Spectral Unsupervised Parsing with Additive Tree Metrics

Ankur P. Parikh
School of Computer Science
Carnegie Mellon University
apparikh@cs.cmu.edu

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

Eric P. Xing
School of Computer Science
Carnegie Mellon University
epxing@cs.cmu.edu

Abstract

We propose a spectral approach for unsupervised constituent parsing that comes with theoretical guarantees on latent structure recovery. Our approach is grammarless – we directly learn the bracketing structure of a given sentence without using a grammar model. The main algorithm is based on lifting the concept of additive tree metrics for structure learning of latent trees in the phylogenetic and machine learning communities to the case where the tree structure varies across examples. Although finding the "minimal" latent tree is NP-hard in general, for the case of projective trees we find that it can be found using bilexical parsing algorithms. Empirically, our algorithm performs favorably compared to the constituent context model of Klein and Manning (2002) without the need for careful initialization.

1 Introduction

Solutions to the problem of grammar induction have been long sought after since the early days of computational linguistics and are interesting both from cognitive and engineering perspectives. Cognitively, it is more plausible to assume that children obtain only terminal strings of parse trees and not the actual parse trees. This means the unsupervised setting is a better model for studying language acquisition. From the engineering perspective, training data for unsupervised parsing exists in abundance (i.e. sentences and part-of-speech tags), and is much cheaper than the syntactically annotated data required for supervised training.

Most existing solutions treat the problem of unsupervised parsing by assuming a generative process over parse trees e.g. probabilistic context free grammars (Jelinek et al., 1992), and the constituent context model (Klein and Manning, 2002). Learning then reduces to finding a set of parameters that are estimated by identifying a local maximum of an objective function such as the likelihood (Klein and Manning, 2002) or a variant of it (Smith and Eisner, 2005; Cohen and Smith, 2009; Headden et al., 2009; Spitkovsky et al., 2010b; Gillenwater et al., 2010; Golland et al., 2012). Unfortunately, finding the global maximum for these objective functions is usually intractable (Cohen and Smith, 2012) which often leads to severe local optima problems (but see Gormley and Eisner, 2013). Thus, strong experimental results are often achieved by initialization techniques (Klein and Manning, 2002; Gimpel and Smith, 2012), incremental dataset use (Spitkovsky et al., 2010a) and other specialized techniques to avoid local optima such as count transforms (Spitkovsky et al., 2013). These approaches, while empirically promising, generally lack theoretical justification.

On the other hand, recently proposed spectral methods approach the problem via restriction of the PCFG model (Hsu et al., 2012) or matrix completion (Bailly et al., 2013). These novel perspectives offer strong theoretical guarantees but are not designed to achieve competitive empirical results.

In this paper, we suggest a different approach, to provide a first step to bridging this theory-experiment gap. More specifically, we approach unsupervised constituent parsing from the perspective of *structure learning* as opposed to parameter learning. We associate each sentence with an undirected latent tree graphical model, which is a tree consisting of both observed variables (corresponding to the words in the sentence) and an additional set of latent variables that are unobserved in the data. This undirected latent tree is then directed via a *direction mapping* to give the final constituent parse.

In our framework, parsing reduces to finding the best latent structure for a given sentence. However, due to the presence of latent variables, structure learning of latent trees is substantially more complicated than in observed models. As before, one solution would be local search heuristics.

Intuitively, however, latent tree models encode low rank dependencies among the observed variables permitting the development of "spec-

Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, pages 1062–1072,
Baltimore, Maryland, USA, June 23-25 2014.

tral" methods that can lead to provably correct solutions. In particular we leverage the concept of additive tree metrics (Buneman, 1971; Buneman, 1974) in phylogenetics and machine learning that can create a special distance metric among the observed variables as a function of the underlying spectral dependencies (Choi et al., 2011; Song et al., 2011; Anandkumar et al., 2011; Ishteva et al., 2012). Additive tree metrics can be leveraged by "meta-algorithms" such as neighbor-joining (Saitou and Nei, 1987) and recursive grouping (Choi et al., 2011) to provide consistent learning algorithms for latent trees.

Moreover, we show that it is desirable to learn the "minimal" latent tree based on the tree metric ("minimum evolution" in phylogenetics). While this criterion is in general NP-hard (Desper and Gascuel, 2005), for projective trees we find that a bilexical parsing algorithm can be used to find an exact solution efficiently (Eisner and Satta, 1999).

Unlike in phylogenetics and graphical models, where a single latent tree is constructed for all the data, in our case, each part of speech sequence is associated with its own parse tree. This leads to a severe data sparsity problem even for moderately long sentences. To handle this issue, we present a strategy that is inspired by ideas from kernel smoothing in the statistics community (Zhou et al., 2010; Kolar et al., 2010b; Kolar et al., 2010a). This allows principled sharing of samples from different but similar underlying distributions.

We provide theoretical guarantees on the recovery of the correct underlying latent tree and characterize the associated sample complexity under our technique. Empirically we evaluate our method on data in English, German and Chinese. Our algorithm performs favorably to Klein and Manning's (2002) constituent-context model (CCM), without the need for careful initialization. In addition, we also analyze CCM's sensitivity to initialization, and compare our results to Seginer's algorithm (Seginer, 2007).

2 Learning Setting and Model

In this section, we detail the learning setting and a conditional tree model we learn the structure for.

2.1 Learning Setting

Let $\boldsymbol{w} = (w_1, ..., w_\ell)$ be a vector of words corresponding to a sentence of length ℓ. Each w_i is represented by a vector in $\mathbb{R}^p$ for $p \in \mathbb{N}$. The vector is an embedding of the word in some space, chosen from a fixed dictionary that maps word types to $\mathbb{R}^p$. In addition, let $\boldsymbol{x} = (x_1, ..., x_\ell)$ be the associated vector of part-of-speech (POS) tags (i.e. x_i is the POS tag of w_i).

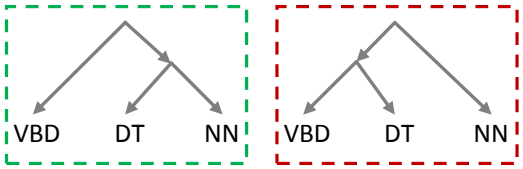

Figure 2: Candidate constituent parses for $\boldsymbol{x} = (\texttt{VBD}, \texttt{DT}, \texttt{NN})$ (left-correct, right -incorrect)

In our learning algorithm, we assume that examples of the form $(\boldsymbol{w}^{(i)}, \boldsymbol{x}^{(i)})$ for $i \in [N] = \{1, \ldots, N\}$ are given, and the goal is to predict a bracketing parse tree for each of these examples. The word embeddings are used during the learning process, but the final decoder that the learning algorithm outputs maps a POS tag sequence $\boldsymbol{x}$ to a parse tree. While ideally we would want to use the word information in decoding as well, much of the syntax of a sentence is determined by the POS tags, and relatively high level of accuracy can be achieved by learning, for example, a supervised parser from POS tag sequences.

