized the error per *example*. To minimize the error per *state*, we assign every training state equal value and share half the value uniformly among the negative examples for the examples generated from that state and the other half uniformly among the positive examples.

We parallelize training by inducing 26 label classifiers (one for each non-terminal label in the Penn Treebank). Parallelization might not uniformly reduce training time because different label classifiers train at different rates. However, parallelization uniformly reduces *memory* usage because each label classifier trains only on inferences whose consequent item has that label.

4 Experiments

Discriminative parsers are notoriously slow to train. For example, Taskar et al. (2004) took several months to train on the $\leq$ 15 word sentences in the English Penn Treebank (Dan Klein, p.c.). The present work makes progress towards faster discriminative parser training: our slowest classifier took fewer than 5 days to train. Even so, it would have taken much longer to train on the entire treebank. We follow Taskar et al. (2004) in training and testing on $\leq$ 15 word sentences in the English Penn Treebank (Taylor et al., 2003). We used sections 02–21 for training, section 22 for development, and section 23 for testing, preprocessed as per Table 1. We evaluated our parser using the standard PARSEVAL measures (Black et al., 1991): labelled precision, labelled recall, and labelled F-measure (Prec., Rec., and F_1, respectively), which are based on the number of non-terminal items in the parser's output that match those in the gold-standard parse.[5]

As mentioned in Section 2, items are inferred bottom-up and the parser cannot infer any item that crosses an item already in the state. Although there are $O(n^2)$ possible (span, label) pairs over a frontier containing n items, we reduce this to the $\approx 5 \cdot n$ inferences that have at most five children.[6]

[4] Nearly all of the examples generated are negative ($y = -1$).

[5] The correctness of a stratified shuffling test has been called into question (Michael Collins, p.c.), so we are not aware of any valid significance tests for observed differences in PARSEVAL scores.

[6] Only 0.57% of non-terminals in the preprocessed develop-

Table 1 Steps for preprocessing the data. Starred steps are performed only when parse trees are available in the data (e.g. not on test data).

1. * Strip functional tags and trace indices, and remove traces.
2. * Convert PRT to ADVP. (This convention was established by Magerman (1995).)
3. Remove quotation marks (i.e. terminal items tagged ‘ ‘ or ’ ’). (Bikel, 2004)
4. * Raise punctuation. (Bikel, 2004)
5. Remove outermost punctuation.[a]
6. * Remove unary projections to self (i.e. duplicate items with the same span and label).
7. POS tag the text using the tagger of Ratnaparkhi (1996).
8. Lowercase headwords.

[a] As pointed out by an anonymous reviewer of Collins (2003), removing outermost punctuation might discard useful information. Collins and Roark (2004) saw a LFMS improvement of 0.8% over their baseline discriminative parser after adding punctuation features, one of which encoded the sentence-final punctuation.

To ensure the parser does not enter an infinite loop, no two items in a state can have both the same span and the same label. Given these restrictions on candidate inferences, there were roughly 40 million training examples generated in the training set. These were partitioned among the 26 constituent label classifiers. Building a decision tree (Steps 1.5–1.9 in Listing 1) using the entire example set I can be very expensive. We estimate loss gradients (Equation 13) using a sample of the inference set, which gives a 100-fold increase in training speed (Turian & Melamed, 2006).

Our atomic feature set A contains 300K features, each of the form “is there an item in group J whose label/headword/headtag/headtagclass is ‘X’?”.[7] Possible values of ‘X’ for each predicate are collected from the training data. For $1 \leq n \leq 3$, possible values for J are:

- the first/last n child items
- the first n left/right context items
- the n children items left/right of the head
- the head item.

The left and right **context** items are the frontier items to the left and right of the children of the candidate inference, respectively.

4.1 Different Parsing Strategies

To demonstrate the flexibility of our learning procedure, we trained three different parsers: left-to-right (l2r), right-to-left (r2l), and non-deterministic bottom-up (b.u.). The non-deterministic parser was allowed to choose any bottom-up inference. The other two parsers were deterministic: bottom-up inferences had to be performed strictly left-to-right or right-to-left, respectively. We stopped training when each parser had 15K active features. Figure 1 shows the accuracy of the different runs over the development set as training progressed. Table 2 gives the PARSEVAL scores of these parsers at their optimal ℓ_1 penalty setting. We found that the perplexity of the r2l model was low so that, in 85% of the sentences, its greedy parse was the optimal one. The l2r parser does poorly because its decisions were more difficult than those of the other parsers. If it inferred far-right items, it was more likely to prevent correct subsequent inferences that were to the left. But if it inferred far-left items, then it went against the right-branching tendency of English sentences. The left-to-right parser would likely improve if we were to use a left-corner transform (Collins & Roark, 2004).

