The Return of Lexical Dependencies: Neural Lexicalized PCFGs

Hao Zhu, Yonatan Bisk, Graham Neubig
Language Technologies Institute, Carnegie Mellon University
zhuhao@cmu.edu, {ybisk, gneubig}@cs.cmu.edu

Abstract

In this paper we demonstrate that *context free grammar (CFG) based methods for grammar induction benefit from modeling lexical dependencies*. This contrasts to the most popular current methods for grammar induction, which focus on discovering *either* constituents *or* dependencies. Previous approaches to marry these two disparate syntactic formalisms (e.g., lexicalized PCFGs) have been plagued by sparsity, making them unsuitable for unsupervised grammar induction. However, in this work, we present novel neural models of lexicalized PCFGs that allow us to overcome sparsity problems and effectively induce both constituents and dependencies within a single model. Experiments demonstrate that this unified framework results in stronger results on both representations than achieved when modeling either formalism alone.[1]

1 Introduction

Unsupervised grammar induction aims at building a formal device for discovering syntactic structure from natural language corpora. Within the scope of grammar induction, there are two main directions of research: unsupervised constituency parsing, which attempts to discover the underlying structure of phrases, and unsupervised dependency parsing, which attempts to discover the underlying relations between words. Early work on induction of syntactic structure focused on learning phrase structure and generally used some variant of probabilistic context-free grammars (PCFGs; Lari and Young, 1990; Charniak, 1996; Clark, 2001). In recent years, dependency grammars have gained favor as an alternative syntactic formulation (Yuret, 1998; Carroll and Charniak, 1992; Paskin, 2002). Specifically, the dependency model with valence (DMV) (Klein and Manning, 2004) forms the basis for many modern approaches in dependency induction. Most recent models for grammar induction, be they for PCFGs, DMVs, or other formulations, have generally coupled these models with some variety of neural model to use embeddings to capture word similarities, improve the flexibility of model parameterization, or both (He et al., 2018; Jin et al., 2019; Kim et al., 2019; Han et al., 2019).

Notably, the two different syntactic formalisms capture very different views of syntax. Phrase structure takes advantage of an abstracted recursive view of language, while the dependency structure more concretely focuses on the propensity of particular words in a sentence to relate to each other syntactically. However, few attempts at unsupervised grammar induction have been made to marry the two and let both benefit each other. This is precisely the issue we attempt to tackle in this paper.

As a specific formalism that allows us to model both formalisms at once, we turn to lexicalized probabilistic context-free grammars (L-PCFGs; Collins, 2003). L-PCFGs borrow the underlying machinery from PCFGs but expand the grammar by allowing rules to include information about the lexical heads of each phrase, an example of which is shown in Figure 1. The head annotation in the L-PCFG provides lexical dependencies that can be informative in estimating the probabilities of generation rules. For example, the probability of VP[CHASING] $\rightarrow$ VBZ[IS] VP[CHASING] is much higher than VP $\rightarrow$ VBZ VP, because ''chasing'' is a present participle. Historically, these grammars have been mostly used for *supervised* parsing, combined with traditional *count-based* estimators of rule probabilities (Collins, 2003). Within this context, lexicalized grammar rules are powerful,

[1] Code is available at https://github.com/neulab/neural-lpcfg.

Transactions of the Association for Computational Linguistics, vol. 8, pp. 647–661, 2020. https://doi.org/10.1162/tacl_a_00337
Action Editor: Slav Petrov. Submission batch: 3/2020; Revision batch: 6/2020; Published 10/2020.

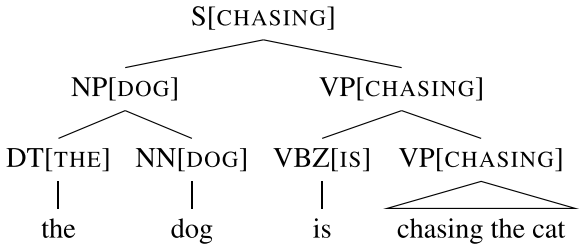


Figure 1: Lexicalized phrase structure tree for "the dog is chasing the cat." The head word of each constituent is indicated with parentheses.

but the counts available are sparse, and thus required extensive smoothing to achieve competitive results (Bikel, 2004; Hockenmaier and Steedman, 2002).

In this paper, we contend that with recent advances in neural modeling, it is time to **return to modeling lexical dependencies**, specifically in the context of unsupervised constituent-based grammar induction. We propose neural L-PCFGs as a parameter-sharing method to alleviate the sparsity problem of lexicalized PCFGs. Figure 2 illustrates the generation procedure of a neural L-PCFG. Different from traditional lexicalized PCFGs, the probabilities of production rules are not independently parameterized, but rather conditioned on the representations of non-terminals, preterminals, and lexical items (§3). Apart from devising lexicalized production rules (§2.3) and their corresponding scoring function, we also follow Kim et al.'s (2019) compound PCFG model for (non-lexicalized) constituency parsing with compound variables (§3.2), enabling modeling of a continuous mixture of grammar rules.[2] We define how to efficiently train (§4.1) and perform inference (§4.2) in this model using dynamic programming and variational inference.

Put together, we expect this to result in a model that both is effective, and *simultaneously* induces both phrase structure and lexical dependencies,[3] whereas previous work has focused on only one. Our empirical evaluation examines this hypothesis, asking the following question:

[2]In other words, we do not induce a single PCFG, but a distribution over a family of PCFGs.

[3]Note that by ''lexical dependencies'' we are referring to unilexical dependencies between the head word and child non-terminals, as opposed to bilexical dependencies between two words (as are modeled in many dependency parsing models).

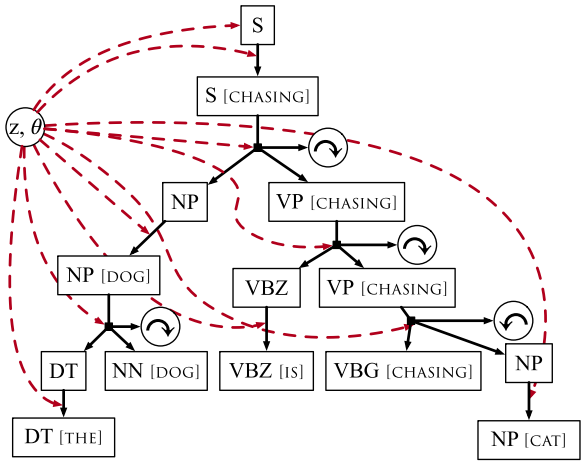


Figure 2: Model diagram of lexicalized compound PCFG. Black lines indicate production rules, and red dashed lines indicate that the compound variable and parameters participating in productions.

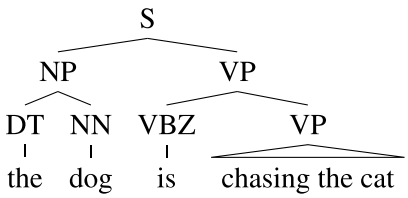
> *In neural grammar induction models, is it possible to jointly and effectively learn both phrase structure and lexical dependencies? Is using both in concert better at the respective tasks than specialized methods that model only one at a time?*

Our experiments (§5.3) answer in the affirmative, with better performance than baselines designed specially for either dependency or constituency parsing under multiple settings. Importantly, our detailed ablations show that methods of factorization play an important role in the performance of neural L-PCFGs (§5.3.2). Finally, qualitatively (§5.4), we find that latent labels induced by our model align with annotated gold non-terminals in PTB.

2 Motivation and Definitions

In this section, we will first provide the background of constituency grammars and dependency grammars, and then formally define the general L-PCFG, illustrating how both dependencies and phrase structures can be induced from L-PCFGs.

2.1 Phrase Structures and CFGs



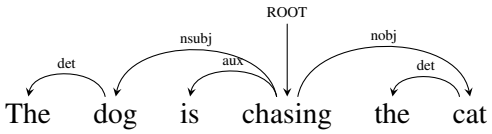
The phrase structure of a sentence is formed by recursively splitting constituents. In the parse above: The sentence is split into a noun phrase (NP) and a verb phrase (VP), which can themselves be further split into smaller constituents; for example, the NP comprises a determiner (DT) ‘‘the’’ and a normal noun (NN) ‘‘dog.’’

