Gaussian Mixture Latent Vector Grammars

Yanpeng Zhao, Liwen Zhang, Kewei Tu
School of Information Science and Technology,
ShanghaiTech University, Shanghai, China
{zhaoyp1,zhanglw1,tukw}@shanghaitech.edu.cn

Abstract

We introduce Latent Vector Grammars (LVeGs), a new framework that extends latent variable grammars such that each nonterminal symbol is associated with a continuous vector space representing the set of (infinitely many) subtypes of the nonterminal. We show that previous models such as latent variable grammars and compositional vector grammars can be interpreted as special cases of LVeGs. We then present Gaussian Mixture LVeGs (GM-LVeGs), a new special case of LVeGs that uses Gaussian mixtures to formulate the weights of production rules over subtypes of nonterminals. A major advantage of using Gaussian mixtures is that the partition function and the expectations of subtype rules can be computed using an extension of the inside-outside algorithm, which enables efficient inference and learning. We apply GM-LVeGs to part-of-speech tagging and constituency parsing and show that GM-LVeGs can achieve competitive accuracies. Our code is available at https://github.com/zhaoyanpeng/lveg.

1 Introduction

In constituency parsing, refining coarse syntactic categories of treebank grammars (Charniak, 1996) into fine-grained subtypes has been proven effective in improving parsing results. Previous approaches to refining syntactic categories use tree annotations (Johnson, 1998), lexicalization (Charniak, 2000; Collins, 2003), or linguistically motivated category splitting (Klein and Manning, 2003). Matsuzaki et al. (2005) introduce latent variable grammars, in which each syntactic category (represented by a nonterminal) is split into a fixed number of subtypes and a discrete latent variable is used to indicate the subtype of the nonterminal when it appears in a specific parse tree. Since the latent variables are not observable in treebanks, the grammar is learned using expectation-maximization. Petrov et al. (2006) present a split-merge approach to learning latent variable grammars, which hierarchically splits each nonterminal and merges ineffective splits. Petrov and Klein (2008b) further allow a nonterminal to have different splits in different production rules, which results in a more compact grammar.

Recently, neural approaches become very popular in natural language processing (NLP). An important technique in neural approaches to NLP is to represent discrete symbols such as words and syntactic categories with continuous vectors or embeddings. Since the distances between such vector representations often reflect the similarity between the corresponding symbols, this technique facilitates more informed smoothing in learning functions of symbols (e.g., the probability of a production rule). In addition, what a symbol represents may subtly depend on its context, and a continuous vector representation has the potential of representing each instance of the symbol in a more precise manner. For constituency parsing, recursive neural networks (Socher et al., 2011) and their extensions such as compositional vector grammars (Socher et al., 2013) can be seen as representing nonterminals in a context-free grammar with continuous vectors. However, exact inference in these models is intractable.

In this paper, we introduce latent vector grammars (LVeGs), a novel framework of grammars with fine-grained nonterminal subtypes. A LVeG associates each nonterminal with a continuous vector space that represents the set of (infinitely many) subtypes of the nonterminal. For each instance of a nonterminal that appears in a parse tree, its subtype is represented by a latent vector. For each production rule over nonterminals, a non-negative continuous function specifies the weight of any fine-grained production rule over subtypes of the nonterminals. Compared with latent variable grammars which assume a small fixed number of subtypes for each nonterminal, LVeGs assume an unlimited number of subtypes and are potentially more expressive. By having weight functions of varying smoothness for different production rules, LVeGs can also control the level of subtype granularity for different productions, which has been shown to improve the parsing accuracy (Petrov and Klein, 2008b). In addition, similarity between subtypes of a nonterminal can be naturally modeled by the distance between the corresponding vectors, so by using continuous and smooth weight functions we can ensure that similar subtypes will have similar syntactic behaviors.

We further present Gaussian Mixture LVeGs (GM-LVeGs), a special case of LVeGs that uses mixtures of Gaussian distributions as the weight functions of fine-grained production rules. A major advantage of GM-LVeGs is that the partition function and the expectations of fine-grained production rules can be computed using an extension of the inside-outside algorithm. This makes it possible to efficiently compute the gradients during discriminative learning of GM-LVeGs. We evaluate GM-LVeGs on part-of-speech tagging and constituency parsing on a variety of languages and corpora and show that GM-LVeGs achieve competitive results.

It shall be noted that many modern state-of-the-art constituency parsers predict how likely a constituent is based on not only local information (such as the production rules used in composing the constituent), but also contextual information of the constituent. For example, the neural CRF parser (Durrett and Klein, 2015) looks at the words before and after the constituent; and RNNG (Dyer et al., 2016) looks at the constituents that are already predicted (in the stack) and the words that are not processed (in the buffer). In this paper, however, we choose to focus on the basic framework and algorithms of LVeGs and leave the incorporation of contextual information for future work. We believe that by laying a solid foundation for LVeGs, our work can pave the way for many interesting extensions of LVeGs in the future.

2 Latent Vector Grammars

A latent vector grammar (LVeG) considers subtypes of nonterminals as continuous vectors and associates each nonterminal with a latent vector space representing the set of its subtypes. For each production rule, the LVeG defines a weight function over the subtypes of the nonterminal involved in the production rule. In this way, it models the space of refinements of the production rule.

2.1 Model Definition

A latent vector grammar is defined as a 5-tuple $\mathcal{G} = (N, S, \Sigma, R, W)$, where N is a finite set of nonterminal symbols, $S \in N$ is the start symbol, Σ is a finite set of terminal symbols such that $N \cap \Sigma = \varnothing$, R is a set production rules of the form $X \rightarrow \gamma$ where $X \in N$ and $\gamma \in (N \cup \Sigma)^*$, W is a set of rule weight functions indexed by production rules in R (to be defined below). In the following discussion, we consider R in the Chomsky normal form (CNF) for clarity of presentation. However, it is straightforward to extend our formulation to the general case.

Unless otherwise specified, we always use capital letters $A, B, C, \ldots$ for nonterminal symbols and use bold lowercase letters $\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots$ for their subtypes. Note that subtypes are represented by continuous vectors. For a production rule of the form $A \rightarrow BC$, its weight function is $W_{A\rightarrow BC}(\mathbf{a}, \mathbf{b}, \mathbf{c})$. For a production rule of the form $A \rightarrow w$ where $w \in \Sigma$, its weight function is $W_{A\rightarrow w}(\mathbf{a})$. The weight functions should be non-negative, continuous and smooth, and hence fine-grained production rules of similar subtypes of a nonterminal would have similar weight assignments. Rule weights can be normalized such that $\sum_{B,C} \int_{\mathbf{b},\mathbf{c}} W_{A\rightarrow BC}(\mathbf{a}, \mathbf{b}, \mathbf{c}) d\mathbf{b} d\mathbf{c} = 1$, which leads to a probabilistic context-free grammar (PCFG). Whether the weights are normalized or not leads to different model classes and accordingly different estimation methods. However, the two model classes are proven equivalent by Smith and Johnson (2007).

2.2 Relation to Other Models

Latent variable grammars (LVGs) (Matsuzaki et al., 2005; Petrov et al., 2006) associate each nonterminal with a discrete latent variable, which is used to indicate the subtype of the nonterminal when it appears in a parse tree. Through nonterminal-splitting and the expectation-maximization algorithm, fine-grained production rules can be automatically induced from a treebank.

