Experiments with Spectral Learning of Latent-Variable PCFGs

Shay B. Cohen[1], **Karl Stratos**[1], **Michael Collins**[1], **Dean P. Foster**[2], **and Lyle Ungar**[3]
[1]Dept. of Computer Science, Columbia University
[2]Dept. of Statistics/[3]Dept. of Computer and Information Science, University of Pennsylvania
{scohen,stratos,mcollins}@cs.columbia.edu, foster@wharton.upenn.edu, ungar@cis.upenn.edu

Abstract

Latent-variable PCFGs (L-PCFGs) are a highly successful model for natural language parsing. Recent work (Cohen et al., 2012) has introduced a spectral algorithm for parameter estimation of L-PCFGs, which—unlike the EM algorithm—is guaranteed to give consistent parameter estimates (it has PAC-style guarantees of sample complexity). This paper describes experiments using the spectral algorithm. We show that the algorithm provides models with the same accuracy as EM, but is an order of magnitude more efficient. We describe a number of key steps used to obtain this level of performance; these should be relevant to other work on the application of spectral learning algorithms. We view our results as strong empirical evidence for the viability of spectral methods as an alternative to EM.

1 Introduction

Latent-variable PCFGS (L-PCFGs) are a highly successful model for natural language parsing (Matsuzaki et al., 2005; Petrov et al., 2006). Recent work (Cohen et al., 2012) has introduced a spectral learning algorithm for L-PCFGs. A crucial property of the algorithm is that it is guaranteed to provide consistent parameter estimates—in fact it has PAC-style guarantees of sample complexity.[1] This is in contrast to the EM algorithm, the usual method for parameter estimation in L-PCFGs, which has the weaker guarantee of reaching a local maximum of the likelihood function. The spectral algorithm is relatively simple and efficient, relying on a singular value decomposition of the training examples, followed by a single pass over the data where parameter values are calculated.

Cohen et al. (2012) describe the algorithm, and the theory behind it, but as yet no experimental results have been reported for the method. This paper describes experiments on natural language parsing using the spectral algorithm for parameter estimation. The algorithm provides models with slightly higher accuracy than EM (88.05% F-measure on test data for the spectral algorithm, vs 87.76% for EM), but is an order of magnitude more efficient (9h52m for training, compared to 187h12m, a speed-up of 19 times).

We describe a number of key steps in obtaining this level of performance. A simple backed-off smoothing method is used to estimate the large number of parameters in the model. The spectral algorithm requires functions mapping inside and outside trees to feature vectors—we make use of features corresponding to single level rules, and larger tree fragments composed of two or three levels of rules. We show that it is important to scale features by their inverse variance, in a manner that is closely related to methods used in canonical correlation analysis. Negative values can cause issues in spectral algorithms, but we describe a solution to these problems.

In recent work there has been a series of results in spectral learning algorithms for latent-variable models (Vempala and Wang, 2004; Hsu et al., 2009; Bailly et al., 2010; Siddiqi et al., 2010; Parikh et al., 2011; Balle et al., 2011; Arora et al., 2012; Dhillon et al., 2012; Anandkumar et al., 2012). Most of these results are theoretical (although see Luque et al. (2012) for empirical results of spectral learning for dependency parsing). While the focus of our experiments is on parsing, our findings should be relevant to the application of spectral methods to other latent-variable models. We view our results as strong empirical evidence for the viability of spectral methods as an alternative to EM.

2 Background

In this section we first give basic definitions for L-PCFGs, and then describe the spectral learning algorithm of Cohen et al. (2012).

[1]under assumptions on certain singular values in the model; see section 2.3.1.

2.1 L-PCFGs: Basic Definitions

We follow the definition in Cohen et al. (2012) of L-PCFGs. An L-PCFG is an 8-tuple $(\mathcal{N}, \mathcal{I}, \mathcal{P}, m, n, \pi, t, q)$ where:

- $\mathcal{N}$ is the set of non-terminal symbols in the grammar. $\mathcal{I} \subset \mathcal{N}$ is a finite set of *in-terminals*. $\mathcal{P} \subset \mathcal{N}$ is a finite set of *pre-terminals*. We assume that $\mathcal{N} = \mathcal{I} \cup \mathcal{P}$, and $\mathcal{I} \cap \mathcal{P} = \emptyset$. Hence we have partitioned the set of non-terminals into two subsets.
- $[m]$ is the set of possible hidden states.[2]
- $[n]$ is the set of possible words.
- For all $a \in \mathcal{I}$, $b, c \in \mathcal{N}$, $h_1, h_2, h_3 \in [m]$, we have a context-free rule $a(h_1) \rightarrow b(h_2)\ \ c(h_3)$. The rule has an associated parameter $t(a \rightarrow b\ c, h_2, h_3 | a, h_1)$.
- For all $a \in \mathcal{P}$, $h \in [m]$, $x \in [n]$, we have a context-free rule $a(h) \rightarrow x$. The rule has an associated parameter $q(a \rightarrow x | a, h)$.
- For all $a \in \mathcal{I}$, $h \in [m]$, $\pi(a, h)$ is a parameter specifying the probability of $a(h)$ being at the root of a tree.

