Incremental Parsing with the Perceptron Algorithm

Michael Collins
MIT CSAIL
mcollins@csail.mit.edu

Brian Roark
AT&T Labs - Research
roark@research.att.com

Abstract

This paper describes an incremental parsing approach where parameters are estimated using a variant of the perceptron algorithm. A beam-search algorithm is used during both training and decoding phases of the method. The perceptron approach was implemented with the same feature set as that of an existing generative model (Roark, 2001a), and experimental results show that it gives competitive performance to the generative model on parsing the Penn treebank. We demonstrate that training a perceptron model to combine with the generative model during search provides a 2.1 percent F-measure improvement over the generative model alone, to 88.8 percent.

1 Introduction

In statistical approaches to NLP problems such as tagging or parsing, it seems clear that the representation used as input to a learning algorithm is central to the accuracy of an approach. In an ideal world, the designer of a parser or tagger would be free to choose any features which might be useful in discriminating good from bad structures, without concerns about how the features interact with the problems of training (parameter estimation) or decoding (search for the most plausible candidate under the model). To this end, a number of recently proposed methods allow a model to incorporate "arbitrary" global features of candidate analyses or parses. Examples of such techniques are Markov Random Fields (Ratnaparkhi et al., 1994; Abney, 1997; Della Pietra et al., 1997; Johnson et al., 1999), and boosting or perceptron approaches to reranking (Freund et al., 1998; Collins, 2000; Collins and Duffy, 2002).

A drawback of these approaches is that in the general case, they can require exhaustive enumeration of the set of candidates for each input sentence in both the training and decoding phases[1]. For example, Johnson et al. (1999) and Riezler et al. (2002) use all parses generated by an LFG parser as input to an MRF approach – given the level of ambiguity in natural language, this set can presumably become extremely large. Collins (2000) and Collins and Duffy (2002) rerank the top N parses from an existing generative parser, but this kind of approach presupposes that there is an existing baseline model with reasonable performance. Many of these baseline models are themselves used with heuristic search techniques, so that the potential gain through the use of discriminative re-ranking techniques is further dependent on effective search.

This paper explores an alternative approach to parsing, based on the perceptron training algorithm introduced in Collins (2002). In this approach the training and decoding problems are very closely related – the training method decodes training examples in sequence, and makes simple corrective updates to the parameters when errors are made. Thus the main complexity of the method is isolated to the decoding problem. We describe an approach that uses an incremental, left-to-right parser, with beam search, to find the highest scoring analysis under the model. The same search method is used in both training and decoding. We implemented the perceptron approach with the same feature set as that of an existing generative model (Roark, 2001a), and show that the perceptron model gives performance competitive to that of the generative model on parsing the Penn treebank, thus demonstrating that an unnormalized discriminative parsing model can be applied with heuristic search. We also describe several refinements to the training algorithm, and demonstrate their impact on convergence properties of the method.

Finally, we describe training the perceptron model with the negative log probability given by the generative model as another feature. This provides the perceptron algorithm with a better starting point, leading to large improvements over using either the generative model or the perceptron algorithm in isolation (the hybrid model achieves 88.8% f-measure on the WSJ treebank, compared to figures of 86.7% and 86.6% for the separate generative and perceptron models). The approach is an extremely simple method for integrating new features into the generative model: essentially all that is needed is a definition of feature-vector representations of entire parse trees, and then the existing parsing algorithms can be used for both training and decoding with the models.

2 The General Framework

In this section we describe a general framework – linear models for NLP – that could be applied to a diverse range of tasks, including parsing and tagging. We then describe a particular method for parameter estimation, which is a generalization of the perceptron algorithm. Finally, we

[1] Dynamic programming methods (Geman and Johnson, 2002; Lafferty et al., 2001) can sometimes be used for both training and decoding, but this requires fairly strong restrictions on the features in the model.

give an abstract description of an incremental parser, and describe how it can be used with the perceptron algorithm.

2.1 Linear Models for NLP

We follow the framework outlined in Collins (2002; 2004). The task is to learn a mapping from inputs $x \in \mathcal{X}$ to outputs $y \in \mathcal{Y}$. For example, $\mathcal{X}$ might be a set of sentences, with $\mathcal{Y}$ being a set of possible parse trees. We assume:

- Training examples (x_i, y_i) for $i = 1 \ldots n$.
- A function $\mathbf{GEN}$ which enumerates a set of candidates $\mathbf{GEN}(x)$ for an input x.
- A **representation** Φ mapping each $(x, y) \in \mathcal{X} \times \mathcal{Y}$ to a feature vector $\Phi(x, y) \in \mathbb{R}^d$.
- A **parameter vector** $\bar{\alpha} \in \mathbb{R}^d$.