Just like our decoder, our model assumes that the bracketing of a given sentence is a function of its POS tags. The POS tags are generated from some distribution, followed by a deterministic generation of the bracketing parse tree. Then, latent states are generated for each bracket, and finally, the latent states at the yield of the bracketing parse tree generate the words of the sentence (in the form of embeddings). The latent states are represented by vectors $z \in \mathbb{R}^m$ where $m < p$.

2.2 Intuition

For intuition, consider the simple tag sequence $\boldsymbol{x} = (\texttt{VBD}, \texttt{DT}, \texttt{NN})$. Two candidate constituent parse structures are shown in Figure 2 and the correct one is boxed in green (the other in red). Recall that our training data contains word phrases that have the tag sequence $\boldsymbol{x}$ e.g. $\boldsymbol{w}^{(1)} = (\texttt{hit}, \texttt{the}, \texttt{ball})$, $\boldsymbol{w}^{(2)} = (\texttt{ate}, \texttt{an}, \texttt{apple})$.

Intuitively, the words in the above phrases exhibit dependencies that can reveal the parse structure. The determiner (w_2) and the direct object (w_3) are correlated in that the choice of determiner depends on the plurality of w_3. However, the choice of verb (w_1) is mostly independent of the determiner. We could thus conclude that w_2 and w_3 should be closer in the parse tree than w_1

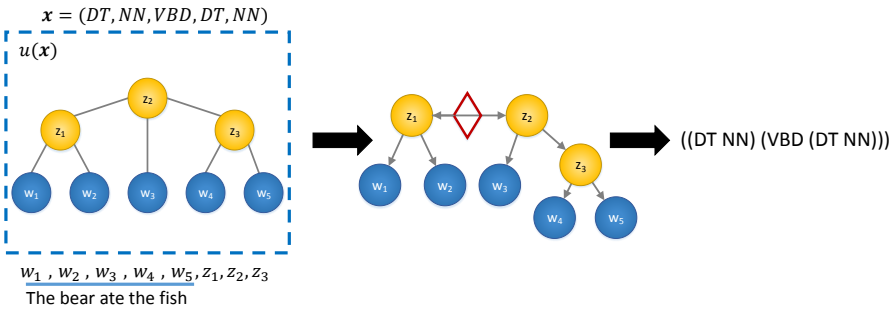

Figure 1: Example for the tag sequence (DT, NN, VBD, DT, NN) showing the overview of our approach. We first learn a undirected latent tree for the sequence (left). We then apply a direction mapping h_{dir} to direct the latent tree (center). This can then easily be converted into a bracketing (right).

and w_2, giving us the correct structure. Informally, the latent state z corresponding to the (w_2, w_3) bracket would store information about the plurality of z, the key to the dependence between w_2 and w_3. It would then be reasonable to assume that w_2 and w_3 are independent given z.

2.3 A Conditional Latent Tree Model

Following this intuition, we propose to model the distribution over the latent bracketing states and words for each tag sequence $\boldsymbol{x}$ as a latent tree graphical model, which encodes conditional independences among the words given the latent states.

Let $\mathcal{V} := \{w_1, ..., w_\ell, z_1, ..., z_H\}$, with w_i representing the word embeddings, and z_i representing the latent states of the bracketings. Then, according to our base model it holds that:

$$p(\boldsymbol{w}, \boldsymbol{z}|\boldsymbol{x}) = \prod_{i=1}^{H} p(z_i|\pi_{\boldsymbol{x}}(z_i), \theta(\boldsymbol{x})) \times \prod_{i=1}^{\ell(\boldsymbol{x})} p(w_i|\pi_{\boldsymbol{x}}(w_i), \theta(\boldsymbol{x})) \quad (1)$$

where $\pi_{\boldsymbol{x}}(\cdot)$ returns the parent node index of the argument in the latent tree corresponding to tag sequence $\boldsymbol{x}$.[1] If z is the root, then $\pi_{\boldsymbol{x}}(z) = \emptyset$. All the w_i are assumed to be leaves while all the z_i are internal (i.e. non-leaf) nodes. The parameters $\theta(\boldsymbol{x})$ control the conditional probability tables. We do not commit to a certain parametric family, but see more about the assumptions we make about θ in §3.2. The parameter space is denoted Θ. The model assumes a factorization according to a latent-variable tree. The latent variables can incorporate various linguistic properties, such as head information, valence of dependency being generated, and so on. This information is expected to be learned automatically from data.

Our generative model deterministically maps a POS sequence to a bracketing via an undirected latent-variable tree. The orientation of the tree is determined by a *direction mapping* $h_{\text{dir}}(u)$, which is fixed during learning and decoding. This means our decoder first identifies (given a POS sequence) an undirected tree, and then orients it by applying h_{dir} on the resulting tree (see below).

Define $\mathcal{U}$ to be the set of undirected latent trees where all internal nodes have degree exactly 3 (i.e. they correspond to binary bracketing), and in addition $h_{\text{dir}}(u)$ for any $u \in \mathcal{U}$ is projective (explained in the h_{dir} section). In addition, let $\mathcal{T}$ be the set of binary bracketings. The complete generative model that we follow is then:

- Generate a tag sequence $\boldsymbol{x} = (x_1, \ldots, x_\ell)$
- Decide on $u(\boldsymbol{x}) \in \mathcal{U}$, the undirected latent tree that $\boldsymbol{x}$ maps to.
- Set $t \in \mathcal{T}$ by computing $t = h_{\text{dir}}(u)$.
- Set $\theta \in \Theta$ by computing $\theta = \theta(\boldsymbol{x})$.
- Generate a tuple $\boldsymbol{v} = (w_1, \ldots, w_\ell, z_1, ..., z_H)$ where $w_i \in \mathbb{R}^p, z_j \in \mathbb{R}^m$ according to Eq. 1.

See Figure 1 (left) for an example.

The Direction Mapping h_{dir}. Generating a bracketing via an undirected tree enables us to build on existing methods for structure learning of latent-tree graphical models (Choi et al., 2011; Anandkumar et al., 2011). Our learning algorithm focuses on recovering the undirected tree based for the generative model that was described above. This undirected tree is converted into a directed tree by applying h_{dir}. The mapping h_{dir} works in three steps:

- It first chooses a top bracket $([1, R-1], [R, \ell])$ where R is the mid-point of the bracket and ℓ is the length of the sentence.
- It marks the edge $e_{i,j}$ that splits the tree according to the top bracket as the "root edge" (marked in red in Figure 1(center))
- It then creates t from u by directing the tree outward from $e_{i,j}$ as shown in Figure 1(center)

[1] At this point, π refers to an arbitrary direction of the undirected tree $u(\boldsymbol{x})$.