Such phrase structures are represented as a context-free grammar[4] (CFG), which can generate an infinite set of sentences via the repeated application of a finite set $\mathcal{R}$ of rules:

$$\begin{aligned} S &\to A, && A \in \mathcal{N} \\ A &\to BC, && A \in \mathcal{N}, B, C \in \mathcal{N} \cup \mathcal{P} \\ T &\to \alpha, && T \in \mathcal{P} \end{aligned}$$

S denotes a start symbol, $\mathcal{N}$ is a finite set of non-terminals, $\mathcal{P}$ is a finite set of preterminals, and Σ is a set of terminal symbols (i.e., words and punctuation).

2.2 Dependency Structures and Grammars



In a dependency tree of a sentence, the syntactic nodes are the words in the sentence. Here the root is the **root word** of the sentence, and the children of each word are its **dependents**. Above, the root word is *chasing*, which has three dependents, its subject (nsubj) *dog*, auxiliary verb (aux) *is*, and object (nobj) *cat*. A dependency grammar[5] specifies the possible **head-dependent** pairs $\mathcal{D} = \{(\alpha_i, \beta_i)\} \in (\mathcal{V} \cup \{\text{ROOT}\}) \times \mathcal{V}$, where the set $\mathcal{V}$ denotes the vocabulary.

2.3 Lexicalized CFGs

Although both the constituency and dependency grammars capture some aspects of syntax, we aim to leverage their relative strengths in a single unified formalism. In a unified grammar, these two types of structure can benefit each other. For example, in *The dog is chasing the cat of my neighbor's*, while the phrase *of my neighbor's* might be incorrectly marked as the adverbial phrase of *chasing* in a dependency model, the constituency parser can provide the constraint that *the cat of my neighbor's* is a constituent, thereby requiring *chasing* to be the head of the phrase.

Lexicalized CFGs are based on a backbone similar to standard CFGs but parameterized to be sensitive to lexical dependencies such as those used in dependency grammars. Similarly to CFGs, L-CFGs are defined as a five-tuple $\mathcal{T} = (S, \mathcal{N}, \mathcal{P}, \Sigma, \mathcal{R})$. The differences lie in the formulation of rules $\mathcal{R}$:

$$\begin{aligned} &(1) && S \to A[\alpha], && A \in \mathcal{N} \\ &(2l) && A[\alpha] \to B[\alpha]C[\beta], && A \in \mathcal{N}, B, C \in \mathcal{N} \cup \mathcal{P} \\ &(2r) && A[\alpha] \to B[\beta]C[\alpha], && A \in \mathcal{N}, B, C \in \mathcal{N} \cup \mathcal{P} \\ &(3) && T[\alpha] \to \alpha, && T \in \mathcal{P} \end{aligned}$$

where $\alpha, \beta \in \Sigma$ are words, and mark the head of constituent when they appear in ‘‘$[\cdot]$’’.[6] Branching rules (2l) and (2r) encode the dependencies (α, β).[7]

In a lexicalized CFG, a sentence $\boldsymbol{x}$ can be generated by iterative binary splitting and emission, forming a **parse tree** $t = [r_1, r_2, \ldots, r_{2|\boldsymbol{x}|}]$, where rules r_i are sorted from top to bottom and from left to right. We will denote the set of parse trees that generate $\boldsymbol{x}$ within grammar $\mathcal{T}$ as $\mathcal{T}_{\boldsymbol{x}}$.

2.4 Grammar Induction with L-PCFGs

In this subsection, we will introduce L-PCFGs, the probabilistic formulation for L-CFGs. The task of *grammar induction* is to ask, given a corpus $\mathcal{C} \subset \Sigma^+$, how can we obtain the probabilistic generative grammar that maximizes its likelihood. With the induced grammar, we are also interested in how to obtain the trees that are most likely given an individual sentence—in other words, syntactic parsing according to this grammar.

We begin by defining the probability distribution over sentences $\boldsymbol{x}$, by marginalizing over all parse trees that may have generated $\boldsymbol{x}$:

$$p_{\boldsymbol{z}}(\boldsymbol{x}) = \sum_{t \in \mathcal{T}_{\boldsymbol{x}}} p_{\boldsymbol{z}}(t) = \frac{1}{\mathcal{Z}(\mathcal{T}, \boldsymbol{z})} \sum_{t \in \mathcal{T}_{\boldsymbol{x}}} \tilde{p}_{\boldsymbol{z}}(t) \quad (1)$$

where $\tilde{p}_{\boldsymbol{z}}(t)$ is an unnormalized probability of a parse tree (which we will refer to as an

[4]Note $\epsilon \notin \mathcal{T}$ and $\Sigma \cap \mathcal{T} = \varnothing$, so this formulation does not capture the structure of sentences of length zero or one.

[5]This work assumes a projective tree.

[6]Without loss of generality, we only consider binary branching in $\mathcal{T}$.

[7]Note that root seeking rule (1) encodes (ROOT, α).

energy function), $\mathcal{Z}(\mathcal{T}, \boldsymbol{z}) = \sum_{t \in \mathcal{T}} \tilde{p}_{\boldsymbol{z}}(t)$ is the normalizing constant, and $\boldsymbol{z}$ is a compound variable (§3.2) that allows for more complex and expressive generative grammars (Robbins et al., 1951).

We define the energy function of a parse tree by exponentiating a score $G_\theta(t, \boldsymbol{z})$

$$\tilde{p}_{\boldsymbol{z}}(t) \propto \exp G_\theta(t, \boldsymbol{z}) \tag{2}$$

where θ is the parameter of function G_θ. Theoretically, $G_\theta(t)$ could be an arbitrary scoring function, but in this paper, as with most previous work, we consider a context-free scoring function, where the score of each rule r_i is independent of the other rules in the parse tree t:

$$G_\theta(t, \boldsymbol{z}) = \sum_{i=1}^{k} g_\theta(r_i, \boldsymbol{z}), \tag{3}$$

where $g_\theta(r, \boldsymbol{z}) : \mathcal{R} \times \mathbb{R}^n \to \mathbb{R}$ is the rule-scoring function which maps the rule and latent variable $z \in \mathbb{R}^n$ to real space, assigning a log likelihood to each rule. This formulation allows for efficient calculation using dynamic programming. We also include a restriction that the energies must be *top-down locally-normalized*, under which the partition function should automatically equate to 1

$$\mathcal{Z}(\mathcal{T}, \boldsymbol{z}) = \sum_{t \in \mathcal{T}} \exp G_\theta(t, \boldsymbol{z}) = 1 \tag{4}$$

To train an L-PCFG, we maximize the log likelihood of the corpus (the latent variable is marginalized out):

$$\theta^* = \arg\max_\theta \sum_{\boldsymbol{x} \in \mathcal{C}} \log \mathbb{E}_{\boldsymbol{z}} p_{\boldsymbol{z}}(\boldsymbol{x}) \tag{5}$$

And obtain the most likely parse tree of a sentence by maximizing the posterior probability:

$$t^* = \arg\max_{t \in \mathcal{T}_{\boldsymbol{x}}} \mathbb{E}_{\boldsymbol{z}} \exp G_{\theta^*}(t, \boldsymbol{z}) \tag{6}$$

3 Neural Lexicalized PCFGs

As noted, one advantage of L-PCFGs is that the obtained t^* encodes both dependencies and phrase structures, allowing both to be induced simultaneously. We also expect this to improve performance, because different information is capture by each of these two structures. However, this expressivity comes at a price: more complex rules. In contrast to the traditional PCFG, which has $\mathcal{O}(|\mathcal{N}|(|\mathcal{N}| + |\mathcal{P}|)^2)$ production rules, the L-PCFG requires $\mathcal{O}(|V||\mathcal{N}|(|\mathcal{N}|+|\mathcal{P}|)^2)$ production rules. Because traditionally rules of L-PCFGs have been parameterized independently by scalars, namely, $g_\theta(r_i, \boldsymbol{z}) = \theta_i$ (Collins, 2003), these parameters were hard to estimate because of data sparsity.