The components $\mathbf{GEN}$, Φ and $\bar{\alpha}$ define a mapping from an input x to an output $F(x)$ through

$$F(x) = \arg \max_{y \in \mathbf{GEN}(x)} \Phi(x, y) \cdot \bar{\alpha} \qquad (1)$$

where $\Phi(x, y) \cdot \bar{\alpha}$ is the inner product $\sum_s \alpha_s \Phi_s(x, y)$. The learning task is to set the parameter values $\bar{\alpha}$ using the training examples as evidence. The *decoding algorithm* is a method for searching for the $\arg\max$ in Eq. 1.

This framework is general enough to encompass several tasks in NLP. In this paper we are interested in parsing, where (x_i, y_i), $\mathbf{GEN}$, and Φ can be defined as follows:

- Each training example (x_i, y_i) is a pair where x_i is a sentence, and y_i is the gold-standard parse for that sentence.
- Given an input sentence x, $\mathbf{GEN}(x)$ is a set of possible parses for that sentence. For example, $\mathbf{GEN}(x)$ could be defined as the set of possible parses for x under some context-free grammar, perhaps a context-free grammar induced from the training examples.
- The representation $\Phi(x, y)$ could track arbitrary features of parse trees. As one example, suppose that there are m rules in a context-free grammar (CFG) that defines $\mathbf{GEN}(x)$. Then we could define the i'th component of the representation, $\Phi_i(x, y)$, to be the number of times the i'th context-free rule appears in the parse tree (x, y). This is implicitly the representation used in probabilistic or weighted CFGs.

Note that the difficulty of finding the $\arg\max$ in Eq. 1 is dependent on the interaction of $\mathbf{GEN}$ and Φ. In many cases $\mathbf{GEN}(x)$ could grow exponentially with the size of x, making brute force enumeration of the members of $\mathbf{GEN}(x)$ intractable. For example, a context-free grammar could easily produce an exponentially growing number of analyses with sentence length. For some representations, such as the "rule-based" representation described above, the $\arg\max$ in the set enumerated by the CFG can be found efficiently, using dynamic programming algorithms, without having to explicitly enumerate all members of $\mathbf{GEN}(x)$. However in many cases we may be interested in representations which do not allow efficient dynamic programming solutions. One way around this problem is to adopt a two-pass approach, where $\mathbf{GEN}(x)$ is the top N analyses under some initial model, as in the reranking approach of Collins (2000). In the current paper we explore alternatives to reranking approaches, namely heuristic methods for finding the $\arg\max$, specifically incremental beam-search strategies related to the parsers of Roark (2001a) and Ratnaparkhi (1999).

2.2 The Perceptron Algorithm for Parameter Estimation

We now consider the problem of setting the parameters, $\bar{\alpha}$, given training examples (x_i, y_i). We will briefly review the perceptron algorithm, and its convergence properties – see Collins (2002) for a full description. The algorithm and theorems are based on the approach to classification problems described in Freund and Schapire (1999).

Figure 1 shows the algorithm. Note that the most complex step of the method is finding $z_i = \arg\max_{z \in \mathbf{GEN}(x_i)} \Phi(x_i, z) \cdot \bar{\alpha}$ – and this is precisely the decoding problem. Thus the training algorithm is in principle a simple part of the parser: any system will need a decoding method, and once the decoding algorithm is implemented the training algorithm is relatively straightforward.

We will now give a first theorem regarding the convergence of this algorithm. First, we need the following definition:

Definition 1 *Let* $\overline{\mathbf{GEN}}(x_i) = \mathbf{GEN}(x_i) - \{y_i\}$. *In other words* $\overline{\mathbf{GEN}}(x_i)$ *is the set of* incorrect *candidates for an example* x_i. *We will say that a training sequence* (x_i, y_i) *for* $i = 1 \ldots n$ *is* **separable with margin** $\delta > 0$ *if there exists some vector* $\mathbf{U}$ *with* $||\mathbf{U}|| = 1$ *such that*

$$\forall i, \forall z \in \overline{\mathbf{GEN}}(x_i), \quad \mathbf{U} \cdot \Phi(x_i, y_i) - \mathbf{U} \cdot \Phi(x_i, z) \geq \delta \qquad (2)$$

($||\mathbf{U}||$ is the 2-norm of $\mathbf{U}$*, i.e.,* $||\mathbf{U}|| = \sqrt{\sum_s \mathbf{U}_s^2}$*.)*

Next, define N_e to be the number of times an error is made by the algorithm in figure 1 – that is, the number of times that $z_i \neq y_i$ for some (t, i) pair. We can then state the following theorem (see (Collins, 2002) for a proof):

Theorem 1 *For any training sequence* (x_i, y_i) *that is separable with margin* δ*, for any value of* T*, then for the perceptron algorithm in figure 1*

$$N_e \leq \frac{R^2}{\delta^2}$$

where R *is a constant such that* $\forall i, \forall z \in \overline{\mathbf{GEN}}(x_i) \quad ||\Phi(x_i, y_i) - \Phi(x_i, z)|| \leq R$.