A *skeletal tree* (s-tree) is a sequence of rules $r_1 \ldots r_N$ where each r_i is either of the form $a \rightarrow b\ c$ or $a \rightarrow x$. The rule sequence forms a top-down, left-most derivation under a CFG with skeletal rules.

A *full tree* consists of an s-tree $r_1 \ldots r_N$, together with values $h_1 \ldots h_N$. Each h_i is the value for the hidden variable for the left-hand-side of rule r_i. Each h_i can take any value in $[m]$.

For a given skeletal tree $r_1 \ldots r_N$, define a_i to be the non-terminal on the left-hand-side of rule r_i. For any $i \in [N]$ such that r_i is of the form $a \rightarrow b\ c$, define $h_i^{(2)}$ and $h_i^{(3)}$ as the hidden state value of the left and right child respectively. The model then defines a probability mass function (PMF) as

$$p(r_1 \ldots r_N, h_1 \ldots h_N) = \pi(a_1, h_1) \prod_{i: a_i \in \mathcal{I}} t(r_i, h_i^{(2)}, h_i^{(3)} | a_i, h_i) \prod_{i: a_i \in \mathcal{P}} q(r_i | a_i, h_i)$$

The PMF over skeletal trees is $p(r_1 \ldots r_N) = \sum_{h_1 \ldots h_N} p(r_1 \ldots r_N, h_1 \ldots h_N)$.

[2] For any integer n, we use $[n]$ to denote the set $\{1, 2, \ldots n\}$.

The parsing problem is to take a sentence as input, and produce a skeletal tree as output. A standard method for parsing with L-PCFGs is as follows. First, for a given input sentence $x_1 \ldots x_n$, for any triple (a, i, j) such that $a \in \mathcal{N}$ and $1 \leq i \leq j \leq n$, the marginal $\mu(a, i, j)$ is defined as

$$\mu(a, i, j) = \sum_{t: (a,i,j) \in t} p(t) \quad (1)$$

where the sum is over all skeletal trees t for $x_1 \ldots x_n$ that include non-terminal a spanning words $x_i \ldots x_j$. A variant of the inside-outside algorithm can be used to calculate marginals. Once marginals have been computed, Goodman's algorithm (Goodman, 1996) is used to find $\arg\max_t \sum_{(a,i,j) \in t} \mu(a, i, j)$.[3]

2.2 The Spectral Learning Algorithm

We now give a sketch of the spectral learning algorithm. The training data for the algorithm is a set of skeletal trees. The output from the algorithm is a set of parameter estimates for t, q and π (more precisely, the estimates are estimates of linearly transformed parameters; see Cohen et al. (2012) and section 2.3.1 for more details).

The algorithm takes two inputs in addition to the set of skeletal trees. The first is an integer m, specifying the number of latent state values in the model. Typically m is a relatively small number; in our experiments we test values such as $m = 8, 16$ or 32. The second is a pair of functions ϕ and ψ, that respectively map *inside* and *outside trees* to feature vectors in $\mathbb{R}^d$ and $\mathbb{R}^{d'}$, where d and d' are integers. Each non-terminal in a skeletal tree has an associated inside and outside tree. The inside tree for a node contains the entire subtree below that node; the outside tree contains everything in the tree excluding the inside tree. We will refer to the node above the inside tree that has been removed as the "foot" of the outside tree. See figure 1 for an example.

Section 3.1 gives definitions of $\phi(t)$ and $\psi(o)$ used in our experiments. The definitions of $\phi(t)$ and

[3] In fact, in our implementation we calculate marginals $\mu(a \rightarrow b\ c, i, k, j)$ for $a, b, c \in \mathcal{N}$ and $1 \leq i \leq k < j$, and $\mu(a, i, i)$ for $a \in \mathcal{N}$, $1 \leq i \leq n$, then apply the CKY algorithm to find the parse tree that maximizes the sum of the marginals. For simplicity of presentation we will refer to marginals of the form $\mu(a, i, j)$ in the remainder of this paper.

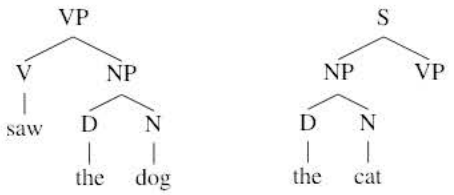


Figure 1: The inside tree (shown left) and outside tree (shown right) for the non-terminal VP in the parse tree [S [NP [D the] [N cat]] [VP [V saw] [NP [D the] [N dog]]]]

$\psi(o)$ are typically high-dimensional, sparse feature vectors, similar to those in log-linear models. For example ϕ might track the rule immediately below the root of the inside tree, or larger tree fragments; ψ might include similar features tracking rules or larger rule fragments above the relevant node.

The spectral learning algorithm proceeds in two steps. In step 1, we learn an m-dimensional representation of inside and outside trees, using the functions ϕ and ψ in combination with a projection step defined through singular value decomposition (SVD). In step 2, we derive parameter estimates directly from training examples.

2.2.1 Step 1: An SVD-Based Projection

For a given non-terminal $a \in \mathcal{N}$, each instance of a in the training data has an associated outside tree, and an associated inside tree. We define O^a to be the set of pairs of inside/outside trees seen with a in the training data: each member of O^a is a pair (o, t) where o is an outside tree, and t is an inside tree.