We show that LVGs can be seen as a special case of LVeGs. Specifically, we can use one-hot vectors in LVeGs to represent latent variables in LVGs and define weight functions in LVeGs accordingly. Consider a production rule $r : A \rightarrow BC$. In a LVG, each nonterminal is split into a number of subtypes. Suppose A, B, and C are split into n_A, n_B, and n_C subtypes respectively. a_x is the x-th subtype of A, b_y is the y-th subtype of B, and c_z is the z-th subtype of C. $a_x \rightarrow b_y c_z$ is a fine-grained production rule of $A \rightarrow BC$, where $x = 1, \ldots, n_A$, $y = 1, \ldots, n_B$, and $z = 1, \ldots, n_C$. The probabilities of all the fine-grained production rules can be represented by a rank-3 tensor $\Theta_{A\rightarrow BC} \in \mathbb{R}^{n_A \times n_B \times n_C}$. To cast the LVG as a LVeG, we require that the latent vectors in the LVeG must be one-hot vectors. We achieve this by defining weight functions that output zero if any of the input vectors is not one-hot. Specifically, we define the weight function of the production rule $A \rightarrow BC$ as:

$$W_r(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \sum_{x,y,z} \Theta_{A\rightarrow BC}\mathbf{cba} \times (\delta(\mathbf{a} - \mathbf{a}_x) \\ \times \delta(\mathbf{b} - \mathbf{b}_y) \times \delta(\mathbf{c} - \mathbf{c}_z)) \,, \quad (1)$$

where $\delta(\cdot)$ is the Dirac delta function, $\mathbf{a}_x \in \mathbb{R}^{n_A}$, $\mathbf{b}_y \in \mathbb{R}^{n_B}$, $\mathbf{c}_z \in \mathbb{R}^{n_C}$ are one-hot vectors (which are zero everywhere with the exception of a single 1 at the x-th index of $\mathbf{a}_x$, the y-th index of $\mathbf{b}_y$, and the z-th index of $\mathbf{c}_z$) and $\Theta_{A\rightarrow BC}$ is multiplied sequentially by $\mathbf{c}$, $\mathbf{b}$, and $\mathbf{a}$.

Compared with LVGs, LVeGs have the following advantages. While a LVG contains a finite, typically small number of subtypes for each nonterminal, a LVeG uses a continuous space to represent an infinite number of subtypes. When equipped with weight functions of sufficient complexity, LVeGs can represent more fine-grained syntactic categories and production rules than LVGs. By controlling the complexity and smoothness of the weight functions, a LVeG is also capable of representing any level of subtype granularity. Importantly, this allows us to change the level of subtype granularity for the same nonterminal in different production rules, which is similar to multi-scale grammars (Petrov and Klein, 2008b). In addition, with a continuous space of subtypes in a LVeG, similarity between subtypes can be naturally modeled by their distance in the space and can be automatically learned from data. Consequently, with continuous and smooth weight functions, fine-grained production rules over similar subtypes would have similar weights in LVeGs, eliminating the need for the extra smoothing steps that are necessary in training LVGs.

Compositional vector grammars (CVGs) (Socher et al., 2013), an extension of recursive neural networks (RNNs) (Socher et al., 2011), can also be seen as a special case of LVeGs. For a production rule $r : A \rightarrow BC$, a CVG can be interpreted as specifying its weight function $W_r(\mathbf{a}, \mathbf{b}, \mathbf{c})$ in the following way. First, a neural network f indexed by B and C is used to compute a parent vector $\mathbf{p} = f_{BC}(\mathbf{b}, \mathbf{c})$. Next, the score of the parent vector is computed using a base PCFG and a vector $\mathbf{v}_{BC}$:

$$s(\mathbf{p}) = \mathbf{v}_{BC}^T \mathbf{p} + \log P(A \rightarrow BC) \,, \quad (2)$$

where $P(A \rightarrow BC)$ is the rule probability from the base PCFG. Then, the weight function of the production rule $A \rightarrow BC$ is defined as:

$$W_r(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \exp\left(s(\mathbf{p})\right) \times \delta(\mathbf{a} - \mathbf{p}) \,. \quad (3)$$

This form of weight functions in CVGs leads to point estimation of latent vectors in a parse tree, i.e., for each nonterminal in a given parse tree, only one subtype in the whole subtype space would lead to a non-zero weight of the parse. In addition, different parse trees of the same substring typically lead to different point estimations of the subtype vector at the root nonterminal. Consequently, CVGs cannot use dynamic programming for inference and hence have to resort to greedy search or beam search.

3 Gaussian Mixture LVeGs

A major challenge in applying LVeGs to parsing is that it is impossible to enumerate the infinite number of subtypes. Previous work such as CVGs resorts to point estimation and greedy search. In this section we present Gaussian Mixture LVeGs (GM-LVeGs), which use mixtures of Gaussian distributions as the weight functions in LVeGs. Because Gaussian mixtures have the nice property of being closed under product, summation, and marginalization, we can compute the partition function and the expectations of fine-grained production rules using dynamic programming. This in turn makes efficient learning and parsing possible.

3.1 Representation

In a GM-LVeG, the weight function of a production rule r is defined as a Gaussian mixture containing K_r mixture components:

$$W_r(\mathbf{r}) = \sum_{k=1}^{K_r} \rho_{r,k} \mathcal{N}(\mathbf{r}|\boldsymbol{\mu}_{r,k}, \boldsymbol{\Sigma}_{r,k}) \,, \tag{4}$$

where $\mathbf{r}$ is the concatenation of the latent vectors of the nonterminals in r, which denotes a fine-grained production rule of r. $\rho_{r,k} > 0$ is the k-th mixture weight (the mixture weights do not necessarily sum up to 1), $\mathcal{N}(\mathbf{r}|\boldsymbol{\mu}_{r,k}, \boldsymbol{\Sigma}_{r,k})$ is the k-th Gaussian distribution parameterized by mean $\boldsymbol{\mu}_{r,k}$ and covariance matrix $\boldsymbol{\Sigma}_{r,k}$, and K_r is the number of mixture components, which can be different for different production rules. Below we write $\mathcal{N}(\mathbf{r}|\boldsymbol{\mu}_{r,k}, \boldsymbol{\Sigma}_{r,k})$ as $\mathcal{N}_{r,k}(\mathbf{r})$ for brevity. Given a production rule of the form $A \rightarrow BC$, the GM-LVeG expects $\mathbf{r} = [\mathbf{a};\mathbf{b};\mathbf{c}]$ and $\mathbf{a},\mathbf{b},\mathbf{c} \in \mathbb{R}^d$, where d is the dimension of the vectors $\mathbf{a},\mathbf{b},\mathbf{c}$. We use the same dimension for all the subtype vectors.

For the sake of computational efficiency, we use diagonal or spherical Gaussian distributions, whose covariance matrices are diagonal, so that the inverse of covariance matrices in Equation 15–16 can be computed in linear time. A spherical Gaussian has a diagonal covariance matrix where all the diagonal elements are equal, so it has fewer free parameters than a diagonal Gaussian and results in faster learning and parsing. We empirically find that spherical Gaussians lead to slightly better balance between the efficiency and the parsing accuracy than diagonal Gaussians.

3.2 Parsing

The goal of parsing is to find the most probable parse tree T^* with unrefined nonterminals for a sentence $\mathbf{w}$ of n words $w_{1:n} = w_1 \dots w_n$. This is formally defined as:

$$T^* = \operatorname*{argmax}_{T \in G(\mathbf{w})} P(T|\mathbf{w}) \,, \tag{5}$$

where $G(\mathbf{w})$ denotes the set of parse trees with unrefined nonterminals for $\mathbf{w}$. In a PCFG, T^* can be found using dynamic programming such as the CYK algorithm. However, parsing becomes intractable with LVeGs, and even with LVGs, the special case of LVeGs.