This theorem implies that if there is a parameter vector $\mathbf{U}$ which makes zero errors on the training set, then after a finite number of iterations the training algorithm will converge to parameter values with zero training error. A crucial point is that the number of mistakes is independent of the number of candidates for each example

Inputs: Training examples (x_i, y_i)
Initialization: Set $\bar{\alpha} = 0$
Output: Parameters $\bar{\alpha}$

Algorithm:
For $t = 1 \ldots T, i = 1 \ldots n$
Calculate $z_i = \arg\max_{z \in \mathbf{GEN}(x_i)} \Phi(x_i, z) \cdot \bar{\alpha}$
If$(z_i \neq y_i)$ then $\bar{\alpha} = \bar{\alpha} + \Phi(x_i, y_i) - \Phi(x_i, z_i)$

Figure 1: A variant of the perceptron algorithm.

(i.e. the size of $\mathbf{GEN}(x_i)$ for each i), depending only on the separation of the training data, where separation is defined above. This is important because in many NLP problems $\mathbf{GEN}(x)$ can be exponential in the size of the inputs. All of the convergence and generalization results in Collins (2002) depend on notions of separability rather than the size of $\mathbf{GEN}$.

Two questions come to mind. First, are there guarantees for the algorithm if the training data is not separable? Second, performance on a training sample is all very well, but what does this guarantee about how well the algorithm generalizes to newly drawn test examples? Freund and Schapire (1999) discuss how the theory for classification problems can be extended to deal with both of these questions; Collins (2002) describes how these results apply to NLP problems.

As a final note, following Collins (2002), we used the *averaged* parameters from the training algorithm in decoding test examples in our experiments. Say $\bar{\alpha}_i^t$ is the parameter vector after the i'th example is processed on the t'th pass through the data in the algorithm in figure 1. Then the averaged parameters $\bar{\alpha}_{AVG}$ are defined as $\bar{\alpha}_{AVG} = \sum_{i,t} \bar{\alpha}_i^t / NT$. Freund and Schapire (1999) originally proposed the averaged parameter method; it was shown to give substantial improvements in accuracy for tagging tasks in Collins (2002).

2.3 An Abstract Description of Incremental Parsing

This section gives a description of the basic incremental parsing approach. The input to the parser is a sentence x with length n. A *hypothesis* is a triple $\langle x, t, i \rangle$ such that x is the sentence being parsed, t is a partial or full analysis of that sentence, and i is an integer specifying the number of words of the sentence which have been processed. Each full parse for a sentence will have the form $\langle x, t, n \rangle$. The initial state is $\langle x, \emptyset, 0 \rangle$ where $\emptyset$ is a "null" or empty analysis.

We assume an "advance" function ADV which takes a hypothesis triple as input, and returns a *set* of new hypotheses as output. The advance function will absorb another word in the sentence: this means that if the input to ADV is $\langle x, t, i \rangle$, then each member of ADV$(\langle x, t, i \rangle)$ will have the form $\langle x, t', i{+}1 \rangle$. Each new analysis t' will be formed by somehow incorporating the $i{+}1$'th word into the previous analysis t.

With these definitions in place, we can iteratively define the full set of partial analyses $\mathcal{H}_i$ for the first i words of the sentence as $\mathcal{H}_0(x) = \{\langle x, \emptyset, 0 \rangle\}$, and $\mathcal{H}_i(x) = \cup_{h' \in \mathcal{H}_{i-1}(x)} \text{ADV}(h')$ for $i = 1 \ldots n$. The full set of parses for a sentence x is then $\mathbf{GEN}(x) = \mathcal{H}_n(x)$ where n is the length of x.

Under this definition $\mathbf{GEN}(x)$ can include a huge number of parses, and searching for the highest scoring parse, $\arg\max_{h \in \mathcal{H}_n(x)} \Phi(h) \cdot \bar{\alpha}$, will be intractable. For this reason we introduce one additional function, FILTER$(\mathcal{H})$, which takes a set of hypotheses $\mathcal{H}$, and returns a much smaller set of "filtered" hypotheses. Typically, FILTER will calculate the score $\Phi(h) \cdot \bar{\alpha}$ for each $h \in \mathcal{H}$, and then eliminate partial analyses which have low scores under this criterion. For example, a simple version of FILTER would take the top N highest scoring members of $\mathcal{H}$ for some constant N. We can then redefine the set of partial analyses as follows (we use $\mathcal{F}_i(x)$ to denote the set of filtered partial analyses for the first i words of the sentence):

$$\mathcal{F}_0(x) = \{\langle x, \emptyset, 0 \rangle\}$$
$$\mathcal{F}_i(x) = \text{FILTER}\left(\cup_{h' \in \mathcal{F}_{i-1}(x)} \text{ADV}(h')\right) \text{ for } i{=}1 \ldots n$$

The parsing algorithm returns $\arg\max_{h \in \mathcal{F}_n} \Phi(h) \cdot \bar{\alpha}$. Note that this is a heuristic, in that there is no guarantee that this procedure will find the highest scoring parse, $\arg\max_{h \in \mathcal{H}_n} \Phi(h) \cdot \bar{\alpha}$. Search errors, where $\arg\max_{h \in \mathcal{F}_n} \Phi(h) \cdot \bar{\alpha} \neq \arg\max_{h \in \mathcal{H}_n} \Phi(h) \cdot \bar{\alpha}$, will create errors in decoding test sentences, and also errors in implementing the perceptron training algorithm in Figure 1. In this paper we give empirical results that suggest that FILTER can be chosen in such a way as to give efficient parsing performance together with high parsing accuracy.