Step 1 of the algorithm is then as follows:

1. For each $a \in \mathcal{N}$ calculate $\hat{\Omega}^a \in \mathbb{R}^{d \times d'}$ as

$$[\hat{\Omega}^a]_{i,j} = \frac{1}{|O^a|} \sum_{(o,t) \in O^a} \phi_i(t)\psi_j(o)$$

2. Perform an SVD on $\hat{\Omega}^a$. Define $U^a \in \mathbb{R}^{d \times m}$ ($V^a \in \mathbb{R}^{d' \times m}$) to be a matrix containing the m left (right) singular vectors corresponding to the m largest singular values; define $\Sigma^a \in \mathbb{R}^{m \times m}$ to be the diagonal matrix with the m largest singular values on its diagonal.

3. For each inside tree in the corpus with root label a, define

$$Y(t) = (U^a)^\top \phi(t)$$

For each outside tree with a foot node labeled a, define

$$Z(o) = (\Sigma^a)^{-1}(V^a)^\top \psi(o)$$

Note that $Y(t)$ and $Z(o)$ are both m-dimensional vectors; thus we have used SVD to project inside and outside trees to m-dimensional vectors.

2.3 Step 2: Parameter Estimation

We now describe how the functions $Y(t)$ and $Z(o)$ are used in estimating parameters of the model. First, consider the $t(a \rightarrow b\ c, h_2, h_3 | a, h_1)$ parameters. Each instance of a given rule $a \rightarrow b\ c$ in the training corpus has an outside tree o associated with the parent labeled a, and inside trees t^2 and t^3 associated with the children labeled b and c. For any rule $a \rightarrow b\ c$ we define $Q^{a \rightarrow b\ c}$ to be the set of triples $(o, t^{(2)}, t^{(3)})$ occurring with that rule in the corpus. The parameter estimate is then

$$\hat{c}(a \rightarrow b\ c, j, k | a, i) \qquad \frac{\text{count}(a \rightarrow b\ c)}{\text{count}(a)} \times E^{a \rightarrow b\ c}_{i,j,k} \tag{2}$$

where

$$E^{a \rightarrow b\ c}_{i,j,k} \qquad \frac{\sum_{(o,t^{(2)},t^{(3)}) \in Q^{a \rightarrow b\ c}} Z_i(o) \times Y_j(t^{(2)}) \times Y_k(t^{(3)})}{|Q^{a \rightarrow b\ c}|}$$

Here we use $\text{count}(a \rightarrow b\ c)$ and $\text{count}(a)$ to refer to the count of the rule $a \rightarrow b\ c$ and the non-terminal a in the corpus. Note that once the SVD step has been used to compute representations $Y(t)$ and $Z(o)$ for each inside and outside tree in the corpus, calculating the parameter value $\hat{c}(a \rightarrow b\ c, j, k | a, i)$ is a very simple operation.

Similarly, for any rule $a \rightarrow x$, define $Q^{a \rightarrow x}$ to be the set of outside trees seen with that rule in the training corpus. The parameter estimate is then

$$\hat{c}(a \rightarrow x | a, i) \qquad \frac{\text{count}(a \rightarrow x)}{\text{count}(a)} \times E^{a \rightarrow x}_i \tag{3}$$

where $E^{a \rightarrow x}_i = \sum_{o \in Q^{a \rightarrow x}} Z_i(o) / |Q^{a \rightarrow x}|$.

A similar method is used for estimating parameters $\hat{c}(a, i)$ that play the role of the π parameters (details omitted for brevity; see Cohen et al. (2012)).

2.3.1 Guarantees for the Algorithm

Once the $\hat{c}(a \rightarrow b\ c, j, k | a, i)$, $\hat{c}(a \rightarrow x | a, i)$ and $\hat{c}(a, i)$ parameters have been estimated from the

training corpus, they can be used in place of the t, q and π parameters in the inside-outside algorithm for computing marginals (see Eq. 1). Call the resulting marginals $\hat{\mu}(a, i, j)$. The guarantees for the parameter estimation method are as follows:

- Define $\Omega^a = \mathbb{E}[\phi(T)(\psi(O))^\top | A = a]$ where A, O, T are random variables corresponding to the non-terminal label at a node, the outside tree, and the inside tree (see Cohen et al. (2012) for a precise definition). Note that $\hat{\Omega}^a$, as defined above, is an estimate of Ω^a. Then if Ω^a has rank m, the marginals $\hat{\mu}$ will converge to the true values μ as the number of training examples goes to infinity, assuming that the training samples are i.i.d. samples from an L-PCFG.

- Define σ to be the m'th largest singular value of Ω^a. Then the number of samples required for $\hat{\mu}$ to be ϵ-close to μ with probability at least $1 - \delta$ is polynomial in $1/\epsilon$, $1/\delta$, and $1/\sigma$.

