A Provably Correct Learning Algorithm for Latent-Variable PCFGs

Citation for published version:
Cohen, S & Collins, M 2014, A Provably Correct Learning Algorithm for Latent-Variable PCFGs. in Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, pp. 1052-1061.

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
A Provably Correct Learning Algorithm for Latent-Variable PCFGs

Shay B. Cohen
School of Informatics
University of Edinburgh
scohen@inf.ed.ac.uk

Michael Collins
Department of Computer Science
Columbia University
mcollins@cs.columbia.edu

Abstract

We introduce a provably correct learning algorithm for latent-variable PCFGs. The algorithm relies on two steps: first, the use of a matrix-decomposition algorithm applied to a co-occurrence matrix estimated from the parse trees in a training sample; second, the use of EM applied to a convex objective derived from the training samples in combination with the output from the matrix decomposition. Experiments on parsing and a language modeling problem show that the algorithm is efficient and effective in practice.

1 Introduction

Latent-variable PCFGs (L-PCFGs) (Matsuzaki et al., 2005; Petrov et al., 2006) give state-of-the-art performance on parsing problems. The standard approach to parameter estimation in L-PCFGs is the EM algorithm (Dempster et al., 1977), which has the usual problems with local optima. Recent work (Cohen et al., 2012) has introduced an alternative algorithm, based on spectral methods, which has provable guarantees. Unfortunately this algorithm does not return parameter estimates for the underlying L-PCFG, instead returning the parameter values up to an (unknown) linear transform. In practice, this is a limitation.

We describe an algorithm that, like EM, returns estimates of the original parameters of an L-PCFG, but, unlike EM, does not suffer from problems of local optima. The algorithm relies on two key ideas:

1) A matrix decomposition algorithm (section 5) which is applicable to matrices $Q$ of the form $Q_{f,g} = \sum_h p(h)p(f|h)p(g|h)$ where $p(h), p(f|h)$ and $p(g|h)$ are multinomial distributions. This matrix form has clear relevance to latent variable models. We apply the matrix decomposition algorithm to a co-occurrence matrix that can be estimated directly from a training set consisting of parse trees without latent annotations. The resulting parameter estimates give us significant leverage over the learning problem.

2) Optimization of a convex objective function using EM. We show that once the matrix decomposition step has been applied, parameter estimation of the L-PCFG can be reduced to a convex optimization problem that is easily solved by EM.

The algorithm provably learns the parameters of an L-PCFG (theorem 1), under an assumption that each latent state has at least one “pivot” feature. This assumption is similar to the “pivot word” assumption used by Arora et al. (2013) and Arora et al. (2012) in the context of learning topic models.

We describe experiments on learning of L-PCFGs, and also on learning of the latent-variable language model of Saul and Pereira (1997). A hybrid method, which uses our algorithm as an initializer for EM, performs at the same accuracy as EM, but requires significantly fewer iterations for convergence: for example in our L-PCFG experiments, it typically requires 2 EM iterations for convergence, as opposed to 20-40 EM iterations for initializers used in previous work.

While this paper’s focus is on L-PCFGs, the techniques we describe are likely to be applicable to many other latent-variable models used in NLP.

2 Related Work

Recently a number of researchers have developed provably correct algorithms for parameter estimation in latent variable models such as hidden Markov models, topic models, directed graphical models with latent variables, and so on (Hsu et al., 2009; Bailly et al., 2010; Siddiqi et al., 2010; Parikh et al., 2011; Balle et al., 2011; Arora et al., 2013; Dhillon et al., 2012; Anandkumar et al., 2012; Arora et al., 2012; Arora et al., 2013). Many of these algorithms have their roots in spectral methods such as canonical correlation analysis (CCA) (Hotelling, 1936), or higher-order tensor decompositions. Previous work (Cohen et al., 2012; Cohen et al., 2013) has developed a spectral method for learning of L-PCFGs; this method learns parameters of the model up to an unknown
linear transformation, which cancels in the inside-
outside calculations for marginalization over la-
tent states in the L-PCFG. The lack of direct pa-
rameter estimates from this method leads to prob-
lems with negative or unnormalized probabilities;
the method does not give parameters that are in-
terpretable, or that can be used in conjunction with
other algorithms, for example as an initializer for
EM steps that refine the model.

Our work is most directly related to the algo-
rithm for parameter estimation in topic models de-
scribed by Arora et al. (2013). This algorithm
forms the core of the matrix decomposition algo-
rithm described in section 5.

3 Background

This section gives definitions and notation for L-
PCFGs, taken from (Cohen et al., 2012).