The exact implementation of the parser will depend on the definition of partial analyses, of ADV and FILTER, and of the representation Φ. The next section describes our instantiation of these choices.

3 A full description of the parsing approach

The parser is an incremental beam-search parser very similar to the sort described in Roark (2001a; 2004), with some changes in the search strategy to accommodate the perceptron feature weights. We first describe the parsing algorithm, and then move on to the baseline feature set for the perceptron model.

3.1 Parser control

The input to the parser is a string w_0^n, a grammar G, a mapping Φ from derivations to feature vectors, and a parameter vector $\bar{\alpha}$. The grammar $G = (V, T, \text{S}^\dagger, \bar{\text{S}}, C, B)$ consists of a set of non-terminal symbols V, a set of terminal symbols T, a start symbol $\text{S}^\dagger \in V$, an end-of-constituent symbol $\bar{\text{S}} \in V$, a set of "allowable chains" C, and a set of "allowable triples" B. $\bar{\text{S}}$ is a special empty non-terminal that marks the end of a constituent. Each chain is a sequence of non-terminals followed by a terminal symbol, for example $\langle \texttt{S}^\dagger \rightarrow \texttt{S} \rightarrow \texttt{NP} \rightarrow \texttt{NN} \rightarrow$

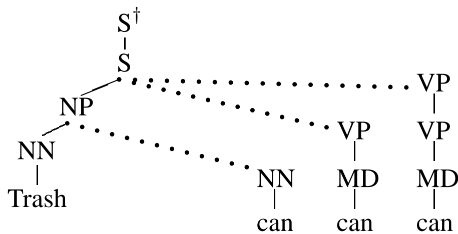

Figure 2: Left child chains and connection paths. Dotted lines represent potential attachments

Trash⟩. Each "allowable triple" is a tuple $\langle X, Y, Z\rangle$ where $X, Y, Z \in V$. The triples specify which non-terminals Z are allowed to follow a non-terminal Y under a parent X. For example, the triple ⟨S,NP,VP⟩ specifies that a VP can follow an NP under an S. The triple ⟨NP,NN,$\bar{\text{S}}$⟩ would specify that the $\bar{\text{S}}$ symbol can follow an NN under an NP – i.e., that the symbol NN is allowed to be the final child of a rule with parent NP

The initial state of the parser is the input string alone, w_0^n. In absorbing the first word, we add all chains of the form $\text{S}^\dagger \ldots \rightarrow w_0$. For example, in figure 2 the chain $\langle \text{S}^\dagger \rightarrow \text{S} \rightarrow \text{NP} \rightarrow \text{NN} \rightarrow \text{Trash}\rangle$ is used to construct an analysis for the first word alone. Other chains which start with $\text{S}^\dagger$ and end with Trash would give competing analyses for the first word of the string.

Figure 2 shows an example of how the next word in a sentence can be incorporated into a partial analysis for the previous words. For any partial analysis there will be a set of potential attachment sites: in the example, the attachment sites are under the NP or the S. There will also be a set of possible chains terminating in the next word – there are three in the example. Each chain could potentially be attached at each attachment site, giving 6 ways of incorporating the next word in the example. For illustration, assume that the set B is {⟨S,NP,VP⟩, ⟨NP,NN,NN⟩, ⟨NP,NN,$\bar{\text{S}}$⟩, ⟨S,NP,VP⟩}. Then some of the 6 possible attachments may be disallowed because they create triples that are not in the set B. For example, in figure 2 attaching either of the VP chains under the NP is disallowed because the triple ⟨NP,NN,VP⟩ is not in B. Similarly, attaching the NN chain under the S will be disallowed if the triple ⟨S,NP,NN⟩ is not in B. In contrast, adjoining ⟨NN → can⟩ under the NP creates a single triple, ⟨NP,NN,NN⟩, which is allowed. Adjoining either of the VP chains under the S creates two triples, ⟨S,NP,VP⟩ and ⟨NP,NN,$\bar{\text{S}}$⟩, which are both in the set B.

Note that the "allowable chains" in our grammar are what Costa et al. (2001) call "connection paths" from the partial parse to the next word. It can be shown that the method is equivalent to parsing with a transformed context-free grammar (a first-order "Markov" grammar) – for brevity we omit the details here.