The resulting t is a binary bracketing parse tree. As implied by the above definition of h_{dir}, selecting which edge is the root can be interpreted as determining the top bracket of the constituent parse. For example, in Figure 1, the top bracket is $([1,2],[3,5]) = ([\texttt{DT},\texttt{NN}],[\texttt{VBD},\texttt{DT},\texttt{NN}])$. Note that the "root" edge e_{z_1,z_2} partitions the leaves into precisely this bracketing. As indicated in the above section, we restrict the set of undirected trees to be those such that after applying h_{dir} the resulting t is projective i.e. there are no crossing brackets. In §4.1, we discuss an effective heuristic to find the top bracket without supervision.

3 Spectral Learning Algorithm based on Additive Tree Metrics

Our goal is to recover $t \in \mathcal{T}$ for tag sequence $\boldsymbol{x}$ using the data $\mathcal{D} = [(\boldsymbol{w}^{(i)}, \boldsymbol{x}^{(i)})]_{i=1}^{N}$. To get an intuition about the algorithm, consider a partition of the set of examples $\mathcal{D}$ into $\mathcal{D}(\boldsymbol{x}) = \{(\boldsymbol{w}^{(i)}, \boldsymbol{x}^{(i)}) \in \mathcal{D} | \boldsymbol{x}^{(i)} = \boldsymbol{x}\}$, i.e. each section in the partition has an identical sequence of part of speech tags. Assume for this section $|\mathcal{D}(\boldsymbol{x})|$ is large (we address the data sparsity issue in §3.4).

We can then proceed by learning how to map a POS sequence $\boldsymbol{x}$ to a tree $t \in \mathcal{T}$ (through $u \in \mathcal{U}$) by focusing only on examples in $\mathcal{D}(\boldsymbol{x})$.

Directly attempting to maximize the likelihood unfortunately results in an intractable optimization problem and greedy heuristics are often employed (Harmeling and Williams, 2011). Instead we propose a method that is provably consistent and returns a tree that can be mapped to a bracketing using h_{dir}.

If all the variables were observed, then the Chow-Liu algorithm (Chow and Liu, 1968) could be used to find the most likely tree structure $u \in \mathcal{U}$. The Chow-Liu algorithm essentially computes the distances among all pairs of variables (the negative of the mutual information) and then finds the minimum cost tree. However, the fact that the z_i are latent variables makes this strategy substantially more complicated. In particular, it becomes challenging to compute the distances among pairs of latent variables. What is needed is a "special" distance function that allows us to reverse engineer the distances among the latent variables given the distances among the observed variables. This is the key idea behind additive tree metrics that are the basis of our approach.

In the following sections, we describe the key steps to our method. §3.1 and §3.2 largely describe existing background on additive tree metrics and latent tree structure learning, while §3.3 and §3.4 discuss novel aspects that are unique to our problem.

3.1 Additive Tree Metrics

Let $u(\boldsymbol{x})$ be the true undirected tree of sentence $\boldsymbol{x}$ and assume the nodes $\mathcal{V}$ to be indexed by $[M] = \{1, \ldots, M\}$ such that $M = |\mathcal{V}| = H + \ell$. Furthermore, let $v \in \mathcal{V}$ refer to a node in the undirected tree (either observed or latent). We assume the existence of a distance function that allows us to compute distances between pairs of nodes. For example, as we see in §3.2 we will define the distance $d(i,j)$ to be a function of the covariance matrix $\mathbb{E}[v_i v_j^\top | u(\boldsymbol{x}), \theta(\boldsymbol{x})]$. Thus if v_i and v_j are both observed variables, the distance can be directly computed from the data.

Moreover, the metrics we construct are such that they are *tree additive*, defined below:

Definition 1 *A function $d_{u(\boldsymbol{x})} : [M] \times [M] \to \mathbb{R}$ is an* additive tree metric *(Erdős et al., 1999) for the undirected tree $u(\boldsymbol{x})$ if it is a distance metric,[2] and furthermore, $\forall i, j \in [M]$ the following relation holds:*

$$d_{u(\boldsymbol{x})}(i,j) = \sum_{(a,b) \in \text{path}_{u(\boldsymbol{x})}(i,j)} d_{u(\boldsymbol{x})}(a,b) \quad (2)$$

where $\text{path}_{u(\boldsymbol{x})}(i,j)$ is the set of all the edges in the (undirected) path from i to j in the tree $u(\boldsymbol{x})$.

As we describe below, given the tree structure, the additive tree metric property allows us to compute "backwards" the distances among the latent variables as a function of the distances among the observed variables.

Define $\boldsymbol{D}$ to be the $M \times M$ distance matrix among the M variables, i.e. $D_{ij} = d_{u(\boldsymbol{x})}(i,j)$. Let $\boldsymbol{D}_{WW}$, $\boldsymbol{D}_{ZW}$ (equal to $\boldsymbol{D}_{WZ}^\top$), and $\boldsymbol{D}_{ZZ}$ indicate the word-word, latent-word and latent-latent sub-blocks of $\boldsymbol{D}$ respectively. In addition, since $u(\boldsymbol{x})$ is assumed to be known from context, we denote $d_{u(\boldsymbol{x})}(i,j)$ just by $d(i,j)$.

Given the fact that the distance between a pair of nodes is a function of the random variables they represent (according to the true model), only $\boldsymbol{D}_{WW}$ can be empirically estimated from data. However, if the underlying tree structure is known, then Definition 1 can be leveraged to compute $\boldsymbol{D}_{ZZ}$ and $\boldsymbol{D}_{ZW}$ as we show below.

[2] This means that it satisfies $d(i,j) = 0$ if and only if $i = j$, the triangle inequality and is also symmetric.

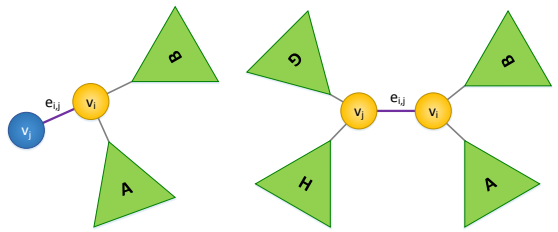

Figure 3: Two types of edges in general undirected latent trees. (a) leaf edge, (b) internal edge

We first show how to compute $d(i,j)$ for all i,j such that i and j are adjacent to each other in $u(\boldsymbol{x})$, based only on observed nodes. It then follows that the other elements of the distance matrix can be computed based on Definition 1. To show how to compute distances between adjacent nodes, consider the two cases: **(1)** (i,j) is a leaf edge; **(2)** (i,j) is an internal edge.