A common practice in parsing with LVGs is to use max-rule parsing (Petrov et al., 2006; Petrov and Klein, 2007). The basic idea of max-rule parsing is to decompose the posteriors over parses into the posteriors over production rules approximately. This requires calculating the expected counts of unrefined production rules in parsing the input sentence. Since Gaussian mixtures are closed under product, summation, and marginalization, in GM-LVeGs the expected counts can be calculated using the inside-outside algorithm in the following way. Given a sentence $w_{1:n}$, we first calculate the inside score $s^A_{\mathbf{I}}(\mathbf{a}, i, j)$ and outside score $s^A_{\mathbf{O}}(\mathbf{a}, i, j)$ for a nonterminal A over a span $w_{i:j}$ using Equation 6 and Equation 7 in Table 1 respectively. Note that both $s^A_{\mathbf{I}}(\mathbf{a}, i, j)$ and $s^A_{\mathbf{O}}(\mathbf{a}, i, j)$ are mixtures of Gaussian distributions of the subtype vector $\mathbf{a}$. Next, using Equation 8 in Table 1, we calculate the score $s(A \rightarrow BC, i, k, j)$ $(1 \leq i \leq k < j \leq n)$, where $\langle A \rightarrow BC, i, k, j \rangle$ represents a production rule $A \rightarrow BC$ with nonterminals A, B, and C spanning words $w_{i:j}$, $w_{i,k}$, and $w_{k+1:j}$ respectively in the sentence $w_{1:n}$. Then the expected count (or posterior) of $\langle A \rightarrow BC, i, k, j \rangle$ is calculated as:

$$q(A \rightarrow BC, i, k, j) = \frac{s(A \rightarrow BC, i, k, j)}{s_{\mathbf{I}}(S, 1, n)} \,, \tag{9}$$

where $s_{\mathbf{I}}(S, 1, n)$ is the inside score for the start symbol S spanning the whole sentence $w_{1:n}$. After calculating all the expected counts, we can use the MAX-RULE-PRODUCT algorithm (Petrov and Klein, 2007) for parsing, which returns a parse with the highest probability that all the production rules are correct. Its objective function is given by

$$T^*_q = \operatorname*{argmax}_{T \in G(\mathbf{w})} \prod_{e \in T} q(e) \,, \tag{10}$$

where e ranges over all the 4-tuples $\langle A \rightarrow BC, i, k, j \rangle$ in the parse tree T. This objective function can be efficiently solved by dynamic programming such as the CYK algorithm.

Although the time complexity of the inside-outside algorithm with GM-LVeGs is polynomial in the sentence length and the nonterminal number, in practice the algorithm is still slow because the number of Gaussian components in the inside and outside scores increases dramatically with the recursion depth. To speed up the computation, we prune Gaussian components in the inside and outside scores using the following technique. Suppose we have a minimum pruning threshold k_{min}

$$s_{\mathbf{I}}^{A}(\mathbf{a},i,j) = \sum_{A\rightarrow BC\in R}\ \sum_{k=i,\cdots,j-1} \iint W_{A\rightarrow BC}(\mathbf{a},\mathbf{b},\mathbf{c}) \times s_{\mathbf{I}}^{B}(\mathbf{b},i,k) \times s_{\mathbf{I}}^{C}(\mathbf{c},k+1,j)\, d\mathbf{b}d\mathbf{c}\,. \tag{6}$$

$$\begin{aligned} s_{\mathbf{O}}^{A}(\mathbf{a},i,j) = & \sum_{B\rightarrow CA\in R}\ \sum_{k=1,\cdots,i-1} \iint W_{B\rightarrow CA}(\mathbf{b},\mathbf{c},\mathbf{a}) \times s_{\mathbf{O}}^{B}(\mathbf{b},k,j) \times s_{\mathbf{I}}^{C}(\mathbf{c},k,i-1)\, d\mathbf{b}d\mathbf{c} \\ + & \sum_{B\rightarrow AC\in R}\ \sum_{k=j+1,\cdots,n} \iint W_{B\rightarrow AC}(\mathbf{b},\mathbf{a},\mathbf{c}) \times s_{\mathbf{O}}^{B}(\mathbf{b},i,k) \times s_{\mathbf{I}}^{C}(\mathbf{c},j+1,k)\, d\mathbf{b}d\mathbf{c}\,. \end{aligned} \tag{7}$$

$$s(A\rightarrow BC,i,k,j) = \iiint W_{A\rightarrow BC}(\mathbf{a},\mathbf{b},\mathbf{c}) \times s_{\mathbf{O}}^{A}(\mathbf{a},i,j) \times s_{\mathbf{I}}^{B}(\mathbf{b},i,k) \times s_{\mathbf{I}}^{C}(\mathbf{c},k+1,j)\, d\mathbf{a}d\mathbf{b}d\mathbf{c}\,. \tag{8}$$

Table 1: Equation 6: $s_{\mathbf{I}}^{A}(\mathbf{a},i,j)$ is the inside score of a nonterminal A over a span $w_{i:j}$ in the sentence $w_{1:n}$, where $1 \leq i < j \leq n$. Equation 7: $s_{\mathbf{O}}^{A}(\mathbf{a},i,j)$ is the outside score of a nonterminal A over a span $w_{i:j}$ in the sentence $w_{1:n}$, where $1 \leq i \leq j \leq n$. Equation 8: $s(A \rightarrow BC, i, k, j)$ is the score of a production rule $A \rightarrow BC$ with nonterminals A, B, and C spanning words $w_{i:j}$, $w_{i,k}$, and $w_{k+1:j}$ respectively in the sentence $w_{1:n}$, where $1 \leq i \leq k < j \leq n$.

and a maximum pruning threshold k_{max}. Given an inside or outside score with k_c Gaussian components, if $k_c \leq k_{min}$, then we do not prune any Gaussian component; otherwise, we compute $k_{allow} = \min\{k_{min} + \text{floor}(k_c^{\vartheta}), k_{max}\}$ $(0 \leq \vartheta \leq 1$ is a constant) and keep only k_{allow} components with the largest mixture weights.

In addition to component pruning, we also employ two constituent pruning techniques to reduce the search space during parsing. The first technique is used by Petrov et al. (2006). Before parsing a sentence with a GM-LVeG, we run the inside-outside algorithm with the treebank grammar and calculate the posterior probability of every nonterminal spanning every substring. Then a nonterminal would be pruned from a span if its posterior probability is below a pre-specified threshold p_{min}. When parsing with GM-LVeGs, we only consider the unpruned nonterminals for each span.

The second constituent pruning technique is similar to the one used by Socher et al. (2013). Note that for a strong constituency parser such as the Berkeley parser (Petrov and Klein, 2007), the constituents in the top 200 best parses of a sentence can cover almost all the constituents in the gold parse tree. So we first use an existing constituency parser to run k-best parsing with $k = 200$ on the input sentence. Then we parse with a GM-LVeG and only consider the constituents that appear in the top 200 parses. Note that this method is different from the re-ranking technique because it may produce a parse different from the top 200 parses.

3.3 Learning

Given a training dataset $D = \{(T_i, \mathbf{w}_i) \,|\, i = 1, \dots, m\}$ containing m samples, where T_i is the gold parse tree with unrefined nonterminals for the sentence $\mathbf{w}_i$, the objective of discriminative learning is to minimize the negative log conditional likelihood:

$$\mathcal{L}(\Theta) = -\log \prod_{i=1}^{m} P(T_i|\mathbf{w}_i;\Theta)\,, \tag{11}$$

where Θ represents the set of parameters of the GM-LVeG.