We propose an alternate parameterization, the *neural L-PCFG*, which ameliorates these sparsity problems through parameter sharing, and the *compound L-PCFG*, which allows a more flexible sentence-by-sentence parameterization of the model. Below, we explain the neural L-PCFG factorization that we found performed best but include ablations of our decisions in Section 5.3.2.

3.1 Neural L-PCFG Factorization

The score of an individual rule is calculated as the combination of several component probabilities:

root to non-terminal probability $p_{\boldsymbol{z}}(S \to A)$**:**
Probability that the start symbol produces a non-terminal A.[8]

word emission probability $p_{\boldsymbol{z}}(A \to \alpha)$**:**
Probability that the head word of a constituent is α conditioned on that the non-terminal of the constituent is A.

head non-terminal probability
$p_{\boldsymbol{z}}(B, \curvearrowleft \mid A, \alpha)$ **or** $p_{\boldsymbol{z}}(C, \curvearrowright \mid A, \alpha)$**:**
Probability of the headedness direction and head-inheriting child[9] conditioned on the parent non-terminal and head words.

non-inheriting child probability
$p_{\boldsymbol{z}}(C \mid A, B, \alpha, \curvearrowleft)$ **or** $p_{\boldsymbol{z}}(B \mid A, C, \alpha, \curvearrowright)$**:**
Probability of the non-inheriting child conditioned on the headedness direction, and parent and head-inheriting child non-terminals.

The score of root-seeking rule (1) is factorized as the product of the root to non-terminal score and word emission scores, as shown in Equation (7).

$$g_\theta(S \to A[\alpha], \boldsymbol{z}) = \log p_{\boldsymbol{z}}(S \to A) + \log p_{\boldsymbol{z}}(A \to \alpha) \tag{7}$$

The scores of branching rules (2l) and (2r) are factorized as the sum of a binary non-terminal score, a head non-terminal score, and

[8] that is, A is the non-terminal of the whole sentence

[9] Child non-terminals that inherit the parent's head word.

a word emission score. Equation (8) describes the factorization of the score of rule (2l) and (2r):

$$\begin{aligned}
&g_\theta(A[\alpha] \to B[\alpha]C[\beta], \boldsymbol{z}) \\
&\quad = \log p_{\boldsymbol{z}}(B,\curvearrowleft| A,\alpha) + \log p_{\boldsymbol{z}}(C \,|\, A,B,\alpha,\curvearrowleft) \\
&\quad + \log p_{\boldsymbol{z}}(C \to \beta) \\
&g_\theta(A[\alpha] \to B[\beta]C[\alpha], \boldsymbol{z}) \\
&\quad = \log p_{\boldsymbol{z}}(C,\curvearrowright| A,\alpha) + \log p_{\boldsymbol{z}}(B \,|\, A,C,\alpha,\curvearrowright) \\
&\quad + \log p_{\boldsymbol{z}}(B \to \beta)
\end{aligned} \tag{8}$$

Because the head of preterminals is already specified upon generation of one of the ancestor non-terminals, the score of emission rule (3) is 0.

The component probabilities are all similarly parameterized, vectors corresponding to component non-terminals or terminals are fed through a multi-layer perceptron denoted $f(\cdot)$, and a dot product is taken with another vector corresponding to a component non-terminal or terminal. Specifically, the root to non-terminal probability is

$$\begin{aligned}
p_{\boldsymbol{z}}(S \to A) &= \exp \pi_{\boldsymbol{z}}(S \to A)/Z(\boldsymbol{z}) \\
\pi_{\boldsymbol{z}}(S \to A) &= f^{(1)}([\boldsymbol{u}_S; \boldsymbol{z}])^T \boldsymbol{v}_A
\end{aligned} \tag{9}$$

where ; denotes concatenation and the word emission probability is

$$\begin{aligned}
p_{\boldsymbol{z}}(A \to \alpha) &= \exp \pi_{\boldsymbol{z}}(A \to \alpha)/Z(A, \boldsymbol{z}) \\
\pi_{\boldsymbol{z}}(A \to \alpha) &= f^{(2)}([\boldsymbol{u}_A; \boldsymbol{z}])^T \boldsymbol{v}_\alpha
\end{aligned} \tag{10}$$

with partition functions $Z(\boldsymbol{z})$ such that $\sum_{A \in \mathcal{N}} p_{\boldsymbol{z}}(S \to A) = 1$ and $Z(A, \boldsymbol{z})$ s.t. $\sum_{\alpha \in \Sigma} p_{\boldsymbol{z}}(A \to \alpha) = 1$.

The non-inheriting child probabilities for left- and right-headed dependencies are

$$\begin{aligned}
p_{\boldsymbol{z}}(C \mid A, B, \alpha, \curvearrowleft) &= \frac{\exp \pi_{\boldsymbol{z}}(A[\alpha] \overset{\curvearrowleft}{\to} B[\alpha]C)}{Z(A, B, \alpha, \curvearrowleft, \boldsymbol{z})} \\
p_{\boldsymbol{z}}(B \mid A, C, \alpha, \curvearrowright) &= \frac{\exp \pi_{\boldsymbol{z}}(A[\alpha] \overset{\curvearrowright}{\to} BC[\alpha])}{Z(A, C, \alpha, \curvearrowleft, \boldsymbol{z})} \\
\pi_{\boldsymbol{z}}(A[\alpha] \overset{\curvearrowleft}{\to} B[\alpha]C) &= [\boldsymbol{w}_A^{\curvearrowleft}; \boldsymbol{w}_\alpha^{\curvearrowleft}; \boldsymbol{z}]^T \boldsymbol{v}_{BC} \\
\pi_{\boldsymbol{z}}(A[\alpha] \overset{\curvearrowright}{\to} BC[\alpha]) &= [\boldsymbol{w}_A^{\curvearrowright}; \boldsymbol{w}_\alpha^{\curvearrowright}; \boldsymbol{z}]^T \boldsymbol{v}_{BC}
\end{aligned} \tag{11}$$

where partition functions satisfy $\sum_C p_{\boldsymbol{z}}(C \mid A, B, \alpha, \curvearrowleft)$ and $\sum_B p_{\boldsymbol{z}}(B \mid A, C, \alpha, \curvearrowright) = 1$.

The respective head non-terminal scores are

$$\begin{aligned}
p_{\boldsymbol{z}}(B, \curvearrowleft| A, \alpha) &= \frac{\exp \pi_{\boldsymbol{z}}(A[\alpha] \overset{\curvearrowleft}{\to} B[\alpha])}{Z(A, \alpha, \boldsymbol{z})} \\
p_{\boldsymbol{z}}(C, \curvearrowright| A, \alpha) &= \frac{\exp \pi_{\boldsymbol{z}}(A[\alpha] \overset{\curvearrowright}{\to} B[\alpha])}{Z(A, \alpha, \boldsymbol{z})} \\
\pi_{\boldsymbol{z}}(A[\alpha] \overset{\curvearrowleft}{\to} B[\alpha]) &= f^{(3)}([\boldsymbol{u}_A; \boldsymbol{u}_\alpha; \boldsymbol{z}])^T \boldsymbol{v}_{B\curvearrowleft} \\
\pi_{\boldsymbol{z}}(A[\alpha] \overset{\curvearrowright}{\to} B[\alpha]) &= f^{(3)}([\boldsymbol{u}_A; \boldsymbol{u}_\alpha; \boldsymbol{z}])^T \boldsymbol{v}_{B\curvearrowright}
\end{aligned} \tag{12}$$

where the partition function satisfies $\sum_B p_{\boldsymbol{z}}(B, \curvearrowleft| A, \alpha) + \sum_C p_{\boldsymbol{z}}(C, \curvearrowright| A, \alpha) = 1$.