Case 1 (leaf edge, figure 3(a)) Assume without loss of generality that j is the leaf and i is an internal latent node. Then i must have exactly two other neighbors $a \in [M]$ and $b \in [M]$. Let A denote the set of nodes that are closer to a than i and similarly let B denote the set of nodes that are closer to b than i. Let A^* and B^* denote all the leaves (word nodes) in A and B respectively. Then using path additivity (Definition 1), it can be shown that for any $a^* \in A^*, b^* \in B^*$ it holds that:

$$d(i,j) = \frac{1}{2}\left(d(j,a^*) + d(j,b^*) - d(a^*,b^*)\right) \quad (3)$$

Note that the right-hand side only depends on distances between observed random variables.

Case 2 (internal edge, figure 3(b)) Both i and j are internal nodes. In this case, i has exactly two other neighbors $a \in [M]$ and $b \in [M]$, and similarly, j has exactly other two neighbors $g \in [M]$ and $h \in [M]$. Let A denote the set of nodes closer to a than i, and analogously for B, G, and H. Let A^*, B^*, G^*, and H^* refer to the leaves in A, B, G, and H respectively. Then for any $a^* \in A^*$, $b^* \in B^*$, $g^* \in G^*$, and $h^* \in H^*$ it can be shown that:

$$d(i,j) = \tfrac{1}{4}\Big(d(a^*,g^*) + d(a^*,h^*) + d(b^*,g^*) + d(b^*,h^*) - 2d(a^*,b^*) - 2d(g^*,h^*)\Big) \quad (4)$$

Empirically, one can obtain a more robust empirical estimate $\widehat{d}(i,j)$ by averaging over all valid choices of a^*, b^* in Eq. 3 and all valid choices of a^*, b^*, g^*, h^* in Eq. 4 (Desper and Gascuel, 2005).

3.2 Constructing a Spectral Additive Metric

In constructing our distance metric, we begin with the following assumption on the distribution in Eq. 1 (analogous to the assumptions made in Anandkumar et al., 2011).

Assumption 1 (Linear, Rank m, Means)

$$\mathbb{E}[z_i|\pi_{\boldsymbol{x}}(z_i), \boldsymbol{x}] = \boldsymbol{A}_{(z_i|z_{\pi_{\boldsymbol{x}}(z_i)},\boldsymbol{x})}\pi_{\boldsymbol{x}}(z_i)\ \forall i \in [H]$$

where $\boldsymbol{A}_{(z_i|\pi_{\boldsymbol{x}}(z_i),\boldsymbol{x})} \in \mathbb{R}^{m\times m}$ *has rank* m.

$$\mathbb{E}[w_i|\pi_{\boldsymbol{x}}(w_i), \boldsymbol{x}] = \boldsymbol{C}_{(w_i|\pi_{\boldsymbol{x}}(w_i),\boldsymbol{x})}\pi_{\boldsymbol{x}}(w_i)\ \forall i \in [\ell(\boldsymbol{x})]$$

where $\boldsymbol{C}_{(w_i|\pi_{\boldsymbol{x}}(w_i),\boldsymbol{x})} \in \mathbb{R}^{p\times m}$ *has rank* m.

Also assume that $\mathbb{E}[z_i z_i^\top|\boldsymbol{x}]$ *has rank* $m\ \forall i \in [H]$.

Note that the matrices $\boldsymbol{A}$ and $\boldsymbol{C}$ are a direct function of $\theta(\boldsymbol{x})$, but we do not specify a model family for $\theta(\boldsymbol{x})$. The only restriction is in the form of the above assumption. If w_i and z_i were discrete, represented as binary vectors, the above assumption would correspond to requiring all conditional probability tables in the latent tree to have rank m. Assumption 1 allows for the w_i to be high dimensional features, as long as the expectation requirement above is satisfied. Similar assumptions are made with spectral parameter learning methods e.g. Hsu et al. (2009), Bailly et al. (2009), Parikh et al. (2011), and Cohen et al. (2012).

Furthermore, Assumption 1 makes it explicit that regardless of the size of p, the relationships among the variables in the latent tree are restricted to be of rank m, and are thus *low rank* since $p > m$. To leverage this low rank structure, we propose using the following additive metric, a normalized variant of that in Anandkumar et al. (2011):

$$d^{\text{spectral}}(i,j) = -\log\Lambda_m(\boldsymbol{\Sigma}_{\boldsymbol{x}}(i,j)) + \tfrac{1}{2}\log\Lambda_m(\boldsymbol{\Sigma}_{\boldsymbol{x}}(i,i)) + \tfrac{1}{2}\log\Lambda_m(\boldsymbol{\Sigma}_{\boldsymbol{x}}(j,j)) \quad (5)$$

where $\Lambda_m(\boldsymbol{A})$ denotes the product of the top m singular values of $\boldsymbol{A}$ and $\boldsymbol{\Sigma}_{\boldsymbol{x}}(i,j) := \mathbb{E}[v_i v_j^\top|\boldsymbol{x}]$, i.e. the uncentered cross-covariance matrix.

We can then show that this metric is additive:

Lemma 1 *If Assumption 1 holds then,* d^{spectral} *is an additive tree metric (Definition 1).*

A proof is in the supplementary for completeness. From here, we use d to denote d^{spectral}, since that is the metric we use for our learning algorithm.

3.3 Recovering the Minimal Projective Latent Tree

It has been shown (Rzhetsky and Nei, 1993) that for any additive tree metric, $u(\boldsymbol{x})$ can be recovered by solving $\arg\min_{u \in \mathcal{U}} c(u)$ for $c(u)$:

$$c(u) = \sum_{(i,j) \in \mathcal{E}_u} d(i,j). \quad (6)$$

where $\mathcal{E}_u$ is the set of pairs of nodes which are adjacent to each other in u and $d(i,j)$ is computed using Eq. 3 and Eq. 4.

Note that the metric d we use in defining $c(u)$ is based on the expectations from the true distribution. In practice, the true distribution is unknown, and therefore we use an approximation for the distance metric $\hat{d}$. As we discussed in §3.1 all elements of the distance matrix are functions of observable quantities if the underlying tree u is known. However, only the word-word sub-block $\boldsymbol{D}_{WW}$ can be directly estimated from the data without knowledge of the tree structure.

This subtlety makes solving the minimization problem in Eq. 6 NP-hard (Desper and Gascuel, 2005) if u is allowed to be an arbitrary undirected tree. However, if we restrict u to be in $\mathcal{U}$, as we do in the above, then maximizing $\hat{c}(u)$ over $\mathcal{U}$ can be solved using the bilexical parsing algorithm from Eisner and Satta (1999). This is because the computation of the other sub-blocks of the distance matrix only depend on the partitions of the nodes shown in Figure 3 into A, B, G, and H, and not on the entire tree structure.