Table 2 Results on the development set, training and testing using only ≤ 15 word sentences.

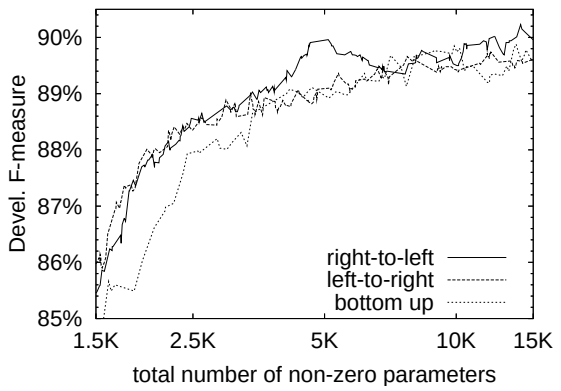
	λ	active features	% Rec.	% Prec.	F_1
l2r	0.040	11.9K	89.86	89.63	89.74
b.u.	0.020	13.7K	89.92	89.84	89.88
r2l	0.014	14.0K	90.66	89.81	90.23

Parsers in the literature typically choose some local threshold on the amount of search, such as a maximum beam width. With an accurate scoring function, restricting the search space using a fixed beam width might be unnecessary. Instead, we imposed a *global* threshold on exploration of the search space. Specifically, if the

ment set have more than five children.

[7] The predicate headtagclass is a supertype of the headtag. Given our compound features, these are not strictly necessary, but they accelerate training. An example is “proper noun,” which contains the POS tags given to singular and plural proper nouns. Space constraints prevent enumeration of the headtagclasses, which are instead provided at the URL given in the abstract.

Figure 1 F_1 scores on the development set of the Penn Treebank, using only ≤ 15 word sentences. The *x*-axis shows the number of non-zero parameters in each parser, summed over all classifiers.



parser has found some complete parse and has explored at least 100K states (i.e. scored at least 100K inferences), search stopped prematurely and the parser would return the (possibly sub-optimal) current best complete parse. The l2r and r2l parsers never exceeded this threshold, and always found the optimal complete parse. However, the non-deterministic bottom-up parser's search was cut-short in 28% of the sentences. The non-deterministic parser can reach each parse state through many different paths, so it searches a larger space than a deterministic parser, with more redundancy.

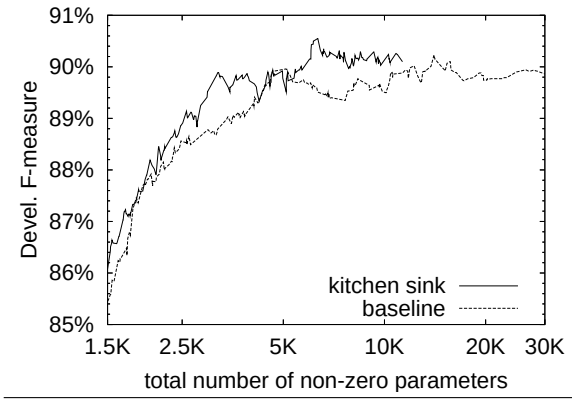
To gain a better understanding of the weaknesses of our parser, we examined a sample of 50 development sentences that the r2l parser did not get entirely correct. Roughly half the errors were due to noise and genuine ambiguity. The remaining errors fell into three types, occurring with roughly the same frequency:

• **`ADVPs and ADJPs`** The r2l parser had F_1 = 81.1% on `ADVP`s, and F_1 = 71.3% on `ADJP`s. Annotation of `ADJP` and `ADVP` in the PTB is inconsistent, particularly for *unary* projections.

• **POS Tagging Errors** Many of the parser's errors were due to incorrect POS tags. In future work we will integrate POS-tagging as inferences of the parser, allowing it to entertain competing hypotheses about the correct tagging.

• **Bilexical dependencies** Although compound features exist to detect affinities between words, the parser had difficulties with bilexical dependency decisions that were unobserved in the training data. The classifier would need more training data to learn these affinities.

Figure 2 F_1 scores of right-to-left parsers with different atomic feature sets on the development set of the Penn Treebank, using only ≤ 15 word sentences.