3.1 L-PCFGs: Basic Definitions

An L-PCFG is an 8-tuple \((\mathcal{N}, \mathcal{I}, \mathcal{P}, m, \pi, t, b, q)\)
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 as-
sume 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.\(^1\) \([n]\) is the set of possible words. For
all \((a, b, c) \in \mathcal{I} \times \mathcal{N} \times \mathcal{N}\), and \((h_1, h_2, h_3) \in
[m] \times [m] \times [m]\), we have a context-free rule
\(a(h_1) \rightarrow b(h_2) c(h_3)\). The rule has an asso-
ciated parameter \(t(a \rightarrow b c, h_2, h_3 \mid a, h_1)\). 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 pa-
rameter \(q(a \rightarrow x \mid a, h)\). 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 \(r_1 \ldots r_N\), to-
gether 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 distribution as

\[
p(r_1 \ldots r_N, h_1 \ldots h_N) = \\
\pi(a_1, h_1) \prod_{i, a \in \mathcal{I}} \left( t(r_i, h_i^{(2)}, h_i^{(3)} \mid a, h_i) \right) \prod_{i, a \in \mathcal{P}} q(r_i \mid a, h_i) \\
\]

The distribution over skeletal trees is

\[
p(r_1 \ldots r_N) = \sum_{h_1 \ldots h_N} p(r_1 \ldots r_N, h_1 \ldots h_N). \\
\]

3.2 Definition of Random Variables

Throughout this paper we will make reference
to random variables derived from the distribution
over full trees from an L-PCFG. These random
variables are defined as follows. First, we select
a random internal node, from a random tree, as
follows: 1) Sample a full tree \(r_1 \ldots r_N, h_1 \ldots h_N\)
from the PMF \(p(r_1 \ldots r_N, h_1 \ldots h_N)\); 2) Choose
a node \(i\) uniformly at random from \([N]\). We then
give the following definition:

Definition 1 (Random Variables). If the rule \(r_i\),
for the node \(i\) is of the form \(a \rightarrow b c\), we define ran-
dom variables as follows: \(R_1\) is equal to the rule \(r_i\)
(e.g., \(NP \rightarrow \mathcal{D}\)). \(A, B, C\) are the labels for node
\(i\), the left child of node \(i\), and the right child of node
\(i\) respectively. (E.g., \(A = NP, B = \mathcal{D}, C = \mathcal{N}\)). \(T_1\)
is the inside tree rooted at node \(i\). \(T_2\) is the inside
tree rooted at the left child of node \(i\), and \(T_3\) is the
inside tree rooted at the right child of node \(i\). \(O\) is
the outside tree at node \(i\). \(H_1, H_2, H_3\) are the hidden
variables associated with node \(i\), the left child
of node \(i\), and the right child of node \(i\) respectively.
\(E\) is equal to 1 if node \(i\) is at the root of the tree
(i.e., \(i = 1\)), 0 otherwise.

If the rule \(r_i\), for the selected node \(i\) is
of the form \(a \rightarrow x\), we have random vari-
ables \(R_1, T_1, H_1, A_1, O, E\) as defined above, but
\(H_2, H_3, T_2, T_3, B, C\) are not defined.

4 The Learning Algorithm for L-PCFGs

Our goal is to design a learning algorithm for L-
PCFGs. The input to the algorithm will be a train-
ing set consisting of skeletal trees, assumed to be
sampled from some underlying L-PCFG. The out-
put of the algorithm will be estimates for the \(\pi, t, q\) parameters. The training set does not
include values for the latent variables; this is the
main challenge in learning.

This section focuses on an algorithm for recov-
ery of the \(t\) parameters. A description of the algo-
rithms for recovery of the \(\pi\) and \(q\) parameters is
deferred until section 6.1 of this paper; these
steps are straightforward once we have derived the method for the $t$ parameters.

We describe an algorithm that correctly recovers the parameters of an L-PCFG as the size of the training set goes to infinity (this statement is made more precise in section 4.2). The algorithm relies on an assumption—the “pivot” assumption—that we now describe.

### 4.1 Features, and the Pivot Assumption

We assume a function $\tau$ from inside trees to a finite set $\mathcal{F}$, and a function $\rho$ that maps outside trees to a finite set $\mathcal{G}$. The function $\tau(t)$ ($\rho(o)$) can be thought of as a function that maps an inside tree $t$ (outside tree $o$) to an underlying feature. As one example, the function $\tau(t)$ might return the context-free rule at the root of the inside tree $t$; in this case the set $\mathcal{F}$ would be equal to the set of all context-free rules in the grammar. As another example, the function $\rho(o)$ might return the context-free rule at the foot of the outside tree $o$.

In the more general case, we might have $K$ separate functions $\tau^{(k)}(t)$ for $k = 1 \ldots K$ mapping inside trees to $K$ separate features, and similarly we might have multiple features for outside trees. Cohen et al. (2013) describe one such feature definition, where features track single context-free rules as well as larger fragments such as two or three-level sub-trees. For simplicity of presentation we describe the case of single features $\tau(t)$ and $\rho(o)$ for the majority of this paper. The extension to multiple features is straightforward, and is discussed in section 6.2; the flexibility allowed by multiple features is important, and we use multiple features in our experiments.

Given functions $\tau$ and $\rho$, we define additional random variables: $F = \tau(T_1), F_2 = \tau(T_2), F_3 = \tau(T_3),$ and $G = \rho(O)$.

We can now give the following assumption:

**Assumption 1 (The Pivot Assumption).** Under the L-PCFG being learned, there exist values $\alpha > 0$ and $\beta > 0$ such that for each non-terminal $a$, for each hidden state $h \in [m]$, the following statements are true: 1) $\exists f \in \mathcal{F}$ such that $P(F = f \mid H_1 = h, A = a) > \alpha$ and for all $h' \neq h$, $P(F = f \mid H_1 = h', A = a) = 0$; 2) $\exists g \in \mathcal{G}$ such that $P(G = g \mid H_1 = h, A = a) > \beta$ and for all $h' \neq h$, $P(G = g \mid H_1 = h', A = a) = 0$.