In this way, given a set of candidates $\mathcal{F}_i(x)$ for the first i words of the string, we can generate a set of candidates for the first $i+1$ words, $\cup_{h' \in \mathcal{F}_i(x)} \text{ADV}(h')$, where the ADV function uses the grammar as described above. We then calculate $\Phi(h) \cdot \bar{\alpha}$ for all of these partial hypotheses, and rank the set from best to worst. A FILTER function is then applied to this ranked set to give $\mathcal{F}_{i+1}$. Let h_k be the kth ranked hypothesis in $\mathcal{H}_{i+1}(x)$. Then $h_k \in \mathcal{F}_{i+1}$ if and only if $\Phi(h_k) \cdot \bar{\alpha} \geq \theta_k$. In our case, we parameterize the calculation of θ_k with γ as follows:

$$\theta_k = \Phi(h_0) \cdot \bar{\alpha} - \frac{\gamma}{k^3}. \quad (3)$$

The problem with using left-child chains is limiting them in number. With a left-recursive grammar, of course, the set of all possible left-child chains is infinite. We use two techniques to reduce the number of left-child chains: first, we remove some (but not all) of the recursion from the grammar through a tree transform; next, we limit the left-child chains consisting of more than two non-terminal categories to those actually observed in the training data more than once. Left-child chains of length less than or equal to two are all those observed in training data. As a practical matter, the set of left-child chains for a terminal x is taken to be the union of the sets of left-child chains for all pre-terminal part-of-speech (POS) tags T for x.

Before inducing the left-child chains and allowable triples from the treebank, the trees are transformed with a selective left-corner transformation (Johnson and Roark, 2000) that has been flattened as presented in Roark (2001b). This transform is only applied to left-recursive productions, i.e. productions of the form $A \rightarrow A\gamma$. The transformed trees look as in figure 3. The transform has the benefit of dramatically reducing the number of left-child chains, without unduly disrupting the immediate dominance relationships that provide features for the model. The parse trees that are returned by the parser are then de-transformed to the original form of the grammar for evaluation[2].

Tree transform	POS tags	f24 Type	f2-21 Type	f2-21 OOV	f2-21, # > 1 Type	f2-21, # > 1 OOV
None	Gold	386	1680	0.1%	1013	0.1%
None	Tagged	401	1776	0.1%	1043	0.2%
FSLC	Gold	289	1214	0.1%	746	0.1%
FSLC	Tagged	300	1294	0.1%	781	0.1%

Table 1: Left-child chain type counts (of length > 2) for sections of the Wall St. Journal Treebank, and out-of-vocabulary (OOV) rate on the held-out corpus.

Table 1 presents the number of left-child chains of length greater than 2 in sections 2-21 and 24 of the Penn Wall St. Journal Treebank, both with and without the flattened selective left-corner transformation (FSLC), for gold-standard part-of-speech (POS) tags and automatically tagged POS tags. When the FSLC has been applied and the set is restricted to those occurring more than once

[2] See Johnson (1998) for a presentation of the transform/de-transform paradigm in parsing.

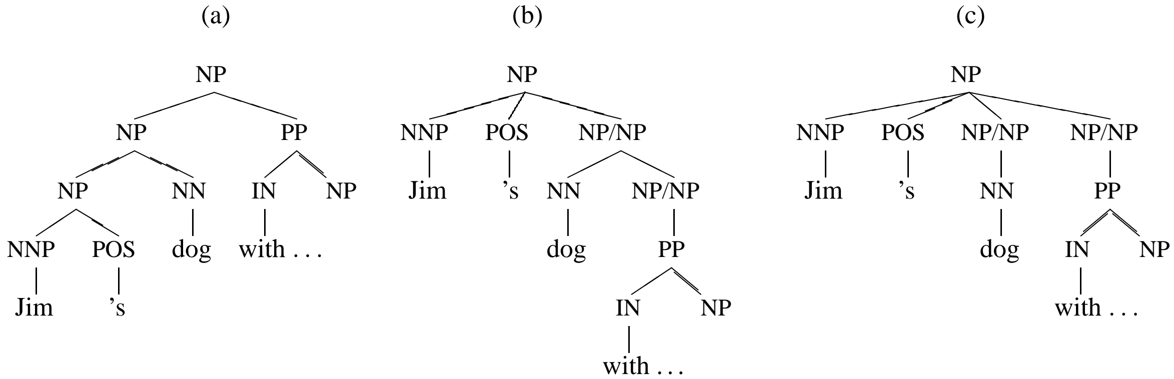

Figure 3: Three representations of NP modifications: (a) the original treebank representation; (b) Selective left-corner representation; and (c) a flat structure that is unambiguously equivalent to (b)