We optimize the objective function using the Adam (Kingma and Ba, 2014) optimization algorithm. The derivative with respect to Θ_r, the parameters of the weight function $W_r(\mathbf{r})$ of an unrefined production rule r, is calculated as follows (the derivation is in the supplementary material):

$$\begin{aligned} \frac{\partial\mathcal{L}(\Theta)}{\partial\Theta_r} = & \sum_{i=1}^{m} \int \left(\frac{\partial W_r(\mathbf{r})}{\partial\Theta_r} \right. \\ & \left. \times \frac{\mathbb{E}_{P(t|\mathbf{w}_i)}[f_r(t)] - \mathbb{E}_{P(t|T_i)}[f_r(t)]}{W_r(\mathbf{r})} \right) d\mathbf{r}\,, \end{aligned} \tag{12}$$

where t indicates a parse tree with nonterminal subtypes, and $f_r(t)$ is the number of occurrences of the unrefined rule r in the unrefined parse tree that is obtained by replacing all the subtypes in t with the corresponding nonterminals. The two expectations in Equation 12 can be efficiently computed using the inside-outside algorithm. Because the second expectation is conditioned on the parse tree T_i, in Equation 6 and Equation 7 we can skip all the summations and assign the values of B, C, and k according to T_i.

In GM-LVeGs, Θ_r is the set of parameters in a Gaussian mixture:

$$\Theta_r = \{(\rho_{r,k}, \boldsymbol{\mu}_{r,k}, \boldsymbol{\Sigma}_{r,k}) | k = 1, \ldots, K_r\} \,. \quad (13)$$

According to Equation 12, we need to take the derivatives of $W_r(\mathbf{r})$ respect to $\rho_{r,k}$, $\boldsymbol{\mu}_{r,k}$, and $\boldsymbol{\Sigma}_{r,k}$ respectively:

$$\partial W_r(\mathbf{r}) / \partial \rho_{r,k} = \mathcal{N}_{r,k}(\mathbf{r}) \,, \quad (14)$$

$$\partial W_r(\mathbf{r}) / \partial \boldsymbol{\mu}_{r,k} = \rho_{r,k} \mathcal{N}_{r,k}(\mathbf{r}) \boldsymbol{\Sigma}_{r,k}^{-1} (\mathbf{r} - \boldsymbol{\mu}_{r,k}) \,, (15)$$

$$\partial W_r(\mathbf{r}) / \partial \boldsymbol{\Sigma}_{r,k} = \rho_{r,k} \mathcal{N}_{r,k}(\mathbf{r}) \boldsymbol{\Sigma}_{r,k}^{-1} \frac{1}{2} \Big(- I + (\mathbf{r} - \boldsymbol{\mu}_{r,k})(\mathbf{r} - \boldsymbol{\mu}_{r,k})^T \boldsymbol{\Sigma}_{r,k}^{-1} \Big) \,. \quad (16)$$

Substituting Equation 14–16 into Equation 12, we have the full gradient formulations of all the parameters. In spite of the integral in Equation 12, we can derive a closed-form solution for the gradient of each parameter, which is shown in the supplementary material.

In order to keep each mixture weight $\rho_{r,k}$ positive, we do not directly optimize $\rho_{r,k}$; instead, we set $\rho_{r,k} = \exp(\theta_{\rho_{r,k}})$ and optimize $\theta_{\rho_{r,k}}$ by gradient descent. We use a similar trick to keep each covariance matrix $\boldsymbol{\Sigma}_{r,k}$ positive definite.

Since we use the inside-outside algorithm described in Section 3.2 to calculate the two expectations in Equation 12, we face the same efficiency problem that we encounter in parsing. To speed up the computation,we again use both component pruning and constituent pruning introduced in Section 3.2.

Because gradient descent is often sensitive to the initial values of the parameters, we employ the following informed initialization method. Mixture weights are initialized using the treebank grammar. Suppose in the treebank grammar $P(r)$ is the probability of a production rule r. We initialize the mixture weights in the weight function W_r by $\rho_{r,k} = \alpha \cdot P(r)$ where $\alpha > 1$ is a constant. We initialize all the covariance matrices to identity matrices and initialize each mean with a value uniformly sampled from $[-0.05, 0.05]$.

4 Experiment

We evaluate the GM-LVeG on part-of-speech (POS) tagging and constituency parsing and compare it against its special cases such as LVGs and CVGs. It shall be noted that in this paper we focus on the basic framework of LVeGs and aim to show its potential advantage over previous special cases. It is therefore not our goal to compete with the latest state-of-the-art approaches to tagging and parsing. In particular, we currently do not incorporate contextual information of words and constituents during tagging and parsing, while such information is critical in achieving state-of-the-art accuracy. We will discuss future improvements of LVeGs in Section 5.

4.1 Datasets

Parsing. We use the Wall Street Journal corpus from the Penn English Treebank (WSJ) (Marcus et al., 1994). Following the standard data splitting, we use sections 2 to 21 for training, section 23 for testing, and section 22 for development. We preprocess the treebank using a right-branching binarization procedure to obtain an unannotated X-bar grammar, so that there are only binary and unary production rules. To deal with the problem of unknown words in testing, we adopt the unknown word features used in the Berkeley parser and set the unknown word threshold to 1. Specifically, any word occurring less than two times is replaced by one of the 60 unknown word categories.

Tagging. (1) We use Wall Street Journal corpus from the Penn English Treebank (WSJ) (Marcus et al., 1994). Following the standard data splitting, we use sections 0 to 18 for training, sections 22 to 24 for testing, and sections 19 to 21 for development. (2) The Universal Dependencies treebank 1.4 (UD) (Nivre et al., 2016), in which English, French, German, Russian, Spanish, Indonesian, Finnish, and Italian treebanks are used. We use the original data splitting of these corpora for training and testing. For both WSJ and UD English treebanks, we deal with unknown words in the same way as we do in parsing. For the rest of the data, we use only one unknown word category and the unknown word threshold is also set to 1.

4.2 POS Tagging

POS tagging is the task of labeling each word in a sentence with the most probable part-of-speech tag. Here we focus on POS tagging with Hidden Markov Models (HMMs). Because HMMs are equivalent to probabilistic regular grammars, we can extend HMMs with both LVGs and LVeGs. Specifically, the hidden states in HMMs can be seen as nonterminals in regular grammars and therefore can be associated with latent variables or latent vectors.

We implement two training methods for LVGs. The first (LVG-G) is generative training using expectation-maximization that maximizes the joint probability of the sentence and the tags. The second (LVG-D) is discriminative training using gradient descent that maximizes the conditional probability of the tags given the sentence. In both cases, each nonterminal is split into a fixed number of subtypes. In our experiments we test 1, 2, 4, 8, and 16 subtypes of each nonterminal. Due to the limited space, we only report experimental results of LVG with 16 subtypes for each nonterminal. Full experimental results can be found in the supplementary material.

We experiment with two different GM-LVeGs: GM-LVeG-D with diagonal Gaussians and GM-LVeG-S with spherical Gaussians. In both cases, we fix the number of Gaussian components K_r to 4 and the dimension of the latent vectors d to 3. We do not use any pruning techniques in learning and inference because we find that our algorithm is fast enough with the current setting of K_r and d. We train the GM-LVeGs for 20 epoches and select the models with the best token accuracy on the development data for the final testing.

We report both token accuracy and sentence accuracy of POS tagging in Table 2. It can be seen that, on all the testing data, GM-LVeGs consistently surpass LVGs in terms of both token accuracy and sentence accuracy. GM-LVeG-D is slightly better than GM-LVeG-S in sentence accuracy, producing the best sentence accuracy on 5 of the 9 testing datasets. GM-LVeG-S performs slightly better than GM-LVeG-D in token accuracy on 5 of the 9 datasets. Overall, there is not significant difference between GM-LVeG-D and GM-LVeG-S. However, GM-LVeG-S admits more efficient learning than GM-LVeG-D in practice since it has fewer parameters.