Here vectors $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^d$ represent the embeddings of non-terminals, preterminals and words. $f^{(i)}, i = 1, 2, 3$ are multilayer perceptrons with different set of parameters, where we use residual connections[10] (He et al., 2016) between layers to facilitate training of deeper models.

3.2 Compound Grammar

Among various existing grammar induction models, the compound PCFG model of Kim et al. (2019) both shows highly competitive results and follows a PCFG-based formalism similar to ours, and thus we build upon this method. The *compound* in compound PCFG refers to the fact that it uses a compound probability distribution (Robbins et al., 1951) in modeling and estimation of its parameters. A compound probability distribution enables continuous variants of grammars, allowing the probabilities of the grammar to change based on the unique characteristics of the sentence. In general, compound variables can be devised in any way that may inform the specification of the rule probabilities (e.g., a structured variable to provide frame semantics or the social context in which the sentence is situated). In this way, compound grammar increases the capacity of the original PCFG.

In this paper, we use a latent compound variable $\boldsymbol{z}$ that is sampled from a standard spherical Gaussian distribution.

$$\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{13}$$

We denote the probability of latent variable $\boldsymbol{z}$ as $p_{\mathcal{N}(\mathbf{0},\mathbf{I})}(\boldsymbol{z})$. By marginalizing out the compound variable, we obtain the log likelihood of a sentence:

$$\log p(\boldsymbol{x}) = \log \int_{\boldsymbol{z}} p_{\boldsymbol{z}}(\boldsymbol{x}) p_{\mathcal{N}(\mathbf{0},\mathbf{I})}(\boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} \tag{14}$$

Algorithm 1 Generative Procedure of Neural L-PCFGs: Sentences Are Generated from Start Symbol S and Compound Variable $\boldsymbol{z}$ Recursively.

Require: $\mathcal{N}, \mathcal{T}, P_1, P_2$
  function RECURSIVE($N, \alpha, \boldsymbol{z}$)
    $N_1, N_2, d, \beta \sim P_2(N_1, N_2, d, \beta \mid N, \alpha, \boldsymbol{z})$
    if $d = \curvearrowright$ **then**
      $\alpha, \beta \leftarrow \beta, \alpha$
    end if
    if $N_1 \in \mathcal{N}$ **then**
      $\mathcal{S}_l \leftarrow$ RECURSIVE($N_1, \alpha, \boldsymbol{z}$)
    else
      $\mathcal{S}_l \leftarrow [\alpha]$
    end if
    if $N_2 \in \mathcal{N}$ **then**
      $\mathcal{S}_r \leftarrow$ RECURSIVE($N_2, \beta, \boldsymbol{z}$)
    else
      $\mathcal{S}_r \leftarrow [\beta]$
    end if
    return CONCATENATE(S_l, S_r)
  end function
  $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $N, \alpha \sim P_1(N, \alpha \mid S, \boldsymbol{z})$
  return RECURSIVE($N, \alpha, \boldsymbol{z}$)

Algorithm 1 shows the procedure to generate a sentence recursively from a random compound variable and a distribution over the production rules in a pre-order traversal manner, where P_1 and P_2 are defined using g_θ from Equations (7) and (8), respectively:

$$\begin{aligned} P_1(N, \alpha \mid S, \boldsymbol{z}) &= \exp(g_\theta(S \rightarrow A[\alpha], \boldsymbol{z})) \\ P_2(N_1, N_2, \curvearrowleft, \beta \mid N, \alpha, \boldsymbol{z}) &\\ &= \exp(g_\theta(A[\alpha] \rightarrow B[\alpha]C[\beta], \boldsymbol{z})) \\ P_2(N_1, N_2, \curvearrowright, \beta \mid N, \alpha, \boldsymbol{z}) &\\ &= \exp(g_\theta(A[\alpha] \rightarrow B[\beta]C[\alpha], \boldsymbol{z})) \end{aligned} \tag{15}$$

4 Training and Inference

4.1 Training

It is intractable to obtain either the exact log likelihood by integration over z, and estimation by Monte Carlo sampling would be hopelessly inefficient. However, we can optimize the evidence lower bound (ELBo):

$$\begin{aligned} \mathcal{L}(\boldsymbol{x}) &= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \log p_{\boldsymbol{z}}(\boldsymbol{x}) \\ &\quad - \mathrm{KL}[q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \| p_{\mathcal{N}(\mathbf{0},\mathbf{I})}(\boldsymbol{z})] \\ &\leq \mathbb{E}_{p_{\mathcal{N}(\mathbf{0},\mathbf{I})}(\boldsymbol{z})} p_{\boldsymbol{z}}(\boldsymbol{x}) \end{aligned} \tag{16}$$

[10] $f(x) = \sigma(W_2(\sigma(W_1 x + b_1)) + b) + x$.

where $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ is the proposal probability parameterized by an inference network, similar to those used in variantial autoencoders (Kingma and Welling, 2014). The ELBo can be estimated by Monte Carlo sampling:

$$\begin{aligned} \mathcal{L}(x) =& \frac{1}{L} \sum_{i=1}^{L} \log p_{\boldsymbol{z}_i}(\boldsymbol{x}) \\ & - \mathrm{KL}[q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \| p_{\mathcal{N}(\mathbf{0},\mathbf{I})}(\boldsymbol{z})] \end{aligned} \tag{17}$$

where $\{\boldsymbol{z}_i\}_{i=1}^{L}$ are sampled from $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$. We model the proposal probability as an orthogonal Gaussian distribution:

$$q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) = p_{\mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}))}(\boldsymbol{z}) \tag{18}$$

where $(\boldsymbol{\mu}, \boldsymbol{\sigma})$ are output by the inference network

$$\boldsymbol{\mu} = f_\mu(\boldsymbol{x}), \boldsymbol{\sigma} = f_\sigma(\boldsymbol{x}) \tag{19}$$

Both f_μ and f_σ are parameterized as LSTMs (Hochreiter and Schmidhuber, 1997). Note that the inference network could be optimized by the reparameterization trick (Kingma and Welling, 2014):

$$\hat{\boldsymbol{z}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \hat{\boldsymbol{z}} \tag{20}$$

where $\odot$ denotes Hadamard operation. The KL divergence between $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ and $\mathcal{N}(\mathbf{0}, \mathbf{I})$ is

$$\begin{aligned} &\mathrm{KL}[q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \| p_{\mathcal{N}(\mathbf{0},\mathbf{I})}(\boldsymbol{z})] \\ &\quad = -\frac{1}{2}(\sum_{i=1}^{n}(\log \sigma_i - \sigma_i + 1) - \|\boldsymbol{\mu}\|_2^2) \end{aligned} \tag{21}$$

Initialization We initialize word embeddings using GloVe embeddings (Pennington et al., 2014). We further cluster word embeddings with k-means (MacQueen, 1967), as shown in Figure 3, and use the centroids of the clusters to initialize the embeddings of preterminals. The k-means algorithm is initialized using the k-means++ method and trained until convergence. The intuition therein is that this gives the model a rough idea of syntactic categories before starting grammar induction. We also consider the variant without pretrained word embeddings, where we initialize word embeddings and preterminals both by drawing from $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Other parameters are initialized by Xavier normal initialization (Glorot and Bengio, 2010).

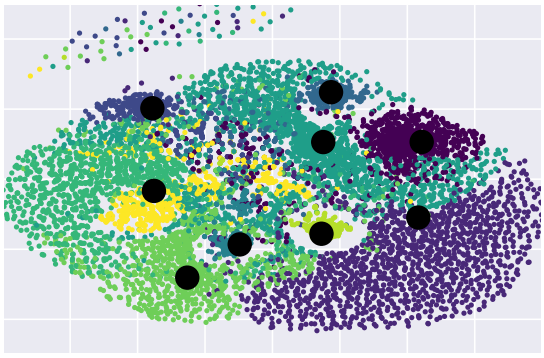


Figure 3: The clustering Result of GloVe Embeddings. Different colors represent cluster class of each word, and larger black points represent the initial embeddings of preterminals, i.e., cluster centroids. The two-dimension visualization is obtained by TSNE (Maaten and Hinton, 2008).