Under the first assumption, (Cohen et al., 2012) show that the $\hat{c}$ parameters converge to values that are linear transforms of the original parameters in the L-PCFG. For example, define $c(a \rightarrow b\ c, j, k|a, i)$ to be the value that $\hat{c}(a \rightarrow b\ c, j, k|a, i)$ converges to in the limit of infinite data. Then there exist invertible matrices $G^a \in \mathbb{R}^{m \times m}$ for all $a \in \mathcal{N}$ such that for any $a \rightarrow b\ c$, for any $h_1, h_2, h_3 \in \mathbb{R}^m$,

$$t(a \rightarrow b\ c, h_2, h_3|a, h_1) \bigtimes_{i,j,k} [G^a]_{i,h_1} [(G^b)^{-1}]_{j,h_2} [(G^c)^{-1}]_{k,h_3} c(a \rightarrow b\ c, j, k|a, i)$$

The transforms defined by the G^a matrices are benign, in that they cancel in the inside-outside algorithm when marginals $\mu(a, i, j)$ are calculated. Similar relationships hold for the π and q parameters.

3 Implementation of the Algorithm

Cohen et al. (2012) introduced the spectral learning algorithm, but did not perform experiments, leaving several choices open in how the algorithm is implemented in practice. This section describes a number of key choices made in our implementation of the algorithm. In brief, they are as follows:

The choice of functions ϕ and ψ. We will describe basic features used in ϕ and ψ (single-level rules, larger tree fragments, etc.). We will also describe a method for scaling different features in ϕ and ψ by their variance, which turns out to be important for empirical results.

Estimation of $E^{a \rightarrow b\ c}_{i,j,k}$ and $E^{a \rightarrow x}_i$. There are a very large number of parameters in the model, leading to challenges in estimation. The estimates in Eqs. 2 and 3 are unsmoothed. We describe a simple backed-off smoothing method that leads to significant improvements in performance of the method.

Handling positive and negative values. As defined, the $\hat{c}$ parameters may be positive or negative; as a result, the $\hat{\mu}$ values may also be positive or negative. We find that negative values can be a significant problem if not handled correctly; but with a very simple fix to the algorithm, it performs well.

We now turn to these three issues in more detail. Section 4 will describe experiments measuring the impact of the different choices.

3.1 The Choice of Functions ϕ and ψ

Cohen et al. (2012) show that the choice of feature definitions ϕ and ψ is crucial in two respects. First, for all non-terminals $a \in \mathcal{N}$, the matrix Ω^a must be of rank m: otherwise the parameter-estimation algorithm will not be consistent. Second, the number of samples required for learning is polynomial in $1/\sigma$, where $\sigma = \min_{a \in \mathcal{N}} \sigma_m(\Omega^a)$, and $\sigma_m(\Omega^a)$ is the m'th smallest singular value of Ω^a. (Note that the second condition is stronger than the first; $\sigma > 0$ implies that Ω^a is of rank m for all a.) The choice of ϕ and ψ has a direct impact on the value for σ: roughly speaking, the value for σ can be thought of as a measure of how informative the functions ϕ and ψ are about the hidden state values.

With this in mind, our goal is to define a relatively simple set of features, which nevertheless provide significant information about hidden-state values, and hence provide high accuracy under the model. The inside-tree feature function $\phi(t)$ makes use of the following indicator features (throughout these definitions assume that $a \rightarrow b\ c$ is at the root of the inside tree t):

• The pair of nonterminals (a, b). E.g., for the inside tree in figure 1 this would be the pair `(VP, V)`.

• The pair (a, c). E.g., (VP, NP).

• The rule $a \rightarrow b\ c$. E.g., VP $\rightarrow$ V NP.

• The rule $a \rightarrow b\ c$ paired with the rule at the root of $t^{(i,2)}$. E.g., for the inside tree in figure 1 this would correspond to the tree fragment (VP (V saw) NP).

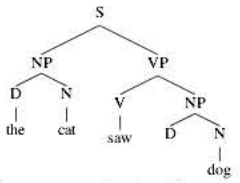
• The rule $a \rightarrow b\ c$ paired with the rule at the root of $t^{(i,3)}$. E.g., the tree fragment (VP V (NP D N)).

• The head part-of-speech of $t^{(i,1)}$ paired with a.[4] E.g., the pair (VP, V).

• The number of words dominated by $t^{(i,1)}$ paired with a (this is an integer valued feature).

In the case of an inside tree consisting of a single rule $a \rightarrow x$ the feature vector simply indicates the identity of that rule.

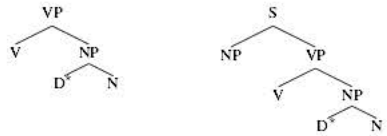
To illustrate the function ψ, it will be useful to make use of the following example outside tree:



Note that in this example the foot node of the outside tree is labeled D. The features are as follows:

• The rule above the foot node. We take care to mark which non-terminal is the foot, using a * symbol. In the above example this feature is NP $\rightarrow$ D* N.

• The two-level and three-level rule fragments above the foot node. In the above example these features would be



• The label of the foot node, together with the label of its parent. In the above example this is (D, NP).

• The label of the foot node, together with the label of its parent and grandparent. In the above example this is (D, NP, VP).

• The part of speech of the first head word along the path from the foot of the outside tree to the root of the tree which is different from the head node of the foot node. In the above example this is N.

• The width of the span to the left of the foot node, paired with the label of the foot node.

• The width of the span to the right of the foot node, paired with the label of the foot node.

Scaling of features. The features defined above are almost all binary valued features. We scale the features in the following way. For each feature $\phi_i(t)$, define count(i) to be the number of times the feature is equal to 1, and M to be the number of training examples. The feature is then redefined to be

$$\phi_i(t) \times \sqrt{\frac{M}{\text{count}(i) + \kappa}}$$

where κ is a smoothing term (the method is relatively insensitive to the choice of κ; we set $\kappa = 5$ in our experiments). A similar process is applied to the ψ features. The method has the effect of decreasing the importance of more frequent features in the SVD step of the algorithm.