4.3 Parsing

For efficiency, we train GM-LVeGs only on sentences with no more than 50 words (totally 39115 sentences). Since we have found that spherical Gaussians are better than diagonal Gaussians considering both model performance and learning efficiency, here we use spherical Gaussians in the weight functions. The dimension of latent vectors d is set to 3, and all the Gaussian mixtures have $K_r = 4$ components. We use $\alpha = 8$ in initializing mixture weights. We train the GM-LVeG for 15 epoches and select the model with the highest F1 score on the development data for the final testing. We use component pruning in both learning and parsing, with $k_{max} = 50$ and $\vartheta = 0.35$ in both learning and parsing, $k_{min} = 40$ in learning and $k_{min} = 20$ in parsing. During learning we use the first constituent pruning technique with the pruning threshold $p_{min} = 1e - 5$, and during parsing we use the second constituent pruning technique based on the Berkeley parser which produced 133 parses on average for each testing sentence. As can be seen, we use weaker pruning during training than during testing. This is because in training stronger pruning (even if accurate) results in worse estimation of the first expectation in Equation 12, which makes gradient computation less accurate.

We compare LVeGs with CVGs and several variants of LVGs: (1) LVG-G-16 and LVG-D-16, which are LVGs with 16 subtypes for each nonterminal with discriminative and generative training respectively (accuracies obtained from Petrov and Klein (2008a)); (2) Multi-scale grammars (Petrov and Klein, 2008b), trained without using the span features in order for a fair comparison; (3) Berkeley parser (Petrov and Klein, 2007) (accuracies obtained from Petrov and Klein (2008b) because Petrov and Klein (2007) do not report exact match scores). The experimental results are shown in Table 3. It can be seen that GM-LVeG-S produces the best F1 scores on both the development data and the testing data. It surpasses the Berkeley parser by 0.92% in F1 score on the testing data. Its exact match score on the testing data is only slightly lower than that of LVG-D-16.

We further investigate the influence of the latent vector dimension and the Gaussian component number on the efficiency and the parsing accuracy . We experiment on a small dataset (statistics of this dataset are in the supplemental material). We first fix the component number to 4 and experiment with the dimension 2, 3, 4, 5, 6, 7, 8, 9. Then we fix the dimension to 3 and experiment with the component number 2, 3, 4, 5, 6, 7, 8, 9. F1 scores on the development data are shown in the first row in Figure 1. Average time consumed per epoch in learning is shown in the second row in Figure 1. When $K_r = 4$, the best dimension is 5; when $d = 3$, the best Gaussian component number is 3. A higher dimension or a larger Gaussian component number hurts the model performance and requires much more time for learning. Thus

Model	WSJ		English		French		German		Russian		Spanish		Indonesian		Finnish		Italian	
	T	S	T	S	T	S	T	S	T	S	T	S	T	S	T	S	T	S
LVG-D-16	96.62	48.74	92.31	52.67	93.75	34.90	87.38	20.98	81.91	12.25	92.47	24.82	89.27	20.29	83.81	19.29	94.81	45.19
LVG-G-16	96.78	50.88	93.30	57.54	94.52	34.90	88.92	24.05	84.03	16.63	93.21	27.37	90.09	21.19	85.01	20.53	95.46	48.26
GM-LVeG-D	96.99	53.10	**93.66**	**59.46**	94.73	**39.60**	89.11	24.77	**84.21**	17.84	**93.76**	**32.48**	**90.24**	**21.72**	85.27	**23.30**	95.61	**50.72**
GM-LVeG-S	**97.00**	**53.11**	93.55	58.11	**94.74**	39.26	**89.14**	**25.58**	84.06	**18.44**	93.52	30.66	90.12	**21.72**	**85.35**	22.07	**95.62**	49.69

Table 2: Token accuracy (T) and sentence accuracy (S) for POS tagging on the testing data.

Model	dev (all)	test $\leq$ 40		test (all)	
	F1	F1	EX	F1	EX
LVG-G-16				88.70	35.80
LVG-D-16				89.30	**39.40**
Multi-Scale		89.70	39.60	89.20	37.20
Berkeley Parser		90.60	39.10	90.10	37.10
CVG (SU-RNN)	91.20	91.10		90.40	
GM-LVeG-S	**91.24**	**91.38**	**41.51**	**91.02**	39.24

Table 3: Parsing accuracy on the testing data of WSJ. EX indicates the exact match score.

our choice of $K_r = 4$ and $d = 3$ in GM-LVeGs for parsing is a good balance between the efficiency and the parsing accuracy.

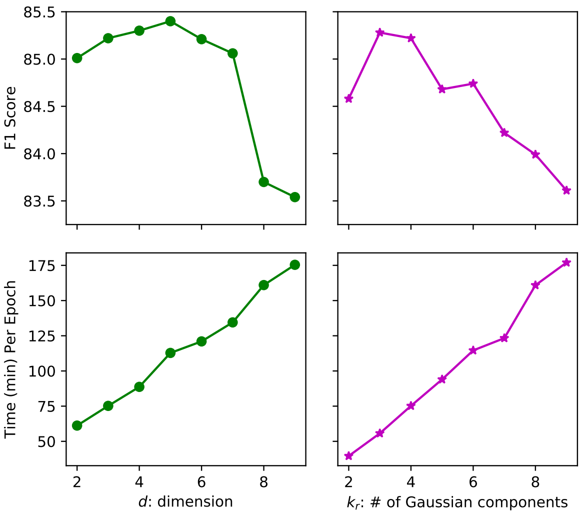

Figure 1: F1 score and average time (min) consumed per epoch in learning. **Left**: # of Gaussian components fixed to 4 with different dimensions; **Right**: dimension of Gaussians fixed to 3 with different # of Gaussian components.

5 Discussion

It shall be noted that in this paper we choose to focus on the basic framework and algorithms of LVeGs, and therefore we leave a few important extensions for future work. One extension is to incorporate contextual information of words and constituents. which is a crucial technique that can be found in most state-of-the-art approaches to parsing or POS tagging. One possible way to utilize contextual information in LVeGs is to allow the words in the context of an anchored production rule to influence the rule's weight function. For example, we may learn neural networks to predict the parameters of the Gaussian mixture weight functions in a GM-LVeG from the pre-trained embeddings of the words in the context.

In GM-LVeGs, we currently use the same number of Gaussian components for all the weight functions. A more desirable way would be automatically determining the number of Gaussian components for each production rule based on the ideal refinement granularity of the rule, e.g., we may need more Gaussian components for NP → DT NN than for NP → DT JJ, since the latter is rarely used. There are a few possible ways to learn the component numbers such as greedy addition and removal, the split-merge method, and sparsity priors over mixture weights.

An interesting extension beyond LVeGs is to have a single continuous space for subtypes of all the nonterminals. Ideally, subtypes of the same nonterminal or similar nonterminals are close to each other. The benefit is that similarity between nonterminals can now be modeled.

6 Conclusion

We present Latent Vector Grammars (LVeGs) that associate each nonterminal with a latent continuous vector space representing the set of subtypes of the nonterminal. For each production rule, a LVeG defines a continuous weight function over the subtypes of the nonterminals involved in the rule. We show that LVeGs can subsume latent variable grammars and compositional vector grammars as special cases. We then propose Gaussian mixture LVeGs (GM-LVeGs). which formulate weight functions of production rules by mixtures of Gaussian distributions. The partition function and the expectations of fine-grained production rules in GM-LVeGs can be efficiently computed using dynamic programming, which makes learning and inference with GM-LVeGs feasible.

We empirically show that GM-LVeGs can achieve competitive accuracies on POS tagging and constituency parsing.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61503248), Major Program of Science and Technology Commission Shanghai Municipal (17JC1404102), and Program of Shanghai Subject Chief Scientist (A type) (No.15XD1502900). We would like to thank the anonymous reviewers for their careful reading and useful comments.