This assumption is very similar to the assumption made by Arora et al. (2012) in the context of learning topic models. It implies that for each $(a, h)$ pair, there are inside and outside tree features—which following Arora et al. (2012) we refer to as pivot features—that occur only in the presence of latent-state value $h$. As in (Arora et al., 2012), the pivot features will give us considerable leverage in learning of the model.

### 4.2 The Learning Algorithm

Figure 1 shows the learning algorithm for L-PCFGs. The algorithm consists of the following steps:

**Step 0:** Calculate estimates $\hat{p}(a \rightarrow b c \mid a)$, $\hat{p}(g, f_2, f_3 \mid a \rightarrow b c)$ and $\hat{p}(f, g \mid a)$. These estimates are easily calculated using counts taken from the training examples.

**Step 1:** Calculate values $\hat{r}(f \mid h, a)$ and $\hat{s}(g \mid h, a)$; these are estimates of $p(f \mid h_1, a)$ and $p(g \mid h_1, a)$ respectively. This step is achieved using a matrix decomposition algorithm, described in section 5 of this paper, on the matrix $\hat{Q}^a$ with entries $[\hat{Q}^a]_{fg} = \hat{p}(f, g \mid a)$.

**Step 2:** Use the EM algorithm to find $\hat{t}$ values that maximize the objective function in Eq. 1 (see figure 1). Crucially, this is a convex optimization problem, and the EM algorithm will converge to the global maximum of this likelihood function.

**Step 3:** Rule estimates are calculated using an application of the laws of probability.

Before giving a theorem concerning correctness of the algorithm we introduce two assumptions:

**Assumption 2 (Strict Convexity).** If we have the equalities $\hat{s}(g \mid h_1, a) = P(G = g \mid H_1 = h_1, A = a)$, $\hat{r}(f_2 \mid h_2, b) = P(F_2 = f_2 \mid H_2 = h_2, B = b)$ and $\hat{r}(f_3 \mid h_3, c) = P(F_3 = f_3 \mid H_2 = h_3, C = c)$, then the function in Eq. 1 (figure 1) is strictly concave.

The function in Eq. 1 is always concave; this assumption adds the restriction that the function must be strictly concave—that is, it has a unique global maximum—in the case that the $\hat{r}$ and $\hat{s}$ estimates are exact estimates.

**Assumption 3 (Infinite Data).** After running Step 0 of the algorithm we have

$\hat{p}(a \rightarrow b c \mid a) = p(a \rightarrow b c \mid a)$

$\hat{p}(g, f_2, f_3 \mid a \rightarrow b c) = p(g, f_2, f_3 \mid a \rightarrow b c)$

$\hat{p}(f, g \mid a) = p(f, g \mid a)$

where $p(\ldots)$ is the probability under the underlying L-PCFG.

---

2The requirements $P(F = f \mid H_1 = h', A = a) = 0$ and $P(G = g \mid H_1 = h', A = a) = 0$ are almost certainly overly strict; in theory and practice these probabilities should be able to take small but strictly positive values.
We use the term “infinite data” because under standard arguments, \( \hat{p}(\ldots) \) converges to \( p(\ldots) \) as \( M \) goes to \( \infty \).

The theorem is then as follows:

**Theorem 1.** Consider the algorithm in figure 1. Assume that assumptions 1-3 (the pivot, strong convexity, and infinite data assumptions) hold for the underlying L-PCFG. Then there is some permutation \( \sigma: [m] \to [m] \) such that for all \( a \to b \), \( h_1, h_2, h_3 \),

\[
\hat{t}(a \to b, h_2, h_3 | a \to b, h_1) = t(a \to b, \sigma(h_2), \sigma(h_3) | a \to b, \sigma(h_1))
\]

where \( \hat{t} \) are the parameters in the output, and \( t \) are the parameters of the underlying L-PCFG.

This theorem states that under assumptions 1-3, the algorithm correctly learns the \( t \) parameters of an L-PCFG, up to a permutation over the latent states defined by \( \sigma \). Given the assumptions we have made, it is not possible to do better than recovering the correct parameter values up to a permutation, due to symmetries in the model. Assuming that the \( \pi \) and \( q \) parameters are recovered in addition to the \( t \) parameters (see section 6.1), the resulting model will define exactly the same distribution over full trees as the underlying L-PCFG up to this permutation, and will define exactly the same distribution over skeletal trees, so in this sense the permutation is benign.

**Proof of theorem 1:** Under the assumptions of the theorem, \( Q^p_{f,g} = p(f,g | a) = \sum_h p(h | a)p(f | h, a)p(g | h, a) \). Under the pivot assumption, and theorem 2 of section 5, step 1 (the matrix decomposition step) will therefore recover values \( \hat{t} \) and \( \hat{s} \) such that \( \hat{r}(f | h, a) = p(f | \sigma(h), a) \) and \( \hat{s}(g | h, a) = p(g | \sigma(h), a) \) for some permutation \( \sigma: [m] \to [m] \). For simplicity, assume that \( \sigma(j) = j \) for all \( j \in [m] \) (the argument for other permutations involves a straightforward extension of the following argument). Under the assumptions of the theorem, \( \hat{p}(g, f_2, f_3 | a \to b c) = p(g, f_2, f_3 | a \to b c) \), hence the function being optimized in Eq. 1 is equal to

\[
\sum_{g,f_2,f_3} p(g, f_2, f_3 | a \to b c) \log \kappa(g, f_2, f_3)
\]

where

\[
\kappa(g, f_2, f_3) = \sum_{h_1, h_2, h_3} (\hat{t}(h_1, h_2, h_3 | a \to b c)
\times p(g | h_1, a)p(f_2 | h_2, b)p(f_3 | h_3, c))
\]