Therefore, the procedure to find a bracketing for a given POS tag $\boldsymbol{x}$ is to first estimate the distance matrix sub-block $\widehat{\boldsymbol{D}}_{WW}$ from raw text data (see §3.4), and then solve the optimization problem $\arg\min_{u \in \mathcal{U}} \hat{c}(u)$ using a variant of the Eisner-Satta algorithm where $\hat{c}(u)$ is identical to $c(u)$ in Eq. 6, with d replaced with $\hat{d}$.

Summary. We first defined a generative model that describes how a sentence, its sequence of POS tags, and its bracketing is generated (§2.3). First an undirected $u \in \mathcal{U}$ is generated (only as a function of the POS tags), and then u is mapped to a bracketing using a direction mapping h_{dir}. We then showed that we can define a distance metric between nodes in the undirected tree, such that minimizing it leads to a recovery of u. This distance metric can be computed based only on the text, without needing to identify the latent information (§3.2). If the true distance metric is known, with respect to the true distribution that generates the words in a sentence, then u can be fully recovered by optimizing the cost function $c(u)$. However, in practice the distance metric must be estimated from data, as discussed below.

Algorithm 1 The learning algorithm for finding the latent structure from a set of examples $(\boldsymbol{w}^{(i)}, \boldsymbol{x}^{(i)})$, $i \in [N]$.

Inputs: Set of examples $(\boldsymbol{w}^{(i)}, \boldsymbol{x}^{(i)})$ for $i \in [N]$, a kernel $K_\gamma(j,k,j',k'|\boldsymbol{x},\boldsymbol{x}')$, an integer m

Data structures: For each $i \in [N], j,k \in \ell(\boldsymbol{x}^{(i)})$ there is a (uncentered) covariance matrix $\widehat{\boldsymbol{\Sigma}}_{\boldsymbol{x}^{(i)}}(j,k) \in \mathbb{R}^{p \times p}$, and a distance $\hat{d}^{\text{spectral}}(j,k)$.

Algorithm:

(Covariance estimation) $\forall i \in [N], j,k \in \ell(\boldsymbol{x}^{(i)})$

- Let $C_{j',k'|i'} = w^{(i')}_{j'} (w^{(i')}_{k'})^\top$, $k_{j,k,j',k',i,i'} = K_\gamma(j,k,j',k'|\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(i')})$ and $\ell_{i'} = \ell(\boldsymbol{x}^{(i')})$, and estimate each $p \times p$ covariance matrix as:

$$\widehat{\boldsymbol{\Sigma}}_{\boldsymbol{x}}(j,k) = \frac{\sum_{i'=1}^{N} \sum_{j'=1}^{\ell_{i'}} \sum_{k'=1}^{\ell_{i'}} k_{j,k,j',k',i,i'} C_{j',k'|i'}}{\sum_{i'=1}^{N} \sum_{j'=1}^{\ell_{i'}} \sum_{k'=1}^{\ell_{i'}} k_{j,k,j',k',i,i'}}$$

- Compute $\hat{d}^{\text{spectral}}(j,k)$ $\forall j,k \in \ell(\boldsymbol{x}^{(i)})$ using Eq. 5.

(Uncover structure) $\forall i \in [N]$

- Find $\hat{u}^{(i)} = \arg\min_{u \in \mathcal{U}} \hat{c}(u)$, and for the ith example, return the structure $h_{\text{dir}}(\hat{u}^{(i)})$.

3.4 Estimation of d from Sparse Data

We now address the data sparsity problem, in particular that $\mathcal{D}(\boldsymbol{x})$ can be very small, and therefore estimating d for each POS sequence separately can be problematic.[3]

In order to estimate d from data, we need to estimate the covariance matrices $\boldsymbol{\Sigma}_{\boldsymbol{x}}(i,j)$ (for $i,j \in \{1,\ldots,\ell(\boldsymbol{x})\}$) from Eq. 5.

To give some motivation to our solution, consider estimating the covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{x}}(1,2)$ for the tag sequence $\boldsymbol{x} = (\texttt{DT}_1, \texttt{NN}_2, \texttt{VBD}_3, \texttt{DT}_4, \texttt{NN}_5)$. $\mathcal{D}(\boldsymbol{x})$ may be insufficient for an accurate empirical es-

[3] This data sparsity problem is quite severe – for example, the Penn treebank (Marcus et al., 1993) has a total number of 43,498 sentences, with 42,246 *unique* POS tag sequences, averaging $|\mathcal{D}(\boldsymbol{x})|$ to be 1.04.

timate. However, consider another sequence $\boldsymbol{x}' = (\texttt{RB}_1, \texttt{DT}_2, \texttt{NN}_3, \texttt{VBD}_4, \texttt{DT}_5, \texttt{ADJ}_6, \texttt{NN}_7)$. Although $\boldsymbol{x}$ and $\boldsymbol{x}'$ are not identical, it is likely that $\boldsymbol{\Sigma}_{\boldsymbol{x}'}(2,3)$ is similar to $\boldsymbol{\Sigma}_{\boldsymbol{x}}(1,2)$ because the determiner and the noun appear in similar syntactic context. $\boldsymbol{\Sigma}_{\boldsymbol{x}'}(5,7)$ also may be somewhat similar, but $\boldsymbol{\Sigma}_{\boldsymbol{x}'}(2,7)$ should not be very similar to $\boldsymbol{\Sigma}_{\boldsymbol{x}}(1,2)$ because the noun and the determiner appear in a different syntactic context.

The observation that the covariance matrices depend on local syntactic context is the main driving force behind our solution. The local syntactic context acts as an "anchor," which enhances or replaces a word index in a sentence with local syntactic context. More formally, an anchor is a function G that maps a word index j and a sequence of POS tags $\boldsymbol{x}$ to a local context $G(j, \boldsymbol{x})$. The anchor we use is $G(j, \boldsymbol{x}) = (j, x_j)$. Then, the covariance matrices $\boldsymbol{\Sigma}_{\boldsymbol{x}}$ are estimated using kernel smoothing (Hastie et al., 2009), where the smoother tests similarity between the different anchors $G(j, \boldsymbol{x})$.