The SVD-based step of the algorithm is very closely related to previous work on CCA (Hotelling, 1936; Hardoon et al., 2004; Kakade and Foster, 2009); and the scaling step is derived from previous work on CCA (Dhillon et al., 2011). In CCA the ϕ and ψ vectors are "whitened" in a preprocessing step, before an SVD is applied. This whitening process involves calculating covariance matrices $C_x = \mathrm{E}[\phi\phi^\top]$ and $C_y = \mathrm{E}[\psi\psi^\top]$, and replacing ϕ by $(C^x)^{-1/2}\phi$ and ψ by $(C^y)^{-1/2}\psi$. The exact calculation of $(C^x)^{-1/2}$ and $(C^y)^{-1/2}$ is challenging in high dimensions, however, as these matrices will not be sparse; the transformation described above can be considered an approximation where off-diagonal members of C_x and C_y are set to zero. We will see that empirically this scaling gives much improved accuracy.

3.2 Estimation of $E^{a \rightarrow b\ c}_{i,j,k}$ and $E^{a \rightarrow x}_i$

The number of $E^{a \rightarrow b\ c}_{i,j,k}$ parameters is very large, and the estimation method described in Eqs. 2–3 is unsmoothed. We have found significant improvements in performance using a relatively simple backoff smoothing method. The intuition behind this method is as follows: given two random variables X and Y, under the assumption that the random variables are independent, $\mathrm{E}[XY] = \mathrm{E}[X] \times \mathrm{E}[Y]$. It

[4] We use the English head rules from the Stanford parser (Klein and Manning, 2003).

makes sense to define "backed off" estimates which make increasingly strong independence assumptions of this form.

Smoothing of binary rules For any rule $a \rightarrow b\ c$ and indices $i, j \in [m]$ we can define a second-order moment as follows:

$$E^{a \rightarrow b\ c}_{i,j,\cdot} = \frac{\sum_{(o,t^{(2)},t^{(3)}) \in Q^{a \rightarrow b\ c}} Z_i(o) \times Y_j(t^{(2)})}{|Q^{a \rightarrow b\ c}|}$$

The definitions of $E^{a \rightarrow b\ c}_{i,\cdot,k}$ and $E^{a \rightarrow b\ c}_{\cdot,j,k}$ are analogous. We can define a first-order estimate as follows:

$$E^{a \rightarrow b\ c}_{\cdot,\cdot,k} = \frac{\sum_{(o,t^{(2)},t^{(3)}) \in Q^{a \rightarrow b\ c}} Y_k(t^{(3)})}{|Q^{a \rightarrow b\ c}|}$$

Again, we have analogous definitions of $E^{a \rightarrow b\ c}_{i,\cdot,\cdot}$ and $E^{a \rightarrow b\ c}_{\cdot,j,\cdot}$. Different levels of smoothed estimate can be derived from these different terms. The first is

$$E^{2,a \rightarrow b\ c}_{i,j,k} = \frac{E^{a \rightarrow b\ c}_{i,j,\cdot} \times E^{a \rightarrow b\ c}_{\cdot,\cdot,k} + E^{a \rightarrow b\ c}_{i,\cdot,k} \times E^{a \rightarrow b\ c}_{\cdot,j,\cdot} + E^{a \rightarrow b\ c}_{\cdot,j,k} \times E^{a \rightarrow b\ c}_{i,\cdot,\cdot}}{3}$$

Note that we give an equal weight of $1/3$ to each of the three backed-off estimates seen in the numerator. A second smoothed estimate is

$$E^{3,a \rightarrow b\ c}_{i,j,k} = E^{a \rightarrow b\ c}_{i,\cdot,\cdot} \times E^{a \rightarrow b\ c}_{\cdot,j,\cdot} \times E^{a \rightarrow b\ c}_{\cdot,\cdot,k}$$

Using the definition of O^a given in section 2.2.1, we also define

$$F^a_i = \frac{\sum_{(o,t) \in O^a} Y_i(t)}{|O^a|} \qquad H^a_i = \frac{\sum_{(o,t) \in O^a} Z_i(o)}{|O^a|}$$

and our next smoothed estimate as $E^{4,a \rightarrow b\ c}_{i,j,k} = H^a_i \times F^b_j \times F^c_k$.

Our final estimate is

$$\lambda E^{a \rightarrow b\ c}_{i,j,k} + (1 - \lambda) \left(\lambda E^{2,a \rightarrow b\ c}_{i,j,k} + (1 - \lambda) K^{a \rightarrow b\ c}_{i,j,k} \right)$$

where $K^{a \rightarrow b\ c}_{i,j,k} = \lambda E^{3,a \rightarrow b\ c}_{i,j,k} + (1 - \lambda) E^{4,a \rightarrow b\ c}_{i,j,k}$.

Here $\lambda \in [0, 1]$ is a smoothing parameter, set to $\sqrt{|Q^{a \rightarrow b\ c}|}/(C + \sqrt{|Q^{a \rightarrow b\ c}|})$ in our experiments, where C is a parameter that is chosen by optimization of accuracy on a held-out set of data.