Now consider the optimization problem in Eq. 1. By standard results for cross entropy, the maximum of the function

\[
\sum_{g,f_2,f_3} p(g, f_2, f_3 | a \to b c) \log q(g, f_2, f_3 | a \to b c)
\]

with respect to the \( q \) values is achieved at

\[
q(g, f_2, f_3 | a \to b c) = p(g, f_2, f_3 | a \to b c)
\]

In addition, under the assumptions of the L-PCFG,

\[
p(g, f_2, f_3 | a \to b c) = \sum_{h_1, h_2, h_3} (p(h_1, h_2, h_3 | a \to b c)
\times p(g | h_1, a)p(f_2 | h_2, b)p(f_3 | h_3, c))
\]

Hence the maximum of Eq. 1 is achieved at

\[
\hat{t}(h_1, h_2, h_3 | a \to b c) = p(h_1, h_2, h_3 | a \to b c)
\]

because this gives \( \kappa(g, f_2, f_3) = p(g, f_2, f_3 | a \to b c) \). Under the strict convexity assumption the maximum of Eq. 1 is unique, hence the \( \hat{t} \) values must satisfy Eq. 2. Finally, it follows from Eq. 2, and the equality \( \hat{p}(a \to b c | a) = p(a \to b c | a) \), that Step 3 of the algorithm gives

\[
\hat{t}(a \to b c, h_2, h_3 | a, h_1) = t(a \to b c, h_2, h_3 | a, h_1).
\]

We can now see how the strict convexity assumption is needed. Without this assumption, there may be multiple settings for \( \hat{t} \) that achieve \( \kappa(g, f_2, f_3) = p(g, f_2, f_3 | a \to b c) \); the values \( \hat{t}(h_1, h_2, h_3 | a \to b c) = p(h_1, h_2, h_3 | a \to b c) \) will be included in this set of solutions, but other, inconsistent solutions will also be included.

As an extreme example of the failure of the strict convexity assumption, consider a feature-vector definition with \( |\mathcal{F}| = |\mathcal{G}| = 1 \). In this case the function in Eq. 1 reduces to

\[
\log \sum_{h_1, h_2, h_3} (\hat{t}(h_1, h_2, h_3 | a \to b c) \times p(g | h_1, a)p(f_2 | h_2, b)p(f_3 | h_3, c)).
\]

This function has a maximum value of 0, achieved at all values of \( \hat{t} \). Intuitively, this definition of inside and outside tree features loses all information about the latent states, and does not allow successful learning of the underlying L-PCFG. More generally, it is clear that the strict convexity assumption will depend directly on the choice of feature functions \( \tau(t) \) and \( \rho(o) \).

**Remark:** The infinite data assumption, and sample complexity. The infinite data assumption deserves more discussion. It is clearly a strong assumption that there is sufficient data for
the estimates $\hat{p}$ in assumption 3 to have converged to the correct underlying values. A more detailed analysis of the algorithm would derive sample complexity results, giving guarantees on the sample size $M$ required to reach a level of accuracy $\epsilon$ in the estimates, with probability at least $1 - \delta$, as a function of $\epsilon$, $\delta$, and other relevant quantities such as $n, d, d', m, \alpha, \beta$ and so on.

In spite of the strength of the infinite data assumption, we stress the importance of this result as a guarantee for the algorithm. First, a guarantee of correct parameter values in the limit of infinite data is typically the starting point for a sample complexity result (see for example (Hsu et al., 2009; Anandkumar et al., 2012)). Second, our sense is that a sample complexity result can be derived for our algorithm using standard methods: specifically, the analysis in (Arora et al., 2012) gives one set of guarantees; the remaining optimization problems we solve are convex maximum-likelihood problems, which are also relatively easy to analyze. Note that several pieces of previous work on spectral methods for latent-variable models focus on algorithms that are correct under the infinite data assumption.

5 The Matrix Decomposition Algorithm

This section describes the matrix decomposition algorithm used in Step 1 of the learning algorithm.

5.1 Problem Setting

Our goal will be to solve the following matrix decomposition problem:

**Matrix Decomposition Problem (MDP)** 1. Design an algorithm with the following inputs, assumptions, and outputs:
The parameters

inputs:

sumptions, and outputs:

sign an algorithm with the following inputs, as-

decomposition problem (MDP 2).

de-

ties of this algorithm is that it solves the following

of our algorithm for MDP 1. One of the proper-

This section describes a variant of the algorithm of

5.2 The Algorithm of Arora et al. (2013)

solves MDP 2 in some sense solves

one half” of MDP 1.