References

Eugene Charniak. 1996. Tree-bank grammars. In *Proceedings of the 30th National Conference on Artificial Intelligence*, volume 2, pages 1031–1036.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 132–139. Association for Computational Linguistics.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637.

Greg Durrett and Dan Klein. 2015. Neural CRF parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 302–312. Association for Computational Linguistics.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209. Association for Computational Linguistics.

Mark Johnson. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Dan Klein and Christopher D Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st annual meeting on Association for Computational Linguistics*, pages 423–430. Association for Computational Linguistics.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The penn treebank: annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology*, pages 114–119. Association for Computational Linguistics.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic CFG with latent annotations. In *Proceedings of the 43rd annual meeting on Association for Computational Linguistics*, pages 75–82. Association for Computational Linguistics.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan T. McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th International Conference on Language Resources and Evaluation*, pages 1659–1666.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 44th annual meeting of the Association for Computational Linguistics*, pages 433–440. Association for Computational Linguistics.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proceedings of the 2007 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 404–411. Association for Computational Linguistics.

Slav Petrov and Dan Klein. 2008a. Discriminative log-linear grammars with latent variables. In *Advances in Neural Information Processing Systems 20*, pages 1153–1160.

Slav Petrov and Dan Klein. 2008b. Sparse multi-scale grammars for discriminative latent variable parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 867–876. Association for Computational Linguistics.

Noah A Smith and Mark Johnson. 2007. Weighted and probabilistic context-free grammars are equally expressive. *Computational Linguistics*, 33(4):477–491.

Richard Socher, John Bauer, Christopher D Manning, et al. 2013. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 455–465. Association for Computational Linguistics.

Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. 2011. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on Machine Learning*, pages 129–136.

Supplementary Material

Abstract

This supplementary material contains the following contents. (1) The derivation of the gradient formulation (Equation 12 in the paper). (2) The general idea of calculating analytic gradients for all the parameters in Gaussian Mixture Latent Vector Grammars (GM-LVeGs). (3) Algorithmic complexity and running time. (4) Statistics of the data used for part-of-speech (POS) tagging and constituency parsing. (5) Additional experimental results and analysis of GM-LVeGs.

A Derivation of Gradient Formulations

Given a training dataset $D = \{(T_i, \mathbf{w}_i) \,|\, i = 1, \ldots, m\}$ containing m samples, we minimize the negative log conditional likelihood during learning of GM-LVeGs:

$$\mathcal{L}(\Theta) = -\log \prod_{i=1}^{m} P(T_i | \mathbf{w}_i; \Theta)\,, \tag{17}$$

where T_i is the gold parse tree with unrefined nonterminals for the sentence $\mathbf{w}_i$, and Θ is the set of parameters in GM-LVeGs.

We define t as a parse tree with nonterminal subtypes, denote by $f_r(t)$ the number of occurrences of the unrefined rule r in the unrefined parse tree that is obtained by replacing all the subtypes in t with the corresponding nonterminals, and use $\mathbf{r}$ to represent a fine-grained production rule of r, which is represented by the concatenation of the latent vectors of the nonterminals in r.

The weight of t is defined as:

$$s_t = \prod_{\mathbf{r} \in t} W_r(\mathbf{r})\,. \tag{18}$$

The weight of T, a parse tree with unrefined nonterminals, is defined as:

$$s_T = \int_{t \sim T} s_t \, dt\,, \tag{19}$$

where $t \sim T$ indicates that t is a parse tree with nonterminal subtypes that can be converted into a parse tree T by replacing its nonterminal subtypes with the corresponding nonterminals. The weight of a sentence $\mathbf{w}$ is defined as:

$$s_{\mathbf{w}} = \int_{t \sim \mathbf{w}} s_t \, dt\,, \tag{20}$$

where $t \sim \mathbf{w}$ indicates that t is a parse tree of $\mathbf{w}$ with nonterminal subtypes. Thus the conditional probability density of t given T is

$$P(t|T) = \frac{s_t}{s_T}\,, \tag{21}$$

the conditional probability density of t given $\mathbf{w}$ is

$$P(t|\mathbf{w}) = \frac{s_t}{s_{\mathbf{w}}}\,, \tag{22}$$

and the conditional probability of T given $\mathbf{w}$ is

$$P(T|\mathbf{w}) = \frac{s_T}{s_{\mathbf{w}}} = \int_{t \sim T} P(t|\mathbf{w}) \, dt\,. \tag{23}$$

Therefore, we rewrite Equation 17 as

$$\mathcal{L}(\Theta) = -\sum_{i=1}^{m} \log \int_{t \sim T_i} P(t|\mathbf{w}_i) \, dt\,. \tag{24}$$

The derivative of $\mathcal{L}(\Theta)$ with respect to Θ_r, where r is an unrefined production rule, is calculated by Equation 25 in Table 4.

B Calculation of Analytic Gradients

In GM-LVeGs, Θ_r is the set of parameters in a Gaussian mixture with K_r mixture components:

$$\Theta_r = \{(\rho_{r,k}, \boldsymbol{\mu}_{r,k}, \boldsymbol{\Sigma}_{r,k}) | k = 1, \ldots, K_r\}\,. \tag{29}$$

According to Equation 25 in Table 4, we need to take derivatives of $W_r(\mathbf{r})$ with respect to $\rho_{r,k}$, $\boldsymbol{\mu}_{r,k}$, and $\boldsymbol{\Sigma}_{r,k}$ respectively:

$$\partial W_r(\mathbf{r}) / \partial \rho_{r,k} = \mathcal{N}_{r,k}(\mathbf{r})\,, \tag{30}$$

$$\partial W_r(\mathbf{r}) / \partial \boldsymbol{\mu}_{r,k} = \rho_{r,k} \mathcal{N}_{r,k}(\mathbf{r}) \boldsymbol{\Sigma}_{r,k}^{-1} (\mathbf{r} - \boldsymbol{\mu}_{r,k})\,, \tag{31}$$

$$\begin{aligned}\partial W_r(\mathbf{r}) / \partial \boldsymbol{\Sigma}_{r,k} = \rho_{r,k} \mathcal{N}_{r,k}(\mathbf{r}) \boldsymbol{\Sigma}_{r,k}^{-1} \frac{1}{2} \Big(& -I \\ & + (\mathbf{r} - \boldsymbol{\mu}_{r,k})(\mathbf{r} - \boldsymbol{\mu}_{r,k})^T \boldsymbol{\Sigma}_{r,k}^{-1} \Big)\,.\end{aligned} \tag{32}$$

For brevity, we define

$$\psi(\mathbf{r}) = \frac{\mathbb{E}_{P(t|\mathbf{w}_i)}[f_r(t)] - \mathbb{E}_{P(t|T_i)}[f_r(t)]}{W_r(\mathbf{r})}\,. \tag{33}$$

Substituting Equations 30–32 into Equation 25, we have the full gradient formulations of all the parameters (Equations 26–28 in Table 4).