$F_0 = \{L_{00}, L_{10}\}$	$F_4 = F_3 \cup \{L_{03}\}$	$F_8 = F_7 \cup \{L_{21}\}$	$F_{12} = F_{11} \cup \{L_{11}\}$
$F_1 = F_0 \cup \{LKP\}$	$F_5 = F_4 \cup \{L_{20}\}$	$F_9 = F_8 \cup \{CL\}$	$F_{13} = F_{12} \cup \{L_{30}\}$
$F_2 = F_1 \cup \{L_{01}\}$	$F_6 = F_5 \cup \{L_{11}\}$	$F_{10} = F_9 \cup \{LK\}$	$F_{14} = F_{13} \cup \{CCP\}$
$F_3 = F_2 \cup \{L_{02}\}$	$F_7 = F_6 \cup \{L_{30}\}$	$F_{11} = F_0 \cup \{L_{20}\}$	$F_{15} = F_{14} \cup \{CC\}$

Table 2: Baseline feature set. Features $F_0 - F_{10}$ fire at non-terminal nodes. Features $F_0, F_{11} - F_{15}$ fire at terminal nodes.

in the training corpus, we can reduce the total number of left-child chains of length greater than 2 by half, while leaving the number of words in the held-out corpus with an unobserved left-child chain (out-of-vocabulary rate – OOV) to just one in every thousand words.

3.2 Features

For this paper, we wanted to compare the results of a perceptron model with a generative model for a comparable feature set. Unlike in Roark (2001a; 2004), there is no look-ahead statistic, so we modified the feature set from those papers to explicitly include the lexical item and POS tag of the next word. Otherwise the features are basically the same as in those papers. We then built a generative model with this feature set and the same tree transform, for use with the beam-search parser from Roark (2004) to compare against our baseline perceptron model.

To concisely present the baseline feature set, let us establish a notation. Features will fire whenever a new node is built in the tree. The features are labels from the left-context, i.e. the already built part of the tree. All of the labels that we will include in our feature sets are i levels above the current node in the tree, and j nodes to the left, which we will denote L_{ij}. Hence, L_{00} is the node label itself; L_{10} is the label of parent of the current node; L_{01} is the label of the sibling of the node, immediately to its left; L_{11} is the label of the sibling of the parent node, etc. We also include: the lexical head of the current constituent (CL); the c-commanding lexical head (CC) and its POS (CCP); and the look-ahead word (LK) and its POS (LKP). All of these features are discussed at more length in the citations above. Table 2 presents the baseline feature set.

In addition to the baseline feature set, we will also present results using features that would be more difficult to embed in a generative model. We included some punctuation-oriented features, which included (i) a Boolean feature indicating whether the final punctuation is a question mark or not; (ii) the POS label of the word after the current look-ahead, if the current look-ahead is punctuation or a coordinating conjunction; and (iii) a Boolean feature indicating whether the look-ahead is punctuation or not, that fires when the category immediately to the left of the current position is immediately preceded by punctuation.

4 Refinements to the Training Algorithm

This section describes two modifications to the “basic” training algorithm in figure 1.

4.1 Making Repeated Use of Hypotheses

Figure 4 shows a modified algorithm for parameter estimation. The input to the function is a gold standard parse, together with a set of candidates $\mathcal{F}$ generated by the incremental parser. There are two steps. First, the model is updated as usual with the current example, which is then added to a cache of examples. Second, the method repeatedly iterates over the cache, updating the model at each cached example if the gold standard parse is not the best scoring parse from among the stored candidates for that example. In our experiments, the cache was restricted to contain the parses from up to N previously processed sentences, where N was set to be the size of the training set.

The motivation for these changes is primarily efficiency. One way to think about the algorithms in this paper is as methods for finding parameter values that satisfy a set of linear constraints – one constraint for each incorrect parse in training data. The incremental parser is

Input: A gold-standard parse $= g$ for sentence k of N. A set of candidate parses $\mathcal{F}$. Current parameters $\bar{\alpha}$. A *Cache* of triples $\langle g_j, \mathcal{F}_j, c_j \rangle$ for $j = 1 \ldots N$ where each g_j is a previously generated gold standard parse, $\mathcal{F}_j$ is a previously generated set of candidate parses, and c_j is a counter of the number of times that $\bar{\alpha}$ has been updated due to this particular triple. Parameters T_1 and T_2 controlling the number of iterations below. In our experiments, $T_1 = 5$ and $T_2 = 50$. Initialize the *Cache* to include, for $j = 1 \ldots N$, $\langle g_j, \emptyset, T_2 \rangle$.

Step 1:

Calculate $z = \arg\max_{t \in \mathcal{F}} \Phi(t) \cdot \bar{\alpha}$
If $(z \neq g)$ then $\bar{\alpha} = \bar{\alpha} + \Phi(g) - \Phi(z)$
Set the kth triple in the *Cache* to $\langle g, \mathcal{F}, 0 \rangle$

Step 2:

For $t = 1 \ldots T_1, j = 1 \ldots N$
If $c_j < T_2$ then
Calculate $z = \arg\max_{t \in \mathcal{F}_j} \Phi(t) \cdot \bar{\alpha}$
If $(z \neq g_j)$ then
$\bar{\alpha} = \bar{\alpha} + \Phi(g_j) - \Phi(z)$
$c_j = c_j + 1$

Figure 4: The refined parameter update method makes repeated use of hypotheses

a method for dynamically generating constraints (i.e. incorrect parses) which are violated, or close to being violated, under the current parameter settings. The basic algorithm in Figure 1 is extremely wasteful with the generated constraints, in that it only looks at one constraint on each sentence (the $\arg\max$), and it ignores constraints implied by previously parsed sentences. This is inefficient because the generation of constraints (i.e., parsing an input sentence), is computationally quite demanding.