Curriculum Learning We also apply curriculum learning (Bengio et al., 2009; Spitkovsky et al., 2010) to learn the grammar gradually. Starting at half of the maximum length in the training set, we raise the length limit by $\alpha\%$ each epoch.

4.2 Inference

We are interested in the induced parse tree for each sentence in the task of unsupervised parsing, that is, the most probable tree $\hat{t}$

$$\begin{aligned}\hat{t} &= \arg\max_{t} p(t \mid \boldsymbol{x}) \\ &= \arg\max_{t \in \mathcal{T}_x} \int_{\boldsymbol{z}} p_{\boldsymbol{z}}(t) p(\boldsymbol{z} \mid \boldsymbol{x}) \, \mathrm{d}\boldsymbol{z}\end{aligned} \tag{22}$$

where $p(\boldsymbol{z} \mid \boldsymbol{x})$ is the posterior over compound variables. However, it is intractable to get the most probable tree. Hence we use the mean $\boldsymbol{\mu} = f_\mu(\boldsymbol{x})$ predicted by the inference network and replace $p(\boldsymbol{z} \mid \boldsymbol{x})$ with a Dirac delta distribution $\delta(\boldsymbol{z} - \boldsymbol{\mu})$ in place of the real distribution to approximate the integral[11]

$$\begin{aligned}\hat{t} &\approx \arg\max_{t \in \mathcal{T}_x} \int_{\boldsymbol{z}} p_{\boldsymbol{z}}(t) \delta(\boldsymbol{z} - \boldsymbol{\mu}) \, \mathrm{d}\boldsymbol{z} \\ &= \arg\max_{t \in \mathcal{T}_x} p_{\boldsymbol{\mu}}(t)\end{aligned} \tag{23}$$

The most probable tree can be obtained via the CYK algorithm.

[11]Note that it is also possible to use other methods for approximation. For example, we can use $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ in place of posterior distribution. However, using it still results in high prediction variance of the max function. We did not observe a significant improvement with other methods.

Hyperparameter	Value
$\lvert\mathcal{N}\rvert$, $\lvert\mathcal{P}\rvert$	10, 20
n	60
d	300
α	10
#layers of $f^{(1)}$, $f^{(2)}$, $f^{(3)}$	6, 6, 4
non-linear activation	`relu`

Table 1: Hyper-parameters and values.

5 Experiments

5.1 Data Setup

All models are evaluated using the Penn Treebank (Marcus et al., 1993) as the test corpus, following the splits and preprocessing methods, including removing punctuation, provided by Kim et al. (2019). To convert the original phrase bracket and label annotations to dependency annotations, we use Stanford typed dependency representations (De Marneffe and Manning, 2008).

We use three standard metrics to measure the performance of models on the validation and test sets: directed and undirected attachment score (DAS and UAS) for dependency parsing, and unlabeled constituent F1 score for constituency parsing.

We tune hyperparameters of the model to minimize perplexity on the validation set. We choose perplexity because it requires only plain text and not annotated parse trees. Specifically, we tuned the architecture of $f^{(i)}, i = 1, 2, 3$ in the space of multilayer perceptrons, with the dimension of each layer being $n + d$, with residual connections and different non-linear activation functions. Table 1 shows the final hyper-parameters of our model. Due to memory constraints on a single graphic card, we set the number of non-terminals and preterminals to 10 and 20, respectively. Later we will show that the compound PCFG's performance is benefited by a larger grammar; it is therefore possible the same is true for our neural L-PCFG. Section 7 includes a more detailed discussion of space complexity.

5.2 Baselines

We compare our neural L-PCFGs with the following baselines:

Compound PCFG The compound PCFG (Kim et al., 2019) is an unsupervised constituency parsing model that is a PCFG model with neural scoring. The main difference between this model

	Gold Tags	Word Embedding	DAS		UAS		F1	
			Dev	Test	Dev	Test	Dev	Test
Compound PCFG**	✗	$\mathcal{N}(\mathbf{0}, \mathbf{I})$	21.2	23.5	38.9	40.8	-	**55.2**
Compound PCFG	✗	$\mathcal{N}(\mathbf{0}, \mathbf{I})$	15.6 (3.9)	17.8 (4.2)	27.7 (4.1)	30.2 (5.3)	45.63 (1.71)	47.79 (2.32)
Compound PCFG	✗	GloVe	16.4 (2.4)	18.6 (3.7)	28.7 (3.5)	31.6 (4.5)	45.52 (2.14)	48.20 (2.53)
DMV	✗	-	24.7 (1.5)	27.2 (1.9)	43.2 (1.9)	44.3 (2.2)	-	-
DMV	✓	-	28.5 (1.9)	29.9 (2.5)	45.5 (2.8)	47.3 (2.7)	-	-
Neural L-PCFGs	✗	$\mathcal{N}(\mathbf{0}, \mathbf{I})$	37.5 (2.7)	39.7 (3.1)	50.6 (3.1)	53.3 (4.2)	**52.90 (3.72)**	**55.31 (4.03)**
Neural L-PCFGs	✗	GloVe	**38.2 (2.1)**	**40.5 (2.9)**	**54.4 (3.6)**	**55.9 (3.8)**	45.67 (0.95)	47.23 (2.06)
Neural L-PCFGs	✓	$\mathcal{N}(\mathbf{0}, \mathbf{I})$	35.4 (0.5)	39.2 (1.1)	50.0 (1.3)	53.8 (1.7)	51.16 (5.11)	54.49 (6.32)

Table 2: Dependency and constituency parsing results. *DAS*/*UAS* stand for directed/undirected accuracy. For the compound PCFG we use heuristic head rules to obtain dependencies (§5.2). Figures in the parenthesis show the standard deviation calculated from five runs with different random seeds. **indicates a large (30 NT, 60 PT) compound PCFG from Kim et al. (2019) – we could not use this size in our experiments due to memory constraints. Results are not directly comparable with the other rows due to model size, but we report them for completeness. Best average performances are indicated in **bold**.

and neural L-PCFG is the modeling of headedness and the dependency between head word and generated non-terminals or preterminals. We apply the same hyperparameters and techniques, including number of non-terminals and preterminals, initialization, curriculum learning and variational training to compound PCFGs for a fair comparison. Because compound PCFGs have no notion of dependencies, we extract dependencies from the compound PCFG with three kinds of heuristic head rules: left-headed, right-headed, and large-headed. Left-/right-headed mean always choosing the root of the left/right child constituent as the root of the parent constituent, whereas large-headedness is generated by a heuristic rule which chooses the root of larger child constituent as the root of the parent constituent. Among these, we choose the method that obtains the best parsing accuracy on the dev set (making these results an oracle with access to *more* information than our proposed method).

Dependency Model with Valence (DMV) The DMV (Klein and Manning, 2004) is a model for unsupervised dependency parsing, where *valence* stands for the number of arguments controlled by a head word. The choices to attach words as children are conditioned on the head words and valences. As shown in Smith (2006), the DMV model can be expressed as a head-driven context-free grammar with a set of generation rules and scores, where the non-terminals represent the valence of head words. For example, ''L[CHASING] $\rightarrow$ L$_0$[IS] R[CHASING]'' denotes that a left-hand constituent with full left valence produces a word and a constituent with full right valence. Therefore, it could be seen as a special case of lexicalized PCFG where the generation rules provide inductive biases for dependency parsing but are also restricted—for example, a void-valence constituent cannot produce a full-valence constituent with the same head.

Note that DMV uses far fewer parameters than the PCFG-based models, $\mathcal{O}(|\mathcal{P}|^2)$. The neural L-PCFG uses a similar number of parameters as we do, $\mathcal{O}(n(|\mathcal{P}| + |\mathcal{N}|) + n^2)$.