4.2 More Atomic Features

We compared our right-to-left parser with the baseline set of atomic features to one with a far richer atomic feature set, including *unbounded* context features, length features, and features of the terminal items. This "kitchen sink" parser merely has access to many more item groups *J*, described in Table 3. All features are all of the form given earlier, except for length features (Eisner & Smith, 2005). Length features compute the size of one of the groups of items in the indented list in Table 3. The feature determines if this length is equal to/greater than to $n, 0 \leq n \leq 15$. The kitchen sink parser had 1.1 million atomic features, 3.7 times the number available in the baseline. In future work, we plan to try linguistically more sophisticated features (Charniak & Johnson, 2005) as well as sub-tree features (Bod, 2003; Kudo et al., 2005).

Figure 2 shows the accuracy of the right-to-left parsers with different atomic feature sets over the development set as training progressed. Even though the baseline training made progress more quickly than the kitchen sink, the kitchen sink's F_1 surpassed the baseline's F_1 early in training, and at 6.3K active parameters it achieved a development set F_1 of 90.55%.

4.3 Test Set Results

To situate our results in the literature, we compare our results to those reported by Taskar et al. (2004) and Turian and Melamed (2005) for their discriminative parsers, which were also trained and tested on ≤ 15 word sentences. We also compare our parser to a representative non-discriminative

Table 3 Item groups available in the kitchen sink run.

- the first/last n child items, $1 \leq n \leq 4$
- the first n left/right context items, $1 \leq n \leq 4$
- the n children items left/right of the head, $1 \leq n \leq 4$
- the nth frontier item left/right of the leftmost/head/rightmost child item, $1 \leq n \leq 3$
- the nth terminal item left/right of the leftmost/head/rightmost terminal item dominated by the item being inferred, $1 \leq n \leq 3$
- the leftmost/head/rightmost child item of the leftmost/head/rightmost child item
- the following groups of frontier items:
  - all items
  - left/right context items
  - non-leftmost/non-head/non-rightmost child items
  - child items left/right of the head item, inclusive/exclusive
- the terminal items dominated by one of the item groups in the indented list above

Table 4 Results of parsers on the test set, training and testing using only ≤ 15 word sentences.

	% Rec.	% Prec.	F_1
Turian and Melamed (2005)	86.47	87.80	87.13
Bikel (2004)	87.85	88.75	88.30
Taskar et al. (2004)	89.10	89.14	89.12
kitchen sink	**89.26**	**89.55**	**89.40**

parser (Bikel, 2004)[8], the only one that we were able to train and test under exactly the same experimental conditions (including the use of POS tags from the tagger of Ratnaparkhi (1996)). Table 4 shows the PARSEVAL results of these four parsers on the test set.

5 Comparison with Related Work

Our parsing approach is based upon a single end-to-end discriminative learning machine. Collins and Roark (2004) and Taskar et al. (2004) beat the generative baseline only after using the standard trick of using the output from a generative model as a feature. Henderson (2004) finds that discriminative training was too slow, and reports accuracy higher than generative models by discriminatively reranking the output of his generative model. Unlike these state-of-the-art discriminative parsers, our method does not (yet) use any information from a generative model to improve training speed or accuracy. As far as we know, we present the first discriminative parser that does not use information from a generative model to beat a generative baseline (the Collins model).

The main limitation of our work is that we can do training reasonably quickly only on short sentences because a sentence with n words generates $O(n^2)$ training inferences in total. Although generating training examples in advance without a working parser (Turian & Melamed, 2005) is much faster than using inference (Collins & Roark, 2004; Henderson, 2004; Taskar et al., 2004), our training time can probably be decreased further by choosing a parsing strategy with a lower branching factor. Like our work, Ratnaparkhi (1999) and Sagae and Lavie (2005) generate examples off-line, but their parsing strategies are essentially shift-reduce so each sentence generates only $O(n)$ training examples.

An advantage of our approach is its flexibility. As our experiments showed, it is quite simple to substitute in different parsing strategies. Although we used very little linguistic information (the head rules and the POS tag classes), our model could also start with more sophisticated task-specific features in its atomic feature set. Atomic features that access arbitrary information are represented directly without the need for an induced intermediate representation (cf. Henderson, 2004).

Other papers (Clark & Curran, 2004; Kaplan et al., 2004, e.g.) have applied log-linear models to parsing. These works are based upon conditional models, which include a normalization term. However, our loss function forgoes normalization, which means that it is easily decomposed into the loss of individual inferences (Equation 5).

[8] Bikel (2004) is a "clean room" reimplementation of the Collins (1999) model with comparable accuracy.

Decomposition of the loss allows the objective to be optimized in parallel. This might be an advantage for larger structured prediction problems where there are more opportunities for parallelization, for example machine translation.