The full learning algorithm is given in Figure 1. The first step in the algorithm is to estimate the covariance matrix block $\widehat{\boldsymbol{\Sigma}}_{\boldsymbol{x}^{(i)}}(j,k)$ for each training example $\boldsymbol{x}^{(i)}$ and each pair of preterminal positions (j,k) in $\boldsymbol{x}^{(i)}$. Instead of computing this block by computing the empirical covariance matrix for positions (j,k) in the data $\mathcal{D}(\boldsymbol{x})$, the algorithm uses all of the pairs (j',k') from all of N training examples. It averages the empirical covariance matrices from these contexts using a kernel weight, which gives a similarity measure for the position (j,k) in $\boldsymbol{x}^{(i)}$ and (j',k') in another example $\boldsymbol{x}^{(i')}$. γ is the kernel "bandwidth", a user-specified parameter that controls how inclusive the kernel will be with respect to examples in $\mathcal{D}$ (see § 4.1 for a concrete example). Note that the learning algorithm is such that it ensures that $\widehat{\boldsymbol{\Sigma}}_{\boldsymbol{x}^{(i)}}(j,k) = \widehat{\boldsymbol{\Sigma}}_{\boldsymbol{x}^{(i')}}(j',k')$ if $G(j, \boldsymbol{x}^{(i)}) = G(j', \boldsymbol{x}^{(i')})$ and $G(k, \boldsymbol{x}^{(i)}) = G(k', \boldsymbol{x}^{(i')})$.

Once the empirical estimates for the covariance matrices are obtained, a variant of the Eisner-Satta algorithm is used, as mentioned in §3.3.

3.5 Theoretical Guarantees

Our main theoretical guarantee is that Algorithm 1 will recover the correct tree $u \in \mathcal{U}$ with high probability, if the given top bracket is correct and if we obtain enough examples $(\boldsymbol{w}^{(i)}, \boldsymbol{x}^{(i)})$ from the model in §2. We give the theorem statement below. The constants lurking in the O-notation and the full proof are in the supplementary.

Denote $\sigma_{\boldsymbol{x}}(j,k)^{(r)}$ as the r^{th} singular value of $\boldsymbol{\Sigma}_{\boldsymbol{x}}(j,k)$. Let $\sigma^*(x) := \min_{j,k \in \ell(\boldsymbol{x})} \min\left(\sigma_{\boldsymbol{x}}(j,k)^{(m)}\right)$.

Theorem 1 *Define $\hat{u}$ as the estimated tree for tag sequence $\boldsymbol{x}$ and $u(\boldsymbol{x})$ as the correct tree. Let*

$$\triangle(\boldsymbol{x}) := \min_{u' \in \mathcal{U}: u' \neq u(\boldsymbol{x})} (c(u(\boldsymbol{x})) - c(u'))/(8|\ell(\boldsymbol{x})|)$$

Assume that

$$N \geq \mathcal{O}\left(\frac{m^2 \log\left(\frac{p^2 \ell(\boldsymbol{x})^2}{\delta}\right)}{\min(\sigma^*(\boldsymbol{x})^2 \triangle(\boldsymbol{x})^2, \sigma^*(\boldsymbol{x})^2)\nu_{\boldsymbol{x}}(\gamma)^2}\right)$$

Then with probability $1-\delta$, $\hat{u} = u(\boldsymbol{x})$.

where $\nu_{\boldsymbol{x}}(\gamma)$, defined in the supplementary, is a function of the underlying distribution over the tag sequences $\boldsymbol{x}$ and the kernel bandwidth γ.

Thus, the sample complexity of our approach depends on the dimensionality of the latent and observed states (m and p), the underlying singular values of the cross-covariance matrices ($\sigma^*(\boldsymbol{x})$) and the difference in the cost of the true tree compared to the cost of the incorrect trees ($\triangle(\boldsymbol{x})$).

4 Experiments

We report results on three different languages: English, German, and Chinese. For English we use the Penn treebank (Marcus et al., 1993), with sections 2–21 for training and section 23 for final testing. For German and Chinese we use the Negra treebank and the Chinese treebank respectively and the first 80% of the sentences are used for training and the last 20% for testing. All punctuation from the data is removed.[4]

We primarily compare our method to the constituent-context model (CCM) of Klein and Manning (2002). We also compare our method to the algorithm of Seginer (2007).

4.1 Experimental Settings

Top bracket heuristic Our algorithm requires the top bracket in order to direct the latent tree. In practice, we employ the following heuristic to find the bracket using the following three steps:

- If there exists a comma/semicolon/colon at index i that has at least a verb before i and both a noun followed by a verb after i, then return $([0, i-1], [i, \ell(x)])$ as the top bracket. (Pick the rightmost comma/semicolon/colon if multiple satisfy the criterion).

[4]We make brief use of punctuation for our top bracket heuristic detailed below before removing it.

Length	CCM	CCM-U	CCM-OB	CCM-UB
≤ 10	72.5	57.1	58.2	62.9
≤ 15	54.1	36	24	23.7
≤ 20	50	34.7	19.3	19.1
≤ 25	47.2	30.7	16.8	16.6
≤ 30	44.8	29.6	15.3	15.2
≤ 40	26.3	13.5	13.9	13.8

Table 1: Comparison of different CCM variants on English (training). U stands for universal POS tagset, OB stands for conjoining original POS tags with Brown clusters and UB stands for conjoining universal POS tags with Brown clusters. The best setting is just the vanilla setting, CCM.

- Otherwise find the first non-participle verb (say at index j) and return $([0, j-1], [j, \ell(\boldsymbol{x})])$.
- If no verb exists, return $([0, 1], [1, \ell(\boldsymbol{x})])$.

Word embeddings As mentioned earlier, each w_i can be an arbitrary feature vector. For all languages we use Brown clustering (Brown et al., 1992) to construct a $\log(C) + C$ feature vector where the first $\log(C)$ elements indicate which mergable cluster the word belongs to, and the last C elements indicate the cluster identity. For English, more sophisticated word embeddings are easily obtainable, and we experiment with neural word embeddings Turian et al. (2010) of length 50. We also explored two types of CCA embeddings: OSCCA and TSCCA, given in Dhillon et al. (2012). The OSCCA embeddings behaved better, so we only report its results.

Choice of kernel For our experiments, we use the kernel

$$K_\gamma(j,k,j',k'|\boldsymbol{x},\boldsymbol{x}') = \max\left\{0, 1 - \frac{\kappa(j,k,j',k'|\boldsymbol{x},\boldsymbol{x}')}{\gamma}\right\}$$

where γ denotes the user-specified bandwidth, and $\kappa(j,k,j',k'|\boldsymbol{x},\boldsymbol{x}') = \dfrac{|j-k|-|j'-k'|}{|j-k|+|j'-k'|}$ if $\boldsymbol{x}(j) = \boldsymbol{x}(j')$ and $\boldsymbol{x}(k') = \boldsymbol{x}(k)$, and $\text{sign}(j-k) = \text{sign}(j'-k')$ (and ∞ otherwise).