Smoothing lexical rules We define a similar method for the $E^{a \rightarrow x}_i$ parameters. Define

$$E^a_i = \frac{\sum_{x'} \sum_{o \in Q^{a \rightarrow x'}} Z_i(o)}{\sum_{x'} |Q^{a \rightarrow x'}|}$$

hence E^a_i ignores the identity of x in making its estimate. The smoothed estimate is then defined as $\nu E^{a \rightarrow x}_i + (1 - \nu) E^a_i$. Here, ν is a value in $[0, 1]$ which is tuned on a development set. We only smooth lexical rules which appear in the data less than a fixed number of times. Unlike binary rules, for which the estimation depends on a high order moment (third moment), the lexical rules use first-order moments, and therefore it is not required to smooth rules with a relatively high count. The maximal count for this kind of smoothing is set using a development set.

3.3 Handling Positive and Negative Values

As described before, the parameter estimates may be positive or negative, and as a result the marginals computed by the algorithm may in some cases themselves be negative. In early experiments we found this to be a signficant problem, with some parses having a very large number of negatives, and being extremely poor in quality. Our fix is to define the output of the parser to be $\arg\max_t \sum_{(a,i,j) \in t} |\mu(a, i, j)|$ rather than $\arg\max_t \sum_{(a,i,j) \in t} \mu(a, i, j)$ as defined in Goodman's algorithm. Thus if a marginal value $\mu(a, i, j)$ is negative, we simply replace it with its absolute value. This step was derived after inspection of the parsing charts for bad parses, where we saw evidence that in these cases the entire set of marginal values had been negated (and hence decoding under Eq. 1 actually leads to the *lowest* probability parse being output under the model). We suspect that this is because in some cases a dominant parameter has had its sign flipped due to sampling error; more theoretical and empirical work is required in fully understanding this issue.

4 Experiments

In this section we describe parsing experiments using the L-PCFG estimation method. We give comparisons to the EM algorithm, considering both speed of training, and accuracy of the resulting model; we also give experiments investigating the various choices described in the previous section.

We use the Penn WSJ treebank (Marcus et al., 1993) for our experiments. Sections 2–21 were used as training data, and sections 0 and 22 were used as development data. Section 23 is used as the final test set. We binarize the trees in training data using the same method as that described in Petrov et al. (2006). For example, the non-binary rule `VP` $\rightarrow$ `V NP PP SBAR` would be converted to the structure `[VP [@VP [@VP V NP] PP] SBAR]` where `@VP` is a new symbol in the grammar. Unary rules are removed by collapsing non-terminal chains: for example the unary rule `S` $\rightarrow$ `VP` would be replaced by a single non-terminal `S|VP`.

For the EM algorithm we use the initialization method described in Matsuzaki et al. (2005). For efficiency, we use a coarse-to-fine algorithm for parsing with either the EM or spectral derived grammar: a PCFG without latent states is used to calculate marginals, and dynamic programming items are removed if their marginal probability is lower than some threshold (0.00005 in our experiments).

For simplicity the parser takes part-of-speech tagged sentences as input. We use automatically tagged data from Turbo Tagger (Martins et al., 2010). The tagger is used to tag both the development data and the test data. The tagger was retrained on sections 2–21. We use the F_1 measure according to the Parseval metric (Black et al., 1991). For the spectral algorithm, we tuned the smoothing parameters using section 0 of the treebank.

4.1 Comparison to EM: Accuracy

We compare models trained using EM and the spectral algorithm using values for m in $\{8, 16, 24, 32\}$.[5]

For EM, we found that it was important to use development data to choose the number of iterations of training. We train the models for 100 iterations, then test accuracy of the model on section 22 (development data) at different iteration numbers. Table 1 shows that a peak level of accuracy is reached for all values of m, other than $m = 8$, at iteration 20–30, with sometimes substantial overtraining beyond that point.

The performance of a regular PCFG model, estimated using maximum likelihood and with no latent states, is 68.62%.

		section 22		section 23	
		EM	spectral	EM	spectral
m	8	86.87	85.60	—	—
m	16	88.32	87.77	—	—
m	24	88.35	88.53	—	—
m	32	88.56	88.82	87.76	88.05

Table 2: Results on the development data (section 22, with machine-generated POS tags) and test data (section 23, with machine-generated POS tags).

Table 2 gives results for the EM-trained models and spectral-trained models. The spectral models give very similar accuracy to the EM-trained model on the test set. Results on the development set with varying m show that the EM-based models perform better for $m = 8$, but that the spectral algorithm quickly catches up as m increases.

4.2 Comparison to EM: Training Speed

Table 3 gives training times for the EM algorithm and the spectral algorithm for $m \in \{8, 16, 24, 32\}$. All timing experiments were done on a single Intel Xeon 2.67GHz CPU. The implementations for the EM algorithm and the spectral algorithm were written in Java. The spectral algorithm also made use of Matlab for several matrix calculations such as the SVD calculation.