The only important hyper-parameter in our method is the ℓ_1 penalty factor. We optimize it as part of the training process, choosing the value that maximizes accuracy on a held-out development set. This technique stands in contrast to more ad-hoc methods for choosing hyper-parameters, which may require prior knowledge or additional experimentation.

6 Conclusion

Our work has made advances in both accuracy and training speed of discriminative parsing. As far as we know, we present the first discriminative parser that surpasses a generative baseline on constituent parsing without using a generative component, and it does so with minimal linguistic cleverness. Our approach performs feature selection incrementally over an exponential feature space during training. Our experiments suggest that the learning algorithm is overfitting-resistant, as hypothesized by Ng (2004). If this is the case, it would reduce the effort required for feature engineering. An engineer can merely design a set of atomic features whose powerset contains the requisite information. Then, the learning algorithm can perform feature selection over the compound feature space, avoiding irrelevant compound features.

In future work, we shall make some standard improvements. Our parser should infer its own POS tags to improve accuracy. A shift-reduce parsing strategy will generate fewer training inferences, and might lead to shorter training times. Lastly, we plan to give the model linguistically more sophisticated features. We also hope to apply the model to other structured prediction tasks, such as syntax-driven machine translation.

Acknowledgments

The authors would like to thank Chris Pike, Cynthia Rudin, and Ben Wellington, as well as the anonymous reviewers, for their helpful comments and constructive criticism. This research was sponsored by NSF grants #0238406 and #0415933.

References

Bikel, D. M. (2004). Intricacies of Collins' parsing model. *Computational Linguistics*, *30*(4).

Black, E., Abney, S., Flickenger, D., Gdaniec, C., Grishman, R., Harrison, P., et al. (1991). A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Speech and Natural Language.*

Bod, R. (2003). An efficient implementation of a new DOP model. In *EACL.*

Charniak, E., & Johnson, M. (2005). Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *ACL.*

Clark, S., & Curran, J. R. (2004). Parsing the WSJ using CCG and log-linear models. In *ACL.*

Collins, M. (1999). *Head-driven statistical models for natural language parsing*. Doctoral dissertation.

Collins, M. (2003). Head-driven statistical models for natural language parsing. *Computational Linguistics*, *29*(4).

Collins, M., & Roark, B. (2004). Incremental parsing with the perceptron algorithm. In *ACL.*

Collins, M., Schapire, R. E., & Singer, Y. (2002). Logistic regression, AdaBoost and Bregman distances. *Machine Learning*, *48*(1-3).

Eisner, J., & Smith, N. A. (2005). Parsing with soft and hard constraints on dependency length. In *IWPT.*

Henderson, J. (2004). Discriminative training of a neural network statistical parser. In *ACL.*

Kaplan, R. M., Riezler, S., King, T. H., Maxwell, III, J. T., Vasserman, A., & Crouch, R. (2004). Speed and accuracy in shallow and deep stochastic parsing. In *HLT/NAACL.*

Kudo, T., Suzuki, J., & Isozaki, H. (2005). Boosting-based parse reranking with subtree features. In *ACL.*

Magerman, D. M. (1995). Statistical decision-tree models for parsing. In *ACL.*

Ng, A. Y. (2004). Feature selection, ℓ_1 vs. ℓ_2 regularization, and rotational invariance. In *ICML.*

Perkins, S., Lacker, K., & Theiler, J. (2003). Grafting: Fast, incremental feature selection by gradient descent in function space. *Journal of Machine Learning Research*, *3*.

Ratnaparkhi, A. (1996). A maximum entropy part-of-speech tagger. In *EMNLP.*

Ratnaparkhi, A. (1999). Learning to parse natural language with maximum entropy models. *Machine Learning*, *34*(1-3).

Russell, S., & Norvig, P. (1995). *Artificial intelligence: A modern approach*.

Sagae, K., & Lavie, A. (2005). A classifier-based parser with linear run-time complexity. In *IWPT.*

Schapire, R. E., & Singer, Y. (1999). Improved boosting using confidence-rated predictions. *Machine Learning*, *37*(3).

Taskar, B., Klein, D., Collins, M., Koller, D., & Manning, C. (2004). Max-margin parsing. In *EMNLP.*

Taylor, A., Marcus, M., & Santorini, B. (2003). The Penn Treebank: an overview. In A. Abeillé (Ed.), *Treebanks: Building and using parsed corpora* (chap. 1).

Turian, J., & Melamed, I. D. (2005). Constituent parsing by classification. In *IWPT.*

Turian, J., & Melamed, I. D. (2006). Computational challenges in parsing by classification. In *HLT-NAACL workshop on computationally hard problems and joint inference in speech and language processing.*