In the following discussion, we assume that all the Gaussians are diagonal. It can be verified that $\psi(\mathbf{r})$ in Equation 33 is in fact a mixture of Gaussians, so multiplying $\psi(\mathbf{r})$ by $\mathcal{N}_{r,k}(\mathbf{r})$ in Equation 26–28 results in another mixture of Gaussians. Below we consider the special case where

$$
\begin{aligned}
\mathcal{L}'(\Theta) &= -\sum_{i=1}^{m}\int_{t\sim T_i}\frac{(P(t|\mathbf{w}_i))'}{\int_{t'\sim T_i}P(t'|\mathbf{w}_i)\,dt'}\,dt \\
&= -\sum_{i=1}^{m}\int_{t\sim T_i}\frac{(P(t|\mathbf{w}_i))'}{\int_{t'\sim T_i}P(t'|\mathbf{w}_i)\,dt'}\times\frac{P(t|\mathbf{w}_i)}{P(t|\mathbf{w}_i)}\,dt \\
&= -\sum_{i=1}^{m}\int_{t\sim T_i}\frac{P(t|\mathbf{w}_i)}{\int_{t'\sim T_i}P(t'|\mathbf{w}_i)\,dt'}\times(\log P(t|\mathbf{w}_i))'\,dt \\
&= -\sum_{i=1}^{m}\int_{t\sim T_i}P(t|T_i)\times\left(\log\frac{\prod_{\mathbf{r}\in t}W_r(\mathbf{r})}{\int_{t'\sim\mathbf{w}_i}\prod_{\mathbf{r}\in t'}W_r(\mathbf{r})\,dt'}\right)'\,dt \\
&= -\sum_{i=1}^{m}\int_{t\sim T_i}P(t|T_i)\times\left(\log\prod_{\mathbf{r}\in t}W_r(\mathbf{r})-\log\int_{t'\sim\mathbf{w}_i}\prod_{\mathbf{r}\in t'}W_r(\mathbf{r})\,dt'\right)'\,dt \\
&= -\sum_{i=1}^{m}\left(\int_{t\sim T_i}P(t|T_i)\left(\sum_{\mathbf{r}\in t}\log W_r(\mathbf{r})\right)'\,dt-\left(\log\int_{t'\sim\mathbf{w}_i}\prod_{\mathbf{r}\in t'}W_r(\mathbf{r})\,dt'\right)'\right) \\
&= -\sum_{i=1}^{m}\left(\int_{t\sim T_i}P(t|T_i)\sum_{\mathbf{r}\in t}\frac{W_r'(\mathbf{r})}{W_r(\mathbf{r})}\,dt-\int_{t'\sim\mathbf{w}_i}P(t'|\mathbf{w}_i)\sum_{\mathbf{r}\in t'}\frac{W_r'(\mathbf{r})}{W_r(\mathbf{r})}\,dt'\right) \\
&= -\sum_{i=1}^{m}\int_{\mathbf{r}}\frac{\mathbb{E}_{P(t|T_i)}[f_r(t)]-\mathbb{E}_{P(t|\mathbf{w}_i)}[f_r(t)]}{W_r(\mathbf{r})}\times W_r'(\mathbf{r})\,d\mathbf{r}\,. \qquad (25)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial\mathcal{L}(\Theta)}{\partial\rho_{r,k}} &= \sum_{i=1}^{m}\int_{\mathbf{r}}\psi(\mathbf{r})\cdot\mathcal{N}_{r,k}(\mathbf{r})\,d\mathbf{r}\,. \qquad (26)\\
\frac{\partial\mathcal{L}(\Theta)}{\partial\boldsymbol{\mu}_{r,k}} &= \sum_{i=1}^{m}\int_{\mathbf{r}}\psi(\mathbf{r})\cdot\mathcal{N}_{r,k}(\mathbf{r})\cdot\rho_{r,k}\boldsymbol{\Sigma}_{r,k}^{-1}(\mathbf{r}-\boldsymbol{\mu}_{r,k})\,d\mathbf{r}\,. \qquad (27)\\
\frac{\partial\mathcal{L}(\Theta)}{\partial\boldsymbol{\Sigma}_{r,k}} &= \sum_{i=1}^{m}\int_{\mathbf{r}}\psi(\mathbf{r})\cdot\mathcal{N}_{r,k}(\mathbf{r})\cdot\rho_{r,k}\boldsymbol{\Sigma}_{r,k}^{-1}\frac{(\mathbf{r}-\boldsymbol{\mu}_{r,k})(\mathbf{r}-\boldsymbol{\mu}_{r,k})^T\boldsymbol{\Sigma}_{r,k}^{-1}-I}{2}\,d\mathbf{r}\,. \qquad (28)
\end{aligned}
$$

Table 4: Derivation of gradient formulations.

the resulting Gaussian mixture contains only a single component:

$$\psi(\mathbf{r})\cdot\mathcal{N}_{r,k}(\mathbf{r})=\lambda\cdot\mathcal{N}(\mathbf{r})\,. \qquad (34)$$

Owing to the sum rule in integral, we can easily extend our derivation on the special case to the general case in which the Gaussian mixture contains multiple components. Because $\mathcal{N}(\mathbf{r})$ is diagonal, it can be factorized as:

$$\mathcal{N}(\mathbf{r})=\mathcal{N}(\mathbf{r}^1)\times\cdots\times\mathcal{N}(\mathbf{r}^{|\mathbf{r}|})\,, \qquad (35)$$

where $\mathcal{N}(\mathbf{r}^1),\ldots,\mathcal{N}(\mathbf{r}^{|\mathbf{r}|})$ are univariate Gaussians (or normal distributions), $\mathbf{r}^d$ $(1\leq d\leq|\mathbf{r}|)$ refers to the d-th element of $\mathbf{r}$, and $|\mathbf{r}|$ is the dimension of $\mathbf{r}$. The integral in Equation 26 can be readily calculated as

$$\lambda\cdot\int\mathcal{N}(\mathbf{r})\,d\mathbf{r}=\lambda\,. \qquad (36)$$

For Equation 27, consider taking the derivative with respect to the mean in dimension d. Since the means in different dimensions are independent, to solve the integral in Equation 27, we only need to solve the following integral:

$$
\begin{aligned}
&\lambda\cdot\int\mathcal{N}(\mathbf{r})\mathbf{r}^d\,d\mathbf{r}^1\ldots\,d\mathbf{r}^{|\mathbf{r}|}\\
&=\lambda\cdot\int\mathcal{N}(\mathbf{r}^1)\,d\mathbf{r}^1\times\cdots\times\int\mathcal{N}(\mathbf{r}^d)\mathbf{r}^d\,d\mathbf{r}^d\\
&\times\ldots\times\int\mathcal{N}(\mathbf{r}^{|\mathbf{r}|})\,d\mathbf{r}^{|\mathbf{r}|}\\
&=\lambda\cdot\int\mathcal{N}(\mathbf{r}^d)\mathbf{r}^d\,d\mathbf{r}^d\,. \qquad (37)
\end{aligned}
$$

For Equation 28, when taking the derivative with respect to the variance in dimension d, we also need to solve Equation 37 and additionally need to solve the following integral:

$$\begin{aligned}
&\lambda \cdot \int \mathcal{N}(\mathbf{r})\mathbf{r}^d\mathbf{r}^d \, d\mathbf{r}^1 \ldots d\mathbf{r}^{|\mathbf{r}|} \\
&= \lambda \cdot \int \mathcal{N}(\mathbf{r}^1)\, d\mathbf{r}^1 \times \cdots \times \int \mathcal{N}(\mathbf{r}^d)\mathbf{r}^d\mathbf{r}^d \, d\mathbf{r}^d \\
&\times \ldots \times \int \mathcal{N}(\mathbf{r}^{|\mathbf{r}|})\, d\mathbf{r}^{|\mathbf{r}|} \\
&= \lambda \cdot \int \mathcal{N}(\mathbf{r}^d)\mathbf{r}^d\mathbf{r}^d \, d\mathbf{r}^d \,. \qquad (38)
\end{aligned}$$

The integrals in Equation 37 and Equation 38 are the first order moment and the second order moment of the univariate Gaussian $\mathcal{N}(\mathbf{r}^d)$ respectively, and both of them can be calculated exactly. Therefore, we can calculate analytic gradients of all the parameters in GM-LVeGs.