For EM we show the time to train a single iteration, and also the time to train the optimal model (time for 30 iterations of training for $m = 8, 16, 24$, and time for 20 iterations for $m = 32$). Note that this latter time is optimistic, as it assumes an oracle specifying exactly when it is possible to terminate EM training with no loss in performance. The spectral method is considerably faster than EM: for example, for $m = 32$ the time for training the spectral model is just under 10 hours, compared to 187 hours for EM, a factor of almost 19 times faster.[6]

The reason for these speed ups is as follows. Step 1 of the spectral algorithm (feature calculation, transfer + scaling, and SVD) is not required by EM, but takes a relatively small amount of time (about 1.2 hours for all values of m). Once step 1 has been completed, step 2 of the spectral algorithm takes a

[5] Lower values of m, such as 2 or 4, lead to substantially lower performance for both models.

[6] In practice, in order to overcome the speed issue with EM training, we parallelized the E-step on multiple cores. The spectral algorithm can be similarly parallelized, computing statistics and parameters for each nonterminal separately.

		10	20	30	40	50	60	70	80	90	100
m	8	83.51	86.45	86.68	86.69	86.63	86.67	86.70	86.82	**86.87**	86.83
m	16	85.18	87.94	**88.32**	88.21	88.10	87.86	87.70	87.46	87.34	87.24
m	24	83.62	88.19	**88.35**	88.25	87.73	87.41	87.35	87.26	87.02	86.80
m	32	83.23	**88.56**	88.52	87.82	87.06	86.47	86.38	85.85	85.75	85.57

Table 1: Results on section 22 for the EM algorithm, varying the number of iterations used. Best results in each row are in boldface.

	single EM iter.	EM best model	spectral algorithm total	feature	transfer + scaling	SVD	$a \rightarrow b\ c$	$a \rightarrow x$
$m = 8$	6m	3h	3h32m	22m	49m	36m	1h34m	10m
$m = 16$	52m	26h6m	5h19m			34m	3h13m	19m
$m = 24$	3h7m	93h36m	7h15m			36m	4h54m	28m
$m = 32$	9h21m	187h12m	9h52m			35m	7h16m	41m

Table 3: Running time for the EM algorithm and the various stages in the spectral algorithm. For EM we show the time for a single iteration, and the time to train the optimal model (time for 30 iterations of training for m 8, 16, 24, time for 20 iterations of training for m 32). For the spectral method we show the following: "total" is the total training time; "feature" is the time to compute the ϕ and ψ vectors for all data points; "transfer + scaling" is time to transfer the data from Java to Matlab, combined with the time for scaling of the features; "SVD" is the time for the SVD computation; $a \rightarrow b\ c$ is the time to compute the $\hat{c}(a \rightarrow b\ c, h_2, h_3 | a, h_1)$ parameters; $a \rightarrow x$ is the time to compute the $\hat{c}(a \rightarrow x, h | a, h)$ parameters. Note that "feature" and "transfer + scaling" are the same step for all values of m, so we quote a single runtime for these steps.

single pass over the data: in contrast, EM requires a few tens of passes (certainly more than 10 passes, from the results in table 1). The computations performed by the spectral algorithm in its single pass are relatively cheap. In contrast to EM, the inside-outside algorithm is not required; however various operations such as calculating smoothing terms in the spectral method add some overhead. The net result is that for $m = 32$ the time for training the spectral method takes a very similar amount of time to a single pass of the EM algorithm.

4.3 Smoothing, Features, and Negatives

We now describe experiments demonstrating the impact of various components described in section 3.

The effect of smoothing (section 3.2) Without smoothing, results on section 22 are 85.05% ($m = 8$, -1.82), 86.84% ($m = 16$, -1.48), 86.47% ($m = 24$, -1.88), 86.47% ($m = 32$, -2.09) (in each case we show the decrease in performance from the results in table 2). Smoothing is clearly important.

Scaling of features (section 3.1) Without scaling of features, the accuracy on section 22 with $m = 32$ is 84.40%, a very significant drop from the 88.82% accuracy achieved with scaling.

Handling negative values (section 3.3) Replacing marginal values $\mu(a, i, j)$ with their absolute values is also important: without this step, accuracy on section 22 decreases to 80.61% ($m = 32$). 319 sentences out of 1700 examples have different parses when this step is implemented, implying that the problem with negative values described in section 3.3 occurs on around 18% of all sentences.

The effect of feature functions To test the effect of features on accuracy, we experimented with a simpler set of features than those described in section 3.1. This simple set just includes an indicator for the rule below a nonterminal (for inside trees) and the rule above a nonterminal (for outside trees). Even this simpler set of features achieves relatively high accuracy ($m = 8$: 86.44 , $m = 16$: 86.86, $m = 24$: 87.24 , $m = 32$: 88.07).

This set of features is reminiscent of a PCFG model where the nonterminals are augmented their parents (vertical Markovization of order 2) and binarization is done while retaining sibling information (horizontal Markovization of order 1). See Klein and Manning (2003) for more information. The per-

formance of this Markovized PCFG model lags behind the spectral model: it is 82.59%. This is probably due to the complexity of the grammar which causes ovefitting. Condensing the sibling and parent information using latent states as done in the spectral model leads to better generalization.