We compare models under two settings: (1) with gold tag information and (2) without it, denoted by ✓ and ✗, respectively in Table 2. To use gold tag information in training the neural L-PCFG, we assign the 19 most frequent tags as categories and combine the rest into a 20th ''other'' category. These categories are used as supervision for the preterminals. In this setting, instead of optimizing the log probability of the sentence, we optimize the log joint probability of the sentence and the tags.

5.3 Quantitative Results

First, in this section, we present and discuss quantitative results, as shown in Table 2.

5.3.1 Main Results

First comparing neural L-PCFGs with compound PCFGs, we can see that L-PCFGs perform slightly better on phrase structure prediction and achieve much better dependency accuracy. This shows that (1) lexical dependencies contribute somewhat to the learning of phrase structure, and (2) the

	PRPN	ON	Compound PCFG	Neural L-PCFG
SBAR	50.0%	51.2%	42.36%	**53.60%**
NP	59.2%	64.5%	59.25%	**67.38%**
VP	46.7%	41.0%	39.50%	**48.58%**
PP	57.2%	54.4%	62.66%	**65.25%**
ADJP	44.3%	38.1%	49.16%	**49.83%**
ADVP	32.8%	31.6%	50.58%	**58.86%**

Table 3: Fraction of ground truth constituents that were predicted as a constituent by the models broken down by label (i.e., label recall). Results of PRPN and ON are from Kim et al. (2019).

head rules learned by neural L-PCFGs are significantly more accurate than the heuristics that we applied to standard compound PCFGs. We also find that GloVe embeddings can help (unsupervised) dependency parsing, but do not benefit constituency parsing.

Next, we can compare the dependency induction accuracy of the neural L-PCFGs with the DMV. The results indicate that neural L-PCFGs without gold tags achieve even better accuracy than DMV with gold tags on both directed accuracy and undirected accuracy. As discussed before, DMV can be seen as a special case of L-PCFG where the attachment of children is conditioned on the valence of the parent tag, while in L-PCFG the generated head directions are conditioned on the parent non-terminal and the head word, which is more general. Comparatively positive results show that conditioning on generation rules not only is more general but also yields a better prediction of attachment.

Table 3 shows label-level recall (i.e., unlabeled recall of constituents annotated by each non-terminal). We observe that the neural L-PCFG outperforms all baselines on these frequent constituent categories.

5.3.2 Impact of Factorization

Table 4 compares the effects of three alternate factorizations of $g_\theta(A[\alpha] \rightarrow B[\alpha]C[\beta], \boldsymbol{z})$:

$$g_\theta(A[\alpha] \rightarrow B[\alpha]C[\beta], \boldsymbol{z}) = p_{\boldsymbol{z}}(C \rightarrow \beta) +$$

F I: $\log p_{\boldsymbol{z}}(B,C \mid A) + \log p_{\boldsymbol{z}}(\curvearrowleft \mid A \rightarrow BC)$

F II: $\log p_{\boldsymbol{z}}(B, C, \curvearrowleft \mid A, \alpha)$

F III: $\log p_{\boldsymbol{z}}(B, \curvearrowleft \mid A, \alpha) + \log p_{\boldsymbol{z}}(C \mid A, B, \alpha)$

Factorization I assumes that the child non-terminals do not depend on the head lexical item, which influences the parsing result significantly. Although Factorization II is as general as our proposed method, it uses separate representations for different directions, $\boldsymbol{v}_{BC\curvearrowleft}$ and $\boldsymbol{v}_{BC\curvearrowright}$. Factorization III assumes the independence between direction and dependent non-terminals. These results indicate that our factorization strikes a good balance between modeling lexical dependencies and directionality, and avoiding over-parameterization of the model that may lead to sparsity and difficulties in learning.

	DAS	UAS	F1
Neural L-PCFG	35.5	51.4	44.5
w/ **xavier init**	27.2	47.6	43.6
w/ **Factorization I**	16.4	33.3	25.7
w/ **Factorization II**	22.3	42.7	39.6
w/ **Factorization III**	25.9	46.9	34.7

Table 4: An ablation of dependency and constituency parsing results on the validation set with different settings of neural L-PCFG. All models are trained with GloVe word embeddings and without gold tags. ''w/ **xavier init**'' means that preterminals are not initialized by clustering centroids by xavier normal distribution. ''w/ Factorization N'' represents different factorization methods (§5.3.2).

5.4 Qualitative Analysis

We analyze our best model without gold tags in detail. Figure 4 visualizes the alignment between our induced non-terminals and gold constituent labels on the overlapping constituents of induced trees and the ground-truth. For each constituent label, we show the frequency of it annotating the same span of each non-terminal. We observe from the first map that a clear alignment between certain linguistic labels and induced non-terminals (e.g., VP and NT-4, S and NT-2, PP, and NT-8). But for other non-terminals, there's no clear alignment with induced classes. One hypothesis for this diffusion is the diversity of the syntactic roles of these constituents. To investigate this, we zoom in on noun phrases in the second map, and observe that NP-SBJ, NP-TMP, and NP-MNR are combined into a single non-terminal NT-5 in the induced grammar, and that NP, NP-PRD, and NP-CLR corresponds to NT-2, NT-6, and NT-0, respectively.

Figure 4: Alignment between all induced non-terminals (x-axis) and gold non-terminals annotated in the PTB (y-axis). In the upper figure, we show the seven most frequent gold non-terminals, and list them by frequency from left to right. For each gold non-terminal, we show the proportion of each induced non-terminal. In the lower map, we breakdown the results of the noun phrase (NP) into subcategories. Darker color indicates higher proportion, and vice versa.

We also include an example set of parses for comparing the DMV and neural L-PCFG in Table 5. Note that DMV uses ‘‘to’’ as the head of ‘‘know’’, the neural L-PCFG correctly inverts this relationship to produce a parse that is better aligned with the gold tree. One of the possible reasons that the DMV tends to use ‘‘to’’ as the head is that DMV has to carry the information that the verb is in the infinitive form, which will be lost if it uses ‘‘know’’ as the head. In our model, however, such information is contained in the types of non-terminals. In this way, our model uses the open class word ‘‘know’’ as the root. Note that we also illustrate a similar failure case in this example. Neural L-PCFG uses ‘‘if’’ as the head of the if-clause, which is probably due to the independency between the root of the if-clause and ‘‘know’’.

A common mistake made by the neural L-PCFG is treating auxiliary verbs like adjectives that combine with the subject instead of modifying verb phrases. For example, the neural L-PCFG parses ‘‘*...the exchange will look at the performance...*’’ as ‘‘((the exchange) will) (look (at (the performance)))’’, whereas the compound PCFG produces the correct parse ‘‘((the exchange) (will (look (at (the performance)))))’’. A possible reason for this mistake is that English verb phrases are commonly left-headed, which makes attaching an auxiliary verb less probable as the left child of a verb phrase. This type of error may stem from the model’s inability to assess the semantic function of auxiliary verbs (Bisk and Hockenmaier, 2015).

6 Related Work

Dependency vs Constituency Induction The decision to focus on modeling dependencies and constituencies has largely split the grammar induction community into two camps. The most popular approach has been focused on dependency formalisms (Klein and Manning, 2004; Spitkovsky et al., 2010, 2011, 2013; Mareček and Straka, 2013; Jiang et al., 2016; Tran and Bisk, 2018), whereas a second community has focused on inducing constituencies (Lane and Henderson, 2001; Ponvert et al., 2011; Golland et al., 2012; Jin et al., 2018). Induced constituencies can in the case of CCG (Bisk and Hockenmaier, 2012, 2013) produce dependencies, but unlike our proposal, existing approaches do not jointly model both representations. CFGs have been used for decades to represent, analyze and model the phrase structure of language (Chomsky, 1956; Pullum and Gazdar, 1982; Lari and Young, 1990; Klein and Manning, 2002; Bod, 2006).

Similarly, the compound PCFG (Kim et al., 2019), which we extend, falls into this camp of models that induce only phrase-structure grammar. However, in this paper we demonstrate a novel lexically informed neural parameterization that extends their model to induce a *unified* phrase-structure and dependency-structure grammar.