The kernel is non-zero if and only if the tags at position j and k in $\boldsymbol{x}$ are identical to the ones in position j' and k' in $\boldsymbol{x}'$, and if the direction between j and k is identical to the one between j' and k'. Note that the kernel is not binary, as opposed to the theoretical kernel in the supplementary material. Our experiments show that using a non-zero value different than 1 that is a function of the distance between j and k compared to the distance between j' and k' does better in practice.

Choice of data For CCM, we found that if the full dataset (all sentence lengths) is used in training, then performance degrades when evaluating on sentences of length ≤ 10. We therefore restrict the data used with CCM to sentences of length $\leq \ell$, where ℓ is the maximal sentence length being evaluated. This does not happen with our algorithm, which manages to leverage lexical information whenever more data is available. We therefore use the full data for our method for all lengths.

We also experimented with the original POS tags and the universal POS tags of Petrov et al. (2011). Here, we found out that our method does better with the universal part of speech tags. For CCM, we also experimented with the original parts of speech, universal tags (CCM-U), the cross-product of the original parts of speech with the Brown clusters (CCM-OB), and the cross-product of the universal tags with the Brown clusters (CCM-UB). The results in Table 1 indicate that the vanilla setting is the best for CCM.

Thus, for all results, we use universal tags for our method and the original POS tags for CCM. We believe that our approach substitutes the need for fine-grained POS tags with the lexical information. CCM, on the other hand, is fully unlexicalized.

Parameter Selection Our method requires two parameters, the latent dimension m and the bandwidth γ. CCM also has two parameters, the number of extra constituent/distituent counts used for smoothing. For both methods we chose the best parameters for sentences of length $\ell \leq 10$ on the English Penn Treebank (training) and used this set for all other experiments. This resulted in $m = 7, \gamma = 0.4$ for our method and 2, 8 for CCM's extra constituent/distituent counts respectively. We also tried letting CCM choose different hyperparameters for different sentence lengths based on dev-set likelihood, but this gave worse results than holding them fixed.

4.2 Results

Test I: Accuracy Table 2 summarizes our results. CCM is used with the initializer proposed in Klein and Manning (2002).[5] NN, CC, and BC indicate the performance of our method for neural embeddings, CCA embeddings, and Brown clustering respectively, using the heuristic for h_{dir} de-

[5]We used the implementation available at http://tinyurl.com/lhwk5n6.

	ℓ	English							German			Chinese		
		NN-O	NN	CC-O	CC	BC-O	BC	CCM	BC-O	BC	CCM	BC-O	BC	CCM
train	≤ 10	70.9	69.2	70.4	68.7	71.1	69.3	72.5	64.6	59.9	62.6	64.9	57.3	46.1
	≤ 20	55.1	53.5	53.2	51.6	53.0	51.5	50	52.7	48.7	47.9	51.4	46	22.4
	≤ 40	46.1	44.5	43.6	41.9	43.3	41.8	26.3	46.7	43.6	19.8	42.6	38.6	15
test	≤ 10	69.2	66.7	68.3	65.5	68.9	66.1	70.5	66.4	61.6	64.7	58.0	53.2	40.7
	≤ 15	60.3	58.3	58.6	56.4	58.6	56.5	53.8	57.5	53.5	49.6	54.3	49.4	35.9
	≤ 20	54.1	52.3	52.3	50.3	51.9	50.2	50.4	52.8	49.2	48.9	49.7	45.5	20.1
	≤ 25	50.8	49.0	48.6	46.6	48.3	46.6	47.4	50.0	46.8	45.6	46.7	42.7	17.8
	≤ 30	48.1	46.3	45.6	43.7	45.4	43.8	44.9	48.3	45.4	21.9	44.6	40.7	16.1
	≤ 40	45.5	43.8	43.0	41.1	42.7	41.1	26.1	46.9	44.1	20.1	42.2	38.6	14.3

Table 2: F_1 bracketing measure for the test sets and train sets in three languages. NN, CC, and BC indicate the performance of our method for neural embeddings, CCA embeddings, and Brown clustering respectively, using the heuristic for h_{dir} described in § 4.1. NN-O, CC-O, and BC-O indicate that the oracle (i.e. true top bracket) was used for h_{dir}.

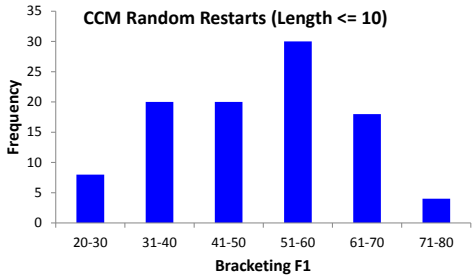

Figure 4: Histogram showing performance of CCM across 100 random restarts for sentences of length ≤ 10.

scribed in § 4.1. NN-O, CC-O, and BC-O indicate that the oracle (i.e. true top bracket) was used for h_{dir}. For our method, test set results can be obtained by using Algorithm 1 (except the distances are computed using the training data).

For English, while CCM behaves better for short sentences ($\ell \leq 10$), our algorithm is more robust with longer sentences. This is especially noticeable for length ≤ 40, where CCM breaks down and our algorithm is more stable. We find that the neural embeddings modestly outperform the CCA and Brown cluster embeddings.

The results for German are similar, except CCM breaks down earlier at sentences of $\ell \leq 30$. For Chinese, our method substantially outperforms CCM for all lengths. Note that CCM performs very poorly, obtaining only around 20% accuracy even for sentences of $\ell \leq 20$. We didn't have neural embeddings for German and Chinese (which worked best for English) and thus only used Brown cluster embeddings.

For English, the disparity between NN-O (oracle top bracket) and NN (heuristic top bracket) is rather low suggesting that our top bracket heuristic is rather effective. However, for German and Chinese note that the "BC-O" performs substantially better, suggesting that if we had a better top bracket heuristic our performance would increase.

Test II: Sensitivity to initialization The EM algorithm with the CCM requires very careful initialization, which is described in Klein and Manning (2002). If, on the other hand, random initialization is used, the variance of the performance of the CCM varies greatly. Figure 4 shows a histogram of the performance level for sentences of length ≤ 10 for different random initializers. As one can see, for some restarts, CCM obtains accuracies lower than 30% due to local optima. Our method does not suffer from local optima and thus does not require careful initialization.

Test III: Comparison to Seginer's algorithm Our approach is not directly comparable to Seginer's because he uses punctuation, while we use POS tags. Using Seginer's parser we were able to get results on the training sets. On English: 75.2% ($\ell \leq 10$), 64.2% ($\ell \leq 20$), 56.7% ($\ell \leq 40$). On German: 57.8% ($\ell \leq 10$), 45.0% ($\ell \leq 20$), and 39.9% ($\ell \leq 40$). On Chinese: 56.6% ($\ell \leq 10$), 45.1% ($\ell \leq 20$), and 38.9% ($\ell \leq 40$).