For completeness we give a sketch of the algo-

rithm that we use; it is inspired by the algorithm

of Arora et al. (2012), but has some important dif-

ferences. The algorithm is as follows:

Step 1: Derive a function \( \phi : \{d'\} \to \mathbb{R}^d \) that

maps each integer \( g \in \{d'\} \) to a representation

\( \phi(g) \in \mathbb{R}^l \). The integer \( l \) is typically much smaller than

\( d' \), implying that the representation is of low

dimension. Arora et al. (2012) derive \( \phi \) as a ran-

don projection with a carefully chosen dimension

\( l \). In our experiments, we use canonical correlation

analysis (CCA) on the matrix \( Q \) to give a representa-

tion \( \phi(g) \in \mathbb{R}^l \) where \( l = m \).

Step 2: For each \( f \in \{d\} \), calculate

\[
  v_f = \mathbb{E}[\phi(g) \mid f] = \sum_{g=1}^{d'} p(g \mid f)\phi(g)
\]

where %Math Processing Error% is equal to \( \sum_{g=1}^{d'} p(g \mid f)\phi(g) \) .

Hence the \( v_f \) vectors lie in the convex hull of a

set of vectors \( w_1 \ldots w_m \in \mathbb{R}^l \). Crucially, for any

pivot word \( f \) for latent state \( h \), we have \( p(h \mid f) = 1 \),

hence \( v_f = w_h \). Thus by the pivot assump-

tion, the set of points \( v_1 \ldots v_d \) includes the ver-

tices of the convex hull. Each point \( v_j \) is a convex

combination of the vertices \( w_1 \ldots w_m \), where the

weights in this combination are equal to \( p(h \mid j) \).

Step 3: Use the FastAnchorWords algo-

rithm of (Arora et al., 2012) to identify \( m \) vectors

\( v_{s_1} \ldots v_{s_m} \). The FastAnchorWords algo-

rithm has the guarantee that there is a permutation

\( \sigma : \{m\} \to \{m\} \) such that \( v_{s_i} = w_{\sigma(i)} \) for all \( i \).

This algorithm recovers the vertices of the convex

hull described in step 2, using a method that greedily

picks points that are as far as possible from the

subspace spanned by previously picked points.

Step 4: For each \( f \in \{d\} \) solve the problem

\[
  \arg \min_{\gamma_1, \gamma_2, \ldots, \gamma_m} \| \sum_h \gamma_h v_{s_h} - v_f \|^2
\]

subject to \( \gamma_h \geq 0 \) and \( \sum_h \gamma_h = 1 \). We use the

algorithm of (Frank and Wolfe, 1956; Clarkson,

2010) for this purpose. Set \( q(h \mid f) = \gamma_h \).
Return the final quantities:

\[ \hat{\pi}(h) = \sum_f p(f) q(h | f) \quad \hat{r}(f | h) = \frac{p(f) q(h | f)}{\sum_f p(f) q(h | f)} \]

where \( p(f) = \sum_g Q_{f,g} \).

5.3 An Algorithm for MDP 1

Figure 2 shows an algorithm that solves MDP 1. In steps 1 and 2 of the algorithm, the algorithm of section 5.2 is used to recover estimates \( \hat{r}(f \mid h) \) and \( \hat{s}(g \mid h) \). These distributions are equal to \( p(f \mid h) \) and \( p(g \mid h) \) up to permutations \( \sigma \) and \( \sigma' \) of the latent states respectively; unfortunately there is no guarantee that \( \sigma \) and \( \sigma' \) are the same permutation. Step 3 estimates parameters \( h(t' \mid h) \) that effectively map the permutation implied by \( \hat{r}(f \mid h) \) to the permutation implied by \( \hat{s}(g \mid h) \); the latter distribution is recalculated as \( \sum_{h'} h(t' \mid h) \hat{s}(g \mid h') \).

We now state the following theorem:

**Theorem 2.** The algorithm in figure 2 solves Matrix Decomposition Problem 1.

**Proof:** See the appendix.

**Remark:** A natural alternative to the algorithm presented would be to run Step 1 of the original algorithm, but to replace steps 2 and 3 with a step that finds \( \hat{s}(g \mid h) \) values that maximize

\[ \sum_{f,g} Q_{f,g} \log \left( \sum_{h} \hat{r}(h \mid f) \hat{s}(g \mid h) \right) \]

This is again a convex optimization problem. We may explore this algorithm in future work.

6 Additional Details of the Algorithm

6.1 Recovery of the \( \pi \) and \( q \) Parameters

The recovery of the \( \pi \) and \( q \) parameters relies on the following additional (but benign) assumptions on the functions \( \tau \) and \( \rho \):

1) For any inside tree \( t \) such that \( t \) is a unary rule of the form \( a \rightarrow x \), the function \( \tau \) is defined as \( \tau(t) = t \).

2) The set of outside tree features \( \mathcal{G} \) contains a special symbol \( \square \), and \( g(o) = \square \) if and only if the outside tree \( o \) is derived from a non-terminal node at the root of a skeletal tree.

\[ \hat{\pi}(h) = \sum_f p(f) q(h | f) \quad \hat{r}(f | h) = \frac{p(f) q(h | f)}{\sum_f p(f) q(h | f)} \]

\[ \sum_{f,g} Q_{f,g} \log \left( \sum_{h} \hat{r}(h \mid f) \hat{s}(g \mid h) \right) \]

\[ \hat{\pi}(h) = \sum_f p(f) q(h | f) \quad \hat{r}(f | h) = \frac{p(f) q(h | f)}{\sum_f p(f) q(h | f)} \]

where \( p(f) = \sum_g Q_{f,g} \).