More formally, it can be shown that the algorithm in figure 4 also has the upper bound in theorem 1 on the number of parameter updates performed. If the cost of steps 1 and 2 of the method are negligible compared to the cost of parsing a sentence, then the refined algorithm will certainly converge no more slowly than the basic algorithm, and may well converge more quickly.

As a final note, we used the parameters T_1 and T_2 to limit the number of passes over examples, the aim being to prevent repeated updates based on outlier examples which are not separable.

4.2 Early Update During Training

As before, define y_i to be the gold standard parse for the i'th sentence, and also define y_i^j to be the *partial analysis* under the gold-standard parse for the first j words of the i'th sentence. Then if $y_i^j \notin \mathcal{F}_j(x_i)$ a search error has been made, and there is no possibility of the gold standard parse y_i being in the final set of parses, $\mathcal{F}_n(x_i)$. We call the following modification to the parsing algorithm during training "early update": if $y_i^j \notin \mathcal{F}_j(x_i)$, exit the parsing process, pass $y_i^j, \mathcal{F}_j(x_i)$ to the parameter estimation method, and move on to the next string in the training set. Intuitively, the motivation behind this is clear. It makes sense to make a correction to the parameter values at the point that a search error has been made, rather than allowing the parser to continue to the end of the sentence. This is likely to lead to less noisy input to the parameter estimation algorithm; and early update will also improve efficiency, as at the early stages of training the parser will frequently give up after a small proportion of each sentence is processed. It is more difficult to justify from a formal point of view, we leave this to future work.

Figure 5 shows the convergence of the training algorithm with neither of the two refinements presented; with just early update; and with both. Early update makes

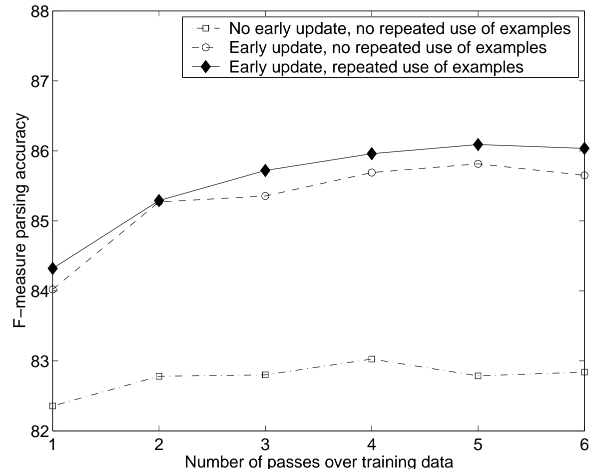

Figure 5: Performance on development data (section f24) after each pass over the training data, with and without repeated use of examples and early update.

an enormous difference in the quality of the resulting model; repeated use of examples gives a small improvement, mainly in recall.

5 Empirical results

The parsing models were trained and tested on treebanks from the Penn Wall St. Journal Treebank: sections 2-21 were kept training data; section 24 was held-out development data; and section 23 was for evaluation. After each pass over the training data, the averaged perceptron model was scored on the development data, and the best performing model was used for test evaluation. For this paper, we used POS tags that were provided either by the Treebank itself (gold standard tags) or by the perceptron POS tagger[3] presented in Collins (2002). The former gives us an upper bound on the improvement that we might expect if we integrated the POS tagging with the parsing.

[3]For trials when the generative or perceptron parser was given POS tagger output, the models were trained on POS tagged sections 2-21, which in both cases helped performance slightly.

Model	Gold-standard tags			POS-tagger tags		
	LP	LR	F	LP	LR	F
Generative	88.1	87.6	87.8	86.8	86.5	86.7
Perceptron (baseline)	87.5	86.9	87.2	86.2	85.5	85.8
Perceptron (w/ punctuation features)	88.1	87.6	87.8	87.0	86.3	86.6

Table 3: Parsing results, section 23, all sentences, including labeled precision (LP), labeled recall (LR), and F-measure

Table 3 shows results on section 23, when either gold-standard or POS-tagger tags are provided to the parser[4]. With the base features, the generative model outperforms the perceptron parser by between a half and one point, but with the additional punctuation features, the perceptron model matches the generative model performance.

Of course, using the generative model and using the perceptron algorithm are not necessarily mutually exclusive. Another training scenario would be to include the generative model score as another feature, with some weight in the linear model learned by the perceptron algorithm. This sort of scenario was used in Roark et al. (2004) for training an n-gram language model using the perceptron algorithm. We follow that paper in fixing the weight of the generative model, rather than learning the weight along the the weights of the other perceptron features. The value of the weight was empirically optimized on the held-out set by performing trials with several values. Our optimal value was 10.