Unifying Phrase Structure and Dependency Grammar Head-driven phrase structure grammar (Sag and Pollard, 1987) and lexicalized tree adjoining grammar (Schabes et al., 1988) are

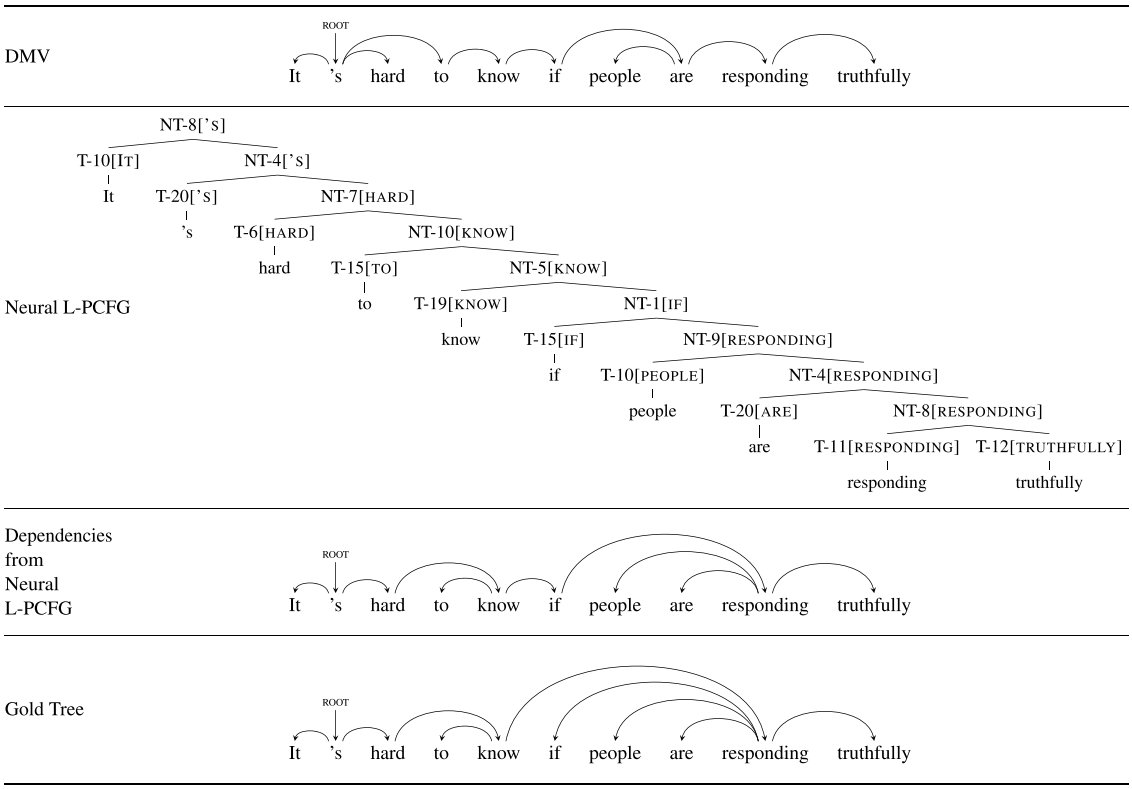


Table 5: Comparison between Neural L-PCFG and DMV on a case from PTB training set.

approaches to representing dependencies directly in phrase structure.

The notion that abstract syntactic structure should provide scaffolding for dependencies, and that lexical dependencies should provide a semantic guide for syntax, was most famously explored in Collins (2003) through the introduction of an L-PCFG. In addition, Carroll and Rooth (1998) explored the problem of head induction in L-PCFG; Charniak and Johnson (2005) improves L-PCFGs with coarse-to-fine parsing and reranking. Recently, (Green and Žabokrtskỳ, 2012; Ren et al., 2013; Yoshikawa et al., 2017) explored various methods to jointly infer phrase structure and dependencies.

Klein and Manning (2004) show that a combined DMV and CCM (Klein and Manning, 2002) model, where each tree is scored with the product of the probabilities from the individual models, outperforms either individual model. These results demonstrate that the two varieties of unsupervised parsing models can benefit from ensembling. In contrast, our model considers both phrase-and dependency structure jointly. Seginer (2007) introduces a parser using a representation like dependency structure, which helps constituency parsing.

Bikel (2004)'s analysis of prominent models at the time found that lexical dependencies provided only very minor benefits and that choosing appropriate smoothing parameters was key to performance and robustness. Hackenmaier and Steedman (2002) also explores this for combinatorial categorial grammar (CCG), showing that lexical sparsity and smoothing have dramatic effects regardless of the formalism. The sparsity and expense of lexicalized PCFGs have precluded their use in most contexts, though Prescher (2005) proposes a latent-head model to alleviate the sparse data problem.

7 Conclusion

In this paper, we propose neural L-PCFG, a neural parameterization method for lexicalized PCFGs, for both unsupervised dependency parsing and constituency parsing. We also provide a variational inference method to train our model. By modeling both representations together, our approach outperforms methods specially designed for either grammar formalism alone.

Importantly, our work also adds novel insights for the unsupervised grammar induction literature by probing the role that factorizations

and initialization have on model performance. Different factorizations of the same probability distribution can lead to dramatically different performance and should be viewed as playing an important role in the inductive bias of learning syntax. Additionally, where others have used pretrained word vectors before, we show that they too contain abstract syntactic information which can bias learning.

Finally, although out of scope for one paper, our results point to several interesting potential roads forward, including the study of the effectiveness of jointly modeling constituency-dependency representations on freer word order languages, and whether other distributed word presentations (e.g., large-scale transformers) might provide even stronger syntactic signals for grammar induction.

Despite the demonstrated success of lexical dependencies, it should be noted that these are only unilexical dependencies, in contrast to bilexical dependencies, which also consider the dependencies between head and dependent words. Modeling these dependencies would require marginalizing over all possible dependents for each span-head pair. In this case, the time complexity of exhaustive dynamic programming over one sentence would become $\mathcal{O}(L^5|\mathcal{N}|(|\mathcal{N}|+|\mathcal{P}|)^2)$, where L stands for the length of the sentence. Assuming enough parallel workers, this time complexity can be reduced to $\mathcal{O}(L)$, but it still requires $\mathcal{O}(L^4|\mathcal{N}|(|\mathcal{N}|+|\mathcal{P}|)^2)$ auxiliary space. In contrast, our model runs for $\mathcal{O}(L^4|\mathcal{N}|(|\mathcal{N}|+|\mathcal{P}|)^2)$. Assuming enough parallel workers, this time complexity can also be reduced to $\mathcal{O}(L)$, but still requires $\mathcal{O}(L^3|\mathcal{N}|(|\mathcal{N}|+|\mathcal{P}|)^2)$ auxiliary space. These auxiliary data can be stored in a 32GB graphic card in our experiments (e.g., with $N = 20$), whereas the bilexical model cannot. There are several potential methods to side-step this problem, including the use of sampling in lieu of dynamic programming, using heuristic methods to prune the grammar, and designing acceleration methods on GPU (Hall et al., 2014).

Acknowledgments

This work was supported by the DARPA GAILA project (award HR00111990063), and some experiments made use of computation credits graciously provided by Amazon AWS. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. The authors would like to thank Junxian He and Yoon Kim for helpful feedback about the project.

References

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 41–48. **DOI:** https://doi.org/10.1145/1553374.1553380

Daniel M. Bikel. 2004. Intricacies of Collins' parsing model. *Computational Linguistics*, 30(4):479–511. **DOI:** https://doi.org/10.1162/0891201042544929

Yonatan Bisk and Julia Hockenmaier. 2012. Simple robust grammar induction with combinatory categorial grammars. In *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence (AAAI-12)*, pages 1643–1649. Toronto, Canada.