**Inputs:** As in Matrix Decomposition Problem 1.

**Assumptions:** As in Matrix Decomposition Problem 1.

**Algorithm:**

**Step 1.** Run the algorithm of section 5.2 on the matrix \( Q \) to derive estimates \( \hat{r}(f \mid h) \) and \( \hat{\pi}(h) \). Note that under the guarantees of the algorithm, there is some permutation \( \sigma \) such that \( \hat{r}(f \mid h) = r(f \mid \sigma(h)) \). Define

\[ \hat{r}(h \mid f) = \frac{\hat{r}(f \mid h) \hat{\pi}(h)}{\sum_h \hat{r}(f \mid h) \hat{\pi}(h)} \]

**Step 2.** Run the algorithm of section 5.2 on the matrix \( Q^\top \) to derive estimates \( \hat{s}(g \mid h) \). Under the guarantees of the algorithm, there is some permutation \( \sigma' \) such that \( \hat{s}(g \mid h) = s(g \mid \sigma'(h)) \). Note however that it is not necessarily the case that \( \sigma = \sigma' \).

**Step 3.** Find \( \hat{i}(h' \mid h) \) for all \( h, h' \in [m] \) that maximize

\[ \sum_{f,g} Q_{f,g} \log \sum_{h,h'} \hat{r}(h \mid f) \hat{i}(h' \mid h) \hat{s}(g \mid h') \] (3)

subject to \( \hat{i}(h' \mid h) \geq 0 \), and \( \forall h, \sum_{h'} \hat{i}(h' \mid h) = 1 \).

**Remark:** the function in Eq. 3 is concave in the \( i \) parameters. We use the EM algorithm to find a global optimum.

**Step 4.** Return the following values:

- \( \hat{\pi}(h) \) for all \( h \), as an estimate of \( \pi(\sigma(h)) \) for some permutation \( \sigma \).
- \( \hat{r}(f \mid h) \) for all \( f, h \) as an estimate of \( r(f \mid \sigma(h)) \) for the same permutation \( \sigma \).
- \( \sum_{h'} \hat{i}(h' \mid h) \hat{s}(g \mid h') \) as an estimate of \( s(f \mid \sigma(h)) \) for the same permutation \( \sigma \).

Under these assumptions, the algorithm in figure 1 recovers estimates \( \hat{\pi}(a, h) \) and \( \hat{q}(a \rightarrow x \mid a, h) \). Simply set

\[ \hat{q}(a \rightarrow x \mid a, h) = \hat{r}(f \mid h, a) \]

and \( \hat{\pi}(a, h) = \hat{p}(\square, h, a) / \sum_{h,a} \hat{p}(\square, h, a) \) where \( \hat{p}(\square, h, a) = \hat{q}(\square \mid h, a) \hat{p}(h \mid a) \hat{p}(a) \). Note that \( \hat{p}(h \mid a) \) can be derived from the matrix decomposition step when applied to \( \hat{Q}^a \), and \( \hat{p}(a) \) is easily recovered from the training examples.

6.2 Extension to Include Multiple Features

We now describe an extension to allow \( K \) separate functions \( \tau^{(k)}(t) \) for \( k = 1 \ldots K \) mapping inside trees to features, and \( L \) feature functions \( \rho^{(l)}(a) \) for \( l = 1 \ldots L \) over outside trees.

The algorithm in figure 1 can be extended as follows. First, Step 1 of the algorithm (the matrix
decomposition step) can be extended to provide estimates \( \hat{r}^{(k)}(f^{(k)} | h, a) \) and \( \hat{s}^{(l)}(g^{(l)} | h, a) \). In brief, this involves running CCA on a matrix
\[
\mathbf{E}[\phi(T)(\psi(O))^\top | A = a]
\]
where \( \phi \) and \( \psi \) are inside and outside binary feature vectors derived directly from the inside and outside features, using a one-hot representation. CCA results in a low-dimensional representation that can be used in the steps described in section 5.2; the remainder of the algorithm is the same. In practice, the addition of multiple features may lead to better CCA representations.

Next, we modify the objective function in Eq. 1 to be the following:
\[
\sum_{i,j,k} \sum_{g^i, f^j, k^k} p(g^i, f^j, k^k | a \rightarrow b c) \log \kappa^{i,j,k}(g^i, f^j, k^k)
\]
where
\[
\kappa^{i,j,k}(g^i, f^j, k^k) = \sum_{h_1, h_2, h_3} (\hat{s}^i(h_1, a) \hat{r}^j(f^j_2 | h_2, b) \hat{r}^k(f^k_3 | h_3, c))
\]
Thus the new objective function consists of a sum of \( L \times M^2 \) terms, each corresponding to a different combination of inside and outside features. The function remains concave.

6.3 Use as an Initializer for EM

The learning algorithm for L-PCFGs can be used as an initializer for the EM algorithm for L-PCFGs. Two-step estimation methods such as these are well known in statistics; there are guarantees for example that if the first estimator is consistent, and the second step finds the closest local maxima of the likelihood function, then the resulting estimator is both consistent and efficient (in terms of number of samples required). See for example page 453 or Theorem 4.3 (page 454) of (Lehmann and Casella, 1998).