In order to train this model, we had to provide generative model scores for strings in the training set. Of course, to be similar to the testing conditions, we cannot use the standard generative model trained on every sentence, since then the generative score would be from a model that had already seen that string in the training data. To control for this, we built ten generative models, each trained on 90 percent of the training data, and used each of the ten to score the remaining 10 percent that was not seen in that training set. For the held-out and testing conditions, we used the generative model trained on all of sections 2-21.

In table 4 we present the results of including the generative model score along with the other perceptron features, just for the run with POS-tagger tags. The generative model score (negative log probability) effectively provides a much better initial starting point for the perceptron algorithm. The resulting F-measure on section 23 is 2.1 percent higher than either the generative model or perceptron-trained model used in isolation.

6 Conclusions

In this paper we have presented a discriminative training approach, based on the perceptron algorithm with a couple of effective refinements, that provides a model capable of effective heuristic search over a very difficult search space. In such an approach, the unnormalized discriminative parsing model can be applied without either an external model to present it with candidates, or potentially expensive dynamic programming. When the training algorithm is provided the generative model scores as an additional feature, the resulting parser is quite competitive on this task. The improvement that was derived from the additional punctuation features demonstrates the flexibility of the approach in incorporating novel features in the model.

[4]When POS tagging is integrated directly into the generative parsing process, the baseline performance is 87.0. For comparison with the perceptron model, results are shown with pre-tagged input.

Model	POS-tagger tags		
	LP	LR	F
Generative baseline	86.8	86.5	86.7
Perceptron (w/ punctuation features)	87.0	86.3	86.6
Generative + Perceptron (w/ punct)	89.1	88.4	88.8

Table 4: Parsing results, section 23, all sentences, including labeled precision (LP), labeled recall (LR), and F-measure

Future research will look in two directions. First, we will look to include more useful features that are difficult for a generative model to include. This paper was intended to compare search with the generative model and the perceptron model with roughly similar feature sets. Much improvement could potentially be had by looking for other features that could improve the models. Secondly, combining with the generative model can be done in several ways. Some of the constraints on the search technique that were required in the absence of the generative model can be relaxed if the generative model score is included as another feature. In the current paper, the generative score was simply added as another feature. Another approach might be to use the generative model to produce candidates at a word, then assign perceptron features for those candidates. Such variants deserve investigation.

Overall, these results show much promise in the use of discriminative learning techniques such as the perceptron algorithm to help perform heuristic search in difficult domains such as statistical parsing.

Acknowledgements

The work by Michael Collins was supported by the National Science Foundation under Grant No. 0347631.

References

Steven Abney. 1997. Stochastic attribute-value grammars. *Computational Linguistics*, 23(4):597–617.

Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over dis-

crete structures and the voted perceptron. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 263–270.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *The Proceedings of the 17th International Conference on Machine Learning*.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–8.

Michael Collins. 2004. Parameter estimation for statistical parsing models: Theory and practice of distribution-free methods. In Harry Bunt, John Carroll, and Giorgio Satta, editors, *New Developments in Parsing Technology*. Kluwer.

Fabrizio Costa, Vincenzo Lombardo, Paolo Frasconi, and Giovanni Soda. 2001. Wide coverage incremental parsing by learning attachment preferences. In *Conference of the Italian Association for Artificial Intelligence (AIIA)*, pages 297–307.

Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. 1997. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:380–393.

Yoav Freund and Robert Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 3(37):277–296.

Yoav Freund, Raj Iyer, Robert Schapire, and Yoram Singer. 1998. An efficient boosting algorithm for combining preferences. In *Proc. of the 15th Intl. Conference on Machine Learning*.

Stuart Geman and Mark Johnson. 2002. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 279–286.

Mark Johnson and Brian Roark. 2000. Compact non-left-recursive grammars using the selective left-corner transform and factoring. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, pages 355–361.

Mark Johnson, Stuart Geman, Steven Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 535–541.

Mark Johnson. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):617–636.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289.

Adwait Ratnaparkhi, Salim Roukos, and R. Todd Ward. 1994. A maximum entropy model for parsing. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, pages 803–806.

Adwait Ratnaparkhi. 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34:151–175.

Stefan Riezler, Tracy King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell III, and Mark Johnson. 2002. Parsing the wall street journal using a lexical-functional grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 271–278.

Brian Roark, Murat Saraclar, and Michael Collins. 2004. Corrective language modeling for large vocabulary ASR with the perceptron algorithm. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 749–752.

Brian Roark. 2001a. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276.

Brian Roark. 2001b. *Robust Probabilistic Predictive Syntactic Processing*. Ph.D. thesis, Brown University. http://arXiv.org/abs/cs/0105019.

Brian Roark. 2004. Robust garden path parsing. *Natural Language Engineering*, 10(1):1–24.