It is important to note that the results for both EM and the spectral algorithm are comparable to state of the art, but there are other results previously reported in the literature which are higher. For example, Hiroyuki et al. (2012) report an accuracy of 92.4 F_1 on section 23 of the Penn WSJ treebank using a Bayesian tree substitution grammar; Charniak and Johnson (2005) report accuracy of 91.4 using a discriminative reranking model; Carreras et al. (2008) report 91.1 F_1 accuracy for a discriminative, perceptron-trained model; Petrov and Klein (2007) report an accuracy of 90.1 F_1, using L-PCFGs, but with a split-merge training procedure. Collins (2003) reports an accuracy of 88.2 F_1, which is comparable to the results in this paper.

5 Conclusion

The spectral learning algorithm gives the same level of accuracy as EM in our experiments, but has significantly faster training times. There are several areas for future work. There are a large number of parameters in the model, and we suspect that more sophisticated regularization methods than the smoothing method we have described may improve performance. Future work should also investigate other choices for the functions ϕ and ψ. There are natural ways to extend the approach to semi-supervised learning; for example the SVD step, where representations of outside and inside trees are learned, could be applied to unlabeled data parsed by a first-pass parser. Finally, the methods we have described should be applicable to spectral learning for other latent variable models.

Acknowledgements

Columbia University gratefully acknowledges the support of the Defense Advanced Research Projects Agency (DARPA) Machine Reading Program under Air Force Research Laboratory (AFRL) prime contract no. FA8750-09-C-0181. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the view of DARPA, AFRL, or the US government. Shay Cohen was supported by the National Science Foundation under Grant #1136996 to the Computing Research Association for the CIFellows Project. Dean Foster was supported by National Science Foundation grant 1106743.

References

A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky. 2012. Tensor decompositions for learning latent-variable models. arXiv:1210.7559.

S. Arora, R. Se, and A. Moitra. 2012. Learning topic models - going beyond SVD. In *Proceedings of FOCS*.

R. Bailly, A. Habrar, and F. Denis. 2010. A spectral approach for probabilistic grammatical inference on trees. In *Proceedings of ALT*.

B. Balle, A. Quattoni, and X. Carreras. 2011. A spectral learning algorithm for finite state transducers. In *Proceedings of ECML*.

E. Black, S. Abney, D. Flickenger, C. Gdaniec, R. Grishman, P Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of DARPA Workshop on Speech and Natural Language*.

X. Carreras, M. Collins, and T. Koo. 2008. TAG, Dynamic Programming, and the Perceptron for Efficient, Feature-rich Parsing. In *Proceedings of CoNLL*, pages 9–16.

E. Charniak and M. Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of ACL*.

S. B. Cohen, K. Stratos, M. Collins, D. F. Foster, and L. Ungar. 2012. Spectral learning of latent-variable PCFGs. In *Proceedings of ACL*.

M. Collins. 2003. Head-driven statistical models for natural language processing. *Computational Linguistics*, 29:589–637.

P. Dhillon, D. P. Foster, and L. H. Ungar. 2011. Multi-view learning of word embeddings via CCA. In *Proceedings of NIPS*.

P. Dhillon, J. Rodu, M. Collins, D. P. Foster, and L. H. Ungar. 2012. Spectral dependency parsing with latent variables. In *Proceedings of EMNLP*.

J. Goodman. 1996. Parsing algorithms and metrics. In *Proceedings of ACL*.

D. Hardoon, S. Szedmak, and J. Shawe-Taylor. 2004. Canonical correlation analysis: An overview with application to learning methods. *Neural Computation*, 16(12):2639–2664.

S. Hiroyuki, M. Yusuke, F. Akinori, and N. Masaaki. 2012. Bayesian symbol-refined tree substitution grammars for syntactic parsing. In *Proceedings of ACL*, pages 440–448.

H. Hotelling. 1936. Relations between two sets of variants. *Biometrika*, 28:321–377.

D. Hsu, S. M. Kakade, and T. Zhang. 2009. A spectral algorithm for learning hidden Markov models. In *Proceedings of COLT*.

S. M. Kakade and D. P. Foster. 2009. Multi-view regression via canonical correlation analysis. In *COLT*.

D. Klein and C. D. Manning. 2003. Accurate unlexicalized parsing. In *Proc. of ACL*, pages 423–430.

F. M. Luque, A. Quattoni, B. Balle, and X. Carreras. 2012. Spectral learning for non-deterministic dependency parsing. In *Proceedings of EACL*.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19:313–330.

A. F. T. Martins, N. A. Smith, E. P. Xing, M. T. Figueiredo, and M. Q. Aguiar. 2010. TurboParsers: Dependency parsing by approximate variational inference. In *Proceedings of EMNLP*.

T. Matsuzaki, Y. Miyao, and J. Tsujii. 2005. Probabilistic CFG with latent annotations. In *Proceedings of ACL*.

A. Parikh, L. Song, and E. P. Xing. 2011. A spectral algorithm for latent tree graphical models. In *Proceedings of The 28th International Conference on Machine Learningy (ICML 2011)*.

S. Petrov and D. Klein. 2007. Improved inference for unlexicalized parsing. In *Proc. of HLT-NAACL*.

S. Petrov, L. Barrett, R. Thibaux, and D. Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of COLING-ACL*.

S. Siddiqi, B. Boots, and G. Gordon. 2010. Reduced-rank hidden markov models. *JMLR*, 9:741–748.

S. Vempala and G. Wang. 2004. A spectral algorithm for learning mixtures of distributions. *Journal of Computer and System Sciences*, 68(4):841–860.