7 Experiments on Parsing

This section describes parsing experiments using the learning algorithm for L-PCFGs. 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 was used as the test set.

The experimental setup is the same as described by Cohen et al. (2013). The trees are binarized (Petrov et al., 2006) and for the EM algorithm we use the initialization method described in Matsuzaki et al. (2005). For the pivot algorithm we use multiple features \( \tau^1(t) \ldots \tau^K(t) \) and \( \rho^1(o) \ldots \rho^L(o) \) over inside and outside trees, using the features described by Cohen et al. (2013).

Table 1 gives the F1 accuracy on the development and test sets for the following methods:

| Method      | sec. 22 | sec. 23 |
|-------------|---------|---------|
| EM          | 86.69   | 88.24   |
| Spectral    | 85.60   | 88.05   |
| Pivot       | 85.86   | 88.05   |
| Pivot+EM    | 86.83   | 88.03   |

Table 1: Results on the development data (section 22) and test data (section 23) for various learning algorithms for L-PCFGs. For EM and pivot+EM experiments, the second line denotes the number of iterations required to reach the given optimal performance on development data. Results for section 23 are used with the best model for section 22 in the corresponding row. The results for EM and spectral are reported from Cohen et al. (2013).

For the EM and Pivot+EM algorithms, we give the number of iterations of EM required to reach optimal performance on the development data.

The results show that the EM, Spectral, and Pivot+EM algorithms all perform at a very similar level of accuracy. The Pivot+EM results show that very few EM iterations—just 2 iterations in most conditions—are required to reach optimal performance when the Pivot model is used as an initializer for EM. The Pivot results lag behind the Pivot+EM results by around 2-3%, but they are close enough to optimality to require very few EM iterations when used as an initializer.

8 Experiments on the Saul and Pereira (1997) Model for Language Modeling

We now describe a second set of experiments, on the Saul and Pereira (1997) model for language modeling. Define \( V \) to be the set of words in the vocabulary. For any \( w_1, w_2 \in V \), the Saul and Pereira (1997) model then defines \( p(w_2 | w_1) = \sum_{h=1}^m r(h | w_1) s(w_2 | h) \) where \( r(h | w_1) \) and...
Table 2: Language model perplexity with the Brown corpus and the Gigaword corpus (New York Times portion) for the second half of the development set, and the test set. With EM and Pivot+EM, the number of iterations for EM to reach convergence is given below the perplexity. The best result for each column (for each \( m \) value) is in bold. The “test” column gives perplexity results on the test set. Each perplexity calculation on the test set is done using the best model on the development set. bi-KN+int and tri-KN+int are bigram and trigram Kneser-Ney interpolated models (Kneser and Ney, 1995), using the SRILM toolkit.

| \( m \) | Brown | \( m \) | NYT |
|---|---|---|---|
| Brown | NYT | Brown | NYT |
| \( m \) | 2 | 4 | 8 | 16 | 32 | 128 | 256 | test | 2 | 4 | 8 | 16 | 32 | 128 | 256 | test |
| EM | 737 | 599 | 488 | 468 | 430 | 388 | 365 | 364 | 926 | 733 | 562 | 420 | 361 | 284 | 265 | 267 |
| \( 12 \) | 14 | 14 | 19 | 12 | 10 | 9 | 8 | 36 | 36 | 39 | 42 | 33 | 38 | 35 | 32 | 271 |
| bi-KN+int. | 408 | 415 | 271 | 279 |
| tri-KN+int. | 386 | 394 | 150 | 158 |
| pivot | 582 | 510 | 425 | 374 | 310 | 327 | 357 | 502 | 510 | 425 | 374 | 310 | 327 | 357 |
| pivot+EM | 758 | 292 | 20 | 14 | 13 | 15 | 10 | 19 | 12 |

In this special case, the L-PCFG learning algorithm is equivalent to a simple algorithm, with the following steps: 1) define the matrix \( Q \) with entries \( Q_{w_1,w_2} = \text{count}(w_1,w_2)/N \) where \( \text{count}(w_1,w_2) \) is the number of times that bigram \((w_1,w_2)\) is seen in the data, and \( N = \sum_{w_1,w_2} \text{count}(w_1,w_2) \). Run the algorithm of section 5.2 on \( Q \) to recover estimates \( \hat{s}(w_2 | h) \); 2) estimate \( \hat{r}(h | w_1) \) using the EM algorithm to optimize the function \( \sum_{w_1,w_2} Q_{w_1,w_2} \log \sum_{h} \hat{r}(h | w_1) \hat{s}(w_2 | h) \) with respect to the \( \hat{r} \) parameters; this function is concave in these parameters.

We performed the language modeling experiments for a number of reasons. First, because in this case the L-PCFG algorithm reduces to a simple algorithm, it allows us to evaluate the core ideas in the method very directly. Second, it allows us to test the pivot method on the very large datasets that are available for language modeling.

We use two corpora for our experiments. The first is the Brown corpus, as used by Bengio et al. (2006) in language modeling experiments. Following Bengio et al. (2006), we use the first 800K words for training (and replace all words that appear once with an UNK token), the next 200K words for development, and the remaining data (165,171 tokens) as a test set. The size of the vocabulary is 24,488 words. The second corpus we use is the New York Times portion of the Gigaword corpus. Here, the training set consists of 1.31 billion tokens. We use 159 million tokens for development set and 156 million tokens for test. All words that appeared less than 20 times in the training set were replaced with the UNK token.