Yonatan Bisk and Julia Hockenmaier. 2013. An HDP model for inducing combinatory categorial grammars. *Transactions of the Association for Computational Linguistics*, pages 75–88. **DOI:** https://doi.org/10.1162/tacl_a_00211

Yonatan Bisk and Julia Hockenmaier. 2015. Probing the linguistic strengths and limitations of unsupervised grammar induction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. **DOI:** https://doi.org/10.3115/v1/P15-1135

Rens Bod. 2006. An all-subtrees approach to unsupervised parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 865–872. **DOI:** https://doi.org/10.3115/1220175.1220284

Glenn Carroll and Eugene Charniak. 1992. *Two experiments on learning probabilistic dependency grammars from corpora*. Department of Computer Science, Univ.

Glenn Carroll and Mats Rooth. 1998. Valence induction with a head-lexicalized PCFG. *arXiv preprint cmp-lg/9805001*.

Eugene Charniak. 1996. Tree-bank grammars. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1031–1036.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 173–180. Association for Computational Linguistics. **DOI:** https://doi.org/10.3115/1219840.1219862

Noam Chomsky. 1956. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124. IEEE. **DOI:** https://doi.org/10.1109/TIT.1956.1056813

Alexander Clark. 2001. Unsupervised induction of stochastic context-free grammars using distributional clustering. In *Proceedings of the 2001 Workshop on Computational Natural Language Learning-Volume 7*, page 13. Association for Computational Linguistics. **DOI:** https://doi.org/10.3115/1117822.1117831

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):589–637. **DOI:** https://doi.org/10.1162/089120103322753356

Marie-Catherine De Marneffe and Christopher D. Manning. 2008. Stanford typed dependencies manual, Technical report, Stanford University. **DOI:** https://doi.org/10.3115/1608858.1608859

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256.

Dave Golland, John DeNero, and Jakob Uszkoreit. 2012. A feature-rich constituent context model for grammar induction. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 17–22, Jeju Island, Korea. Association for Computational Linguistics.

Nathan David Green and Zdeněk Žabokrtskỳ. 2012. Hybrid combination of constituency and dependency trees into an ensemble dependency parser. In *Proceedings of the Workshop on Innovative Hybrid Approaches to the Processing of Textual Data*, pages 19–26. Association for Computational Linguistics.

David Hall, Taylor Berg-Kirkpatrick, and Dan Klein. 2014. Sparser, better, faster GPU parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 208–217, Baltimore, Maryland. Association for Computational Linguistics. **DOI:** https://doi.org/10.3115/v1/P14-1020

Wenjuan Han, Yong Jiang, and Kewei Tu. 2019. Enhancing unsupervised generative dependency parser with contextual information. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5315–5325.

Junxian He, Graham Neubig, and Taylor Berg-Kirkpatrick. 2018. Unsupervised learning of syntactic structure with invertible neural projections. *arXiv preprint arXiv:1808.09111*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780. **DOI:** https://doi.org/10.1162/neco.1997.9.8.1735, **PMID:** 9377276

Julia Hockenmaier and Mark Steedman. 2002. Generative models for statistical parsing with combinatory categorial grammar. In *Proceedings of 40th Annual Meeting of the Association*

for Computational Linguistics, pages 335–342. Philadelphia, Pennsylvania, USA. **DOI:** https://doi.org/10.3115/1073083.1073139

Yong Jiang, Wenjuan Han, and Kewei Tu. 2016. Unsupervised neural dependency parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 763–771. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/D16-1073

Lifeng Jin, Finale Doshi-Velez, Timothy Miller, William Schuler, and Lane Schwartz. 2018. Unsupervised grammar induction with depth-bounded PCFG. *Transactions of the Association for Computational Linguistics*, 6:211–224. **DOI:** https://doi.org/10.1162/tacl_a_00016

Lifeng Jin, Finale Doshi-Velez, Timothy Miller, Lane Schwartz, and William Schuler. 2019. Unsupervised learning of PCFGS with normalizing flow. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2442–2452.

Yoon Kim, Chris Dyer, and Alexander M. Rush. 2019. Compound probabilistic context-free grammars for grammar induction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2369–2385. **DOI:** https://doi.org/10.18653/v1/P19-1228, **PMID:** 31697821, **PMCID:** PMC6539514

Diederik P. Kingma and Max Welling. 2014. Auto-encoding variational bayes. In *Proceedings of ICLR*.

Dan Klein and Christopher D. Manning. 2002. A generative constituent-context model for improved grammar induction. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 128–135. Association for Computational Linguistics. **DOI:** https://doi.org/10.3115/1073083.1073106

Dan Klein and Christopher D. Manning. 2004. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 478. Association for Computational Linguistics. **DOI:** https://doi.org/10.3115/1218955.1219016

Peter C.R. Lane and James B. Henderson. 2001. Incremental syntactic parsing of natural language corpora with simple synchrony networks. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):219–231. **DOI:** https://doi.org/10.1109/69.917562

Karim Lari and Steve J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech & Language*, 4(1):35–56. **DOI:** https://doi.org/10.1016/0885-2308(90)90022-X

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605.

James MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. Oakland, CA, USA.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330. **DOI:** https://doi.org/10.21236/ADA273556

David Mareček and Milan Straka. 2013. Stop-probability estimates computed on a large corpus improve unsupervised dependency parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 281–290, Sofia, Bulgaria. Association for Computational Linguistics.

Mark A. Paskin. 2002. Grammatical bigrams. In *Advances in Neural Information Processing Systems*, pages 91–97.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods*

in Natural Language Processing (EMNLP), pages 1532–1543. **DOI:** https://doi.org/10.3115/v1/D14-1162

Elias Ponvert, Jason Baldridge, and Katrin Erk. 2011. Simple unsupervised grammar induction from raw text with cascaded finite state models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1077–1086, Portland, Oregon, USA. Association for Computational Linguistics.

Detlef Prescher. 2005. Head-driven PCFGs with latent-head statistics. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 115–124, Vancouver, British Columbia. Association for Computational Linguistics.

Geoffrey K. Pullum and Gerald Gazdar. 1982. Natural languages and context-free languages. *Linguistics and Philosophy*, 4(4):471–504. **DOI:** https://doi.org/10.1007/BF00360802

Xiaona Ren, Xiao Chen, and Chunyu Kit. 2013. Combine constituent and dependency parsing via reranking. In *Twenty-Third International Joint Conference on Artificial Intelligence*.

Herbert Robbins and others. 1951. Asymptotically subminimax solutions of compound statistical decision problems. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*. The Regents of the University of California.

Ivan A. Sag and Carl Pollard. 1987. Information-based syntax and semantics. *CSLI Lecture Notes*, 13.

Yves Schabes, Anne Abeille, and Aravind K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: application to tree adjoining grammars. In *Proceedings of the 12th Conference on Computational Linguistics-Volume 2*, pages 78–583. Association for Computational Linguistics. **DOI:** https://doi.org/10.3115/991719.991757

Yoav Seginer. 2007. Fast unsupervised incremental parsing. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 384–391.

Noah Smith. 2006. *Novel estimation methods for unsupervised discovery of latent structure in natural language text*. Ph.D. thesis, Johns Hopkins University.

Valentin I. Spitkovsky, Hiyan Alshawi, Angel X. Chang, and Daniel Jurafsky. 2011. Unsupervised dependency parsing without gold part-of-speech tags. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1281–1290, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. 2010. From baby steps to leapfrog: How ''less is more'' in unsupervised dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 751–759. Los Angeles, California.

Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. 2013. Breaking out of local optima with count transforms and model recombination: A study in grammar induction. In *Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Ke Tran and Yonatan Bisk. 2018. Inducing grammars with and for neural machine translation. In *Proceedings of the 2nd Workshop on Neural Machine Translation*. Melbourne, Australia.

Masashi Yoshikawa, Hiroshi Noji, and Yuji Matsumoto. 2017. A* CCG parsing with a supertag and dependency factored model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 277–287, Vancouver, Canada. Association for Computational Linguistics.

Deniz Yuret. 1998. Discovery of linguistic relations using lexical attraction. *arXiv preprint cmp-lg/9805009*.