C Algorithmic Complexity and Running Time

The time complexity of the learning algorithm for each sentence in each epoch is approximately $\mathcal{O}(cn^3kmd + cklmd^2)$. The first term is the time complexity of the extended inside-outside algorithm, where c is the number of binary productions in CNF, n is the length of the sentence, k is the Gaussian component number of each rule weight function, m is the maximum Gaussian component number of an inside or outside score after pruning, and d is the dimension of diagonal Gaussians. kmd is generally much smaller than cn^3. It shall be noted that m is bounded by k_{max}, which is set to 50 in our experiments. The second term is the approximate time complexity of gradient calculation, where l is the number of times that a production rule is used in all possible parses of a sentence. The second term is much smaller than the first term in general.

We run our learning and inference algorithms with a CPU cluster without any GPU. In our POS tagging experiments, GM-LVeGs are only slightly slower than the baseline LVG models, and we can perform all the tagging experiments on all the datasets with our model within one day. For parsing, there is a trade-off between running time and parsing accuracy based on the amount of pruning. For the best parsing accuracy of GM-LVeGs, it takes two weeks for training. However, once we complete training, parsing can be done within three minutes on the whole testing data of WSJ.

There are a few ways to improve the training efficiency. We currently use CPU parallelization at the sentence level in training, but in the future we may take the advantage of GPU parallelization, e.g., we can vectorize the inside-outside algorithm for a batch of sentences of the same length. Besides, for each long sentence, we can parallelize the inside or outside computation at the same recursive depth.

D Data Statistics

Dataset	# of tokens	# of sentences		
		train	test	dev
WSJ	950028	39832	2416	1700

Table 5: Statistics of WSJ used for constituency parsing.

Statistics of the data used for constituency parsing are shown in Table 5. Statistics of the data used for POS tagging are summarized in Table 6. In the experiments of constituency parsing, in order to study the influence of the dimension of Gaussians and the number of Gaussian components on the parsing accuracy, we experimented on a small dataset. The small dataset only contains section 4 and section 5 of WSJ. In the two sections, we use file IDs from 80-89 in the two sections are used for testing, 90-99 for development, and the rest are used for training. The resulting dataset contains 3599 training samples, 426 test samples, and 375 development samples.

E Additional Experimental Results

The complete experimental results of POS tagging are shown in Table 7. In addition to the results shown in the paper, this table includes the tagging results of LVGs with 1, 2, 4, 8 subtypes for each nonterminal.

In the experiments of constituency parsing, in order to investigate the impact of the maximum Gaussian component number of inside and outside scores, we experiment with a new pruning technique. Specifically, we use a maximum component-pruning threshold k_{hard}. We do not prune any Gaussian component if an inside or outside score has no more than k_{hard} Gaussian components; otherwise we keep only k_{hard} Gaussian components with the largest mixture weights. We experiment on the small dataset mentioned in Section D. For efficiency, we train GM-LVeG-D only on sentences of no more than 20 words

	WSJ	English	French	German	Russian	Spanish	Indonesian	Finnish	Italian
# of tokens	1173766	254830	402197	298242	99389	431587	121923	181022	292471
train	38219	12543	14554	14118	4029	14187	4477	12217	12837
dev	5462	2002	1596	799	502	1552	559	716	489
test	5527	2077	298	977	499	274	557	648	489

Table 6: Statistics of WSJ and UD (English, French, German, Russian, Spanish, Indonesian, Finnish, and Italian treebanks). Numbers in the rows of train, test, and dev indicate the number of sentences in training, testing, and development data respectively.

Model	WSJ		English		French		German		Russian		Spanish		Indonesian		Finnish		Italian	
	T	S	T	S	T	S	T	S	T	S	T	S	T	S	T	S	T	S
LVG-D-1	96.50	48.04	91.80	50.79	93.55	30.20	86.52	16.99	81.21	9.24	91.79	22.63	89.08	18.85	83.15	16.82	94.00	37.42
LVG-D-2	96.57	47.60	92.17	52.05	93.86	33.56	86.93	18.32	81.46	10.04	92.10	24.82	89.16	19.21	83.34	18.52	94.45	40.90
LVG-D-4	96.57	48.76	92.30	52.34	93.96	34.90	87.18	19.86	81.95	11.85	92.37	24.82	89.28	19.57	83.76	18.83	94.60	42.54
LVG-D-8	96.60	49.14	92.31	53.06	93.78	34.90	87.52	21.60	81.54	11.25	92.26	23.72	89.23	19.39	83.68	18.67	94.70	42.95
LVG-D-16	96.62	48.74	92.31	52.67	93.75	34.90	87.38	20.98	81.91	12.25	92.47	24.82	89.27	20.29	83.81	19.29	94.81	45.19
LVG-G-1	96.11	43.68	90.84	44.92	92.69	26.51	86.71	17.40	81.22	10.22	91.85	22.63	88.93	18.31	82.94	16.36	93.64	33.74
LVG-G-2	96.27	45.57	92.11	51.37	93.28	28.19	87.87	19.86	81.51	11.45	92.29	23.36	89.19	18.49	83.29	17.44	94.20	38.45
LVG-G-4	96.50	48.19	92.90	54.31	94.06	32.55	88.31	20.78	82.64	11.85	92.58	24.45	89.58	19.03	83.76	19.44	95.00	45.40
LVG-G-8	96.76	50.38	93.29	56.67	94.57	37.25	88.75	21.70	82.85	14.86	92.95	29.20	89.78	20.29	84.69	21.76	95.42	46.83
LVG-G-16	96.78	50.88	93.30	57.54	94.52	34.90	88.92	24.05	84.03	16.63	93.21	27.37	90.09	21.19	85.01	20.53	95.46	48.26
GM-LVeG-D	96.99	53.10	**93.66**	**59.46**	94.73	**39.60**	89.11	24.77	**84.21**	17.84	**93.76**	**32.48**	**90.24**	**21.72**	85.27	**23.30**	95.61	**50.72**
GM-LVeG-S	**97.00**	**53.11**	93.55	58.11	**94.74**	39.26	**89.14**	**25.58**	84.06	**18.44**	93.52	30.66	90.12	**21.72**	**85.35**	22.07	**95.62**	49.69

Table 7: Token accuracy (T) and sentence accuracy (S) for POS tagging on the testing data. The numerical postfix of each LVG model indicates the number of nonterminal subtypes, and hence LVG-G-1 denotes HMM.

and test GM-LVeG-D only on testing sentences of no more than 25 words. We experiment with $k_{hard} = 10, 20, 30, 40, 50, 60, 70, 80$. The results are shown in Figure 2. We also experiment without component pruning, which corresponds to the rightmost point in Figure 2.

We can see that a weaker component pruning or a larger k_{hard} results in a better F1 score. However, it takes much more time per epoch for learning, as is shown in the lower figure in Figure 2. We find that $k_{hard} = 40$ produces a good F1 score and also admits efficient learning. Therefore, in the experiments of constituency parsing, we use $k_{min} = 40$ in learning for the component-pruning technique introduced in Section 3.2 in the paper.

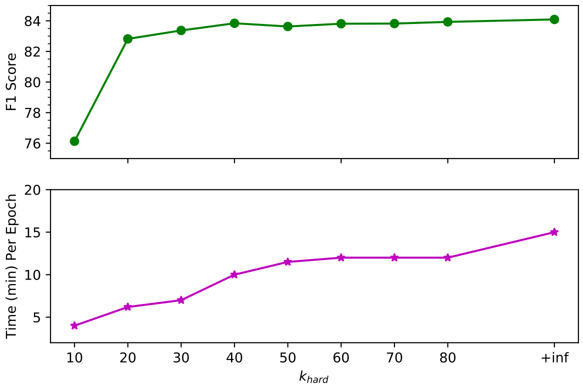

Figure 2: **Upper**: F1 scores with different k_{hard}; **Lower**: time (min) per epoch in learning with different k_{hard}.