The size of the vocabulary is 235,223 words. Unknown words in test data are ignored when calculating perplexity (this is the standard set-up in the SRILM toolkit).

In our experiments we use the first half of each development set to optimize the number of iterations of the EM or Pivot+EM algorithms. As before, Pivot+EM uses 1 or more EM steps with parameter initialization from the Pivot method.

We have described a new algorithm for parameter estimation in L-PCFGs. The algorithm is provably correct, and performs well in practice when used in conjunction with EM.
Appendix: Proof of Theorem 2

Assume without loss of generality that \( \forall f, \sum_g Q_{f,g} > 0 \). If for some \( f \) we have \( \sum_g Q_{f,g} = 0 \), then the row corresponding to \( f \) can simply be removed from the matrix before running the algorithm.

Under the guarantees given by Arora et al. (2012), steps 1 and 2 recover parameters such that there are permutations \( \sigma \) and \( \sigma' \) such that \( \hat{r}(f \mid h) = r(f \mid \sigma(h)), \hat{\pi}(h) = \pi(\sigma(h)), \hat{r}(h \mid f) = r(\sigma(h) \mid f) \), and \( \hat{s}(g \mid h) = s(g \mid \sigma'(h)) \).

It follows that if we define \( \forall h, \hat{i}(\sigma'(h)\mid \sigma(h)) = 1 \), with all other values of \( \hat{i}(h' \mid h) \) equal to 0, this gives a maximizer of Eq. 3. To see this, note that for this definition of \( \hat{i} \) we have

\[
\sum_{h,h'} \hat{r}(h \mid f) \hat{i}(h' \mid h) \hat{s}(g \mid h') = \sum_h r(h \mid f) s(g \mid h) = p(g \mid f)
\]

where \( p(g \mid f) = \frac{Q_{f,g}}{\sum_g Q_{f,g}} \). By standard results from cross entropy, the maximizer of

\[
\sum_{f,g} Q_{f,g} \log p(g \mid f)
\]

over all conditional distributions \( p(g \mid f) \) is given by \( p(g \mid f) = \frac{Q_{f,g}}{\sum_g Q_{f,g}} \), hence this setting for the \( \hat{i} \) parameters is a maximizer.

It follows that any solution to the optimization problem in Eq. 3 (the function may have multiple global maxima) must satisfy

\[
\sum_{h,h'} \hat{r}(h \mid f) \hat{i}(h' \mid h) \hat{s}(g \mid h') = p(g \mid f)
\]

Hence if we define

\[
q(g \mid h) = \sum_{h'} \hat{i}(h' \mid h) \hat{s}(g \mid h')
\]

then

\[
\sum_h \hat{r}(h \mid f) q(g \mid h) = p(g \mid f)
\]

Substituting

\[
\hat{r}(h \mid f) = \frac{\hat{r}(f \mid h) \hat{\pi}(h)}{p(f)}
\]

(note that \( p(f) = \sum_g Q_{f,g} = \sum_h \hat{r}(f \mid h) \hat{\pi}(h) \)), and rearranging terms, gives

\[
\sum_h \hat{r}(f \mid h) \hat{\pi}(h) q(g \mid h) = Q_{f,g}
\]

The final output from the algorithm is the parameters \( \hat{r}(f \mid h), \hat{\pi}(h), \) and \( q(g \mid h) \), so this proves the theorem.

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. Ge, and A. Moitra. 2012. Learning topic models—going beyond SVD. In Proceedings of FOCS.

S. Arora, R. Ge, Y. Halpern, D. M. Mimno, A. Moitra, D. Sontag, Y. Wu, and M. Zhu. 2013. A practical algorithm for topic modeling with provable guarantees. In Proceedings of ICML.

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.

Y. Bengio, H. Schwenk, J.-S. Senécal, F. Morin, and J.-L. Gauvain. 2006. Neural probabilistic language models. In Innovations in Machine Learning, pages 137–186. Springer.

K. L. Clarkson. 2010. Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. ACM Transactions on Algorithms (TALG), 6(4):63.

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

S. B. Cohen, K. Stratos, M. Collins, D. P. Foster, and L. Ungar. 2013. Experiments with spectral learning of latent-variable PCFGs. In Proceedings of NAAACL.

A. Dempster, N. Laird, and D. Rubin. 1977. Maximum likelihood estimation from incomplete data via the EM algorithm. Journal of the Royal Statistical Society B, 39:1–38.

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

M. Frank and P. Wolfe. 1956. An algorithm for quadratic programming. Naval research logistics quarterly, 3(1-2):95–110.
H. Hotelling. 1936. Relations between two sets of variates. *Biometrika*, 28(3/4):321–377.

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

R. Kneser and H. Ney. 1995. Improved back-off for m-gram language modeling. In *Proceedings of ICASSP*.

E. L. Lehmann and G. Casella. 1998. *Theory of Point Estimation (Second edition)*. Springer.

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.

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 ICML*.

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

L. Saul and F. Pereira. 1997. Aggregate and mixed-order markov models for statistical language processing. In *Proceedings of EMNLP*.

S. Siddiqi, B. Boots, and G. Gordon. 2010. Reduced-rank hidden markov models. *Journal of Machine Learning Research*, 9:741–748.