Thus, while Seginer's method performs better on English, our approach performs 2-3 points better on German, and both methods give similar performance on Chinese.

5 Conclusion

We described a spectral approach for unsupervised constituent parsing that comes with theoretical guarantees on latent structure recovery. Empirically, our algorithm performs favorably to the CCM of Klein and Manning (2002) without the need for careful initialization.

Acknowledgements: This work is supported by NSF IIS1218282, NSF IIS1111142, NIH R01GM093156, and the NSF Graduate Research Fellowship Program under Grant No. 0946825 (NSF Fellowship to APP).

References

A. Anandkumar, K. Chaudhuri, D. Hsu, S. M. Kakade, L. Song, and T. Zhang. 2011. Spectral methods for learning multivariate latent tree structure. *arXiv preprint arXiv:1107.1283*.

R. Bailly, F. Denis, and L. Ralaivola. 2009. Grammatical inference as a principal component analysis problem. In *Proceedings of ICML*.

R. Bailly, X. Carreras, F. M. Luque, and A. Quattoni. 2013. Unsupervised spectral learning of WCFG as low-rank matrix completion. In *Proceedings of EMNLP*.

P. F. Brown, P.V. Desouza, R.L. Mercer, V.J.D. Pietra, and J.C. Lai. 1992. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479.

O. P. Buneman. 1971. The recovery of trees from measures of dissimilarity. *Mathematics in the archaeological and historical sciences*.

P. Buneman. 1974. A note on the metric properties of trees. *Journal of Combinatorial Theory, Series B*, 17(1):48–50.

M.J. Choi, V. YF Tan, A. Anandkumar, and A.S. Willsky. 2011. Learning latent tree graphical models. *The Journal of Machine Learning Research*, 12:1771–1812.

C. K. Chow and C. N. Liu. 1968. Approximating Discrete Probability Distributions With Dependence Trees. *IEEE Transactions on Information Theory*, IT-14:462–467.

S. B. Cohen and N. A. Smith. 2009. Shared logistic normal distributions for soft parameter tying in unsupervised grammar induction. In *Proceedings of HLT-NAACL*.

S. B. Cohen and N. A. Smith. 2012. Empirical risk minimization for probabilistic grammars: Sample complexity and hardness of learning. *Computational Linguistics*, 38(3):479–526.

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

R. Desper and O. Gascuel. 2005. The minimum evolution distance-based approach to phylogenetic inference. *Mathematics of evolution and phylogeny*, pages 1–32.

P. S. Dhillon, J. Rodu, D. P. Foster, and L. H. Ungar. 2012. Two step cca: A new spectral method for estimating vector models of words. In *Proceedings of ICML*.

J. Eisner and G. Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of ACL*.

P. Erdõs, M. Steel, L. Székely, and T. Warnow. 1999. A few logs suffice to build (almost) all trees: Part ii. *Theoretical Computer Science*, 221(1):77–118.

J. Gillenwater, K. Ganchev, J. Graça, F. Pereira, and B. Taskar. 2010. Sparsity in dependency grammar induction. In *Proceedings of ACL*.

K. Gimpel and N.A. Smith. 2012. Concavity and initialization for unsupervised dependency parsing. In *Proceedings of NAACL*.

D. Golland, J. DeNero, and J. Uszkoreit. 2012. A feature-rich constituent context model for grammar induction. In *Proceedings of ACL*.

M. Gormley and J. Eisner. 2013. Nonconvex global optimization for latent-variable models. In *Proceedings of ACL*.

S. Harmeling and C. KI Williams. 2011. Greedy learning of binary latent trees. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(6):1087–1097.

T. Hastie, R. Tibshirani, and J. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer Verlag.

W. P. Headden, M. Johnson, and D. McClosky. 2009. Improving unsupervised dependency parsing with richer contexts and smoothing. In *Proceedings of NAACL-HLT*.

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

D. Hsu, S. M. Kakade, and P. Liang. 2012. Identifiability and unmixing of latent parse trees. *arXiv preprint arXiv:1206.3137*.

M. Ishteva, H. Park, and L. Song. 2012. Unfolding latent tree structures using 4th order tensors. *arXiv preprint arXiv:1210.1258*.

F. Jelinek, J. D. Lafferty, and R. L. Mercer. 1992. *Basic methods of probabilistic context free grammars*. Springer.

D. Klein and C. D. Manning. 2002. A generative constituent-context model for improved grammar induction. In *Proceedings of ACL*.

M. Kolar, A. P. Parikh, and E. P. Xing. 2010a. On sparse nonparametric conditional covariance selection. In *Proceedings of ICML*.

M. Kolar, L. Song, A. Ahmed, and E. P. Xing. 2010b. Estimating time-varying networks. *The Annals of Applied Statistics*, 4(1):94–123.

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.P. Parikh, L. Song, and E.P. Xing. 2011. A spectral algorithm for latent tree graphical models. In *Proceedings of ICML*.

S. Petrov, D. Das, and R. McDonald. 2011. A universal part-of-speech tagset. *ArXiv:1104.2086*.

A. Rzhetsky and M. Nei. 1993. Theoretical foundation of the minimum-evolution method of phylogenetic inference. *Molecular Biology and Evolution*, 10(5):1073–1095.

N. Saitou and M. Nei. 1987. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4):406–425.

Y. Seginer. 2007. Fast unsupervised incremental parsing. In *Proceedings of ACL*.

N. A. Smith and J. Eisner. 2005. Contrastive estimation: Training log-linear models on unlabeled data. In *Proceedings of ACL*.

L. Song, A.P. Parikh, and E.P. Xing. 2011. Kernel embeddings of latent tree graphical models. In *Proceedings of NIPS*.

V. I. Spitkovsky, H. Alshawi, and D. Jurafsky. 2010a. From baby steps to leapfrog: how less is more in unsupervised dependency parsing. In *Proceedings of NAACL*.

V. I. Spitkovsky, H. Alshawi, D. Jurafsky, and C. D. Manning. 2010b. Viterbi training improves unsupervised dependency parsing. In *Proceedings of CoNLL*.

V. I. Spitkovsky, H. Alshawi, and D. Jurafsky. 2013. Breaking out of local optima with count transforms and model recombination: A study in grammar induction. In *Proceedings of EMNLP*.

J. P. Turian, L.-A. Ratinov, and Y. Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of ACL*.

S. Zhou, J. Lafferty, and L. Wasserman. 2010. Time varying undirected graphs. *Machine Learning*, 80(2-3):295–319.