Dynamic Programming in Rank Space: Scaling Structured Inference with Low-Rank HMMs and PCFGs

Anonymous ACL submission

Abstract

Hidden Markov Models (HMMs) and Probabilistic Context-Free Grammars (PCFGs) are widely used structured models, both of which can be represented as factor graph grammars (FGGs), a powerful formalism capable of describing a wide range of models. Recent research found it beneficial to use large state spaces for HMMs and PCFGs. However, inference with large state spaces is computationally demanding, especially for PCFGs. To tackle this challenge, we leverage tensor rank decomposition (aka. CPD) to decrease inference computational complexities for a subset of FGGs subsuming HMMs and PCFGs. We apply CPD on the factors of an FGG and then construct a new FGG defined in the rank space. Inference with the new FGG produces the same result but has a lower time complexity when the rank size is smaller than the state size. We conduct experiments on HMM language modeling and unsupervised PCFG parsing, showing better performance than previous work. We will release our code at `github.com/xxx`.

1 Introduction

Hidden Markov Models (HMMs) and Probabilistic Context-Free Grammars (PCFGs) are widely used structured models in natural language processing. They can both be represented as factor graph grammars (FGGs) (Chiang and Riley, 2020), which are a powerful tool to describe a wide range of models, allowing exact and tractable inference in most situations of interest.

Recently, researchers found it beneficial to use large state spaces for HMMs and PCFGs. However, inference with large state spaces is computationally demanding, especially for PCFGs. Chiu and Rush (2020) propose a neural VL-HMM with 2^{15} states for language modeling, narrowing down the performance gap between HMMs and LSTMs. They impose a strong sparsity constraint (i.e., each hidden state can only generate a small subset of terminal symbols) to decrease the time complexity of the forward algorithm, thus requiring pre-clustering of terminal symbols. Yang et al. (2021b) use a large state space for neural PCFG induction and achieve superior unsupervised constituency parsing performance. They use tensor rank decomposition (aka. canonical-polyadic decomposition (CPD) (Rabanser et al., 2017)) to decrease the computational complexity of the inside algorithm, but only scale the state size from tens to hundreds because the resulting complexity is still high. Chiu et al. (2021) use tensor matricization and low-rank matrix decomposition to accelerate structured inference on chain and tree structure models. However, their method has an even higher complexity than Yang et al. (2021b) on PCFGs.

In this work, we propose a new approach to scaling structured inference, which can be described by FGG notations intuitively. We first provide an intuitive and unifying perspective toward the work of Yang et al. (2021b) and Chiu et al. (2021), showing that their low-rank decomposition-based models can be viewed as decomposing large factors in an FGG—e.g., the binary rule probability tensor in PCFGs— into several smaller factors connected by new "rank" nodes. Then we target at a subset of FGGs—which we refer to as B-FGGs—subsuming all models considered by Chiu et al. (2021), whereby the inference algorithms can be formulated via B-graphs (Gallo et al., 1993; Klein and Manning, 2001). We propose a novel framework to support a family of inference algorithms in the rank space for B-FGGs. Within the framework, we apply CPD on the factors of a B-FGG and then construct a new B-FGG defined in the rank space by marginalizing all the state nodes. Inference with the new B-FGG has the same result and a lower time complexity if the rank size is smaller than the state size.

We conduct experiments in unsupervised PCFG

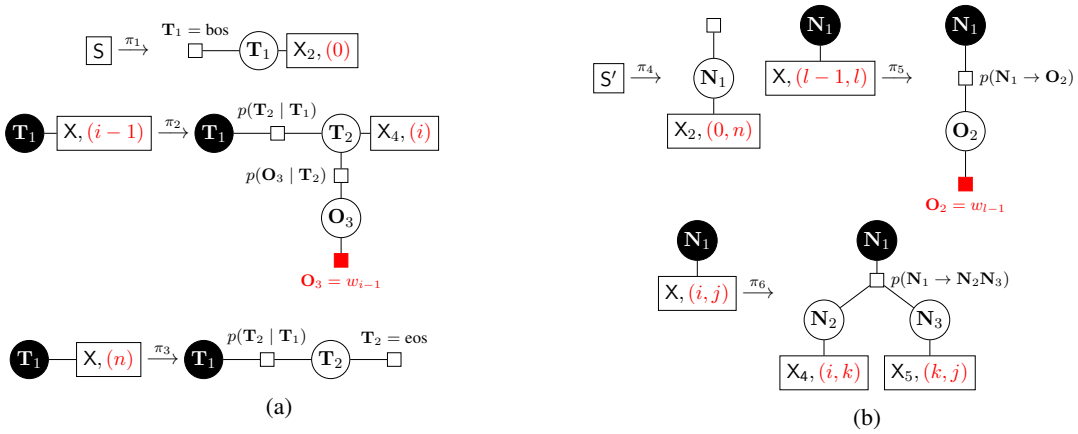


Figure 1: FGG representations of (a) HMMs and (b) PCFGs. Examples come from Chiang and Riley (2020).

parsing and HMM language modeling. For PCFG induction, we manage to use 20 times more hidden states than Yang et al. (2021b), obtaining much better unsupervised parsing performance. For HMM language modeling, we achieve lower perplexity and lower inference complexity than Chiu et al. (2021).

2 Background

2.1 Factor graph grammar

Factor graphs are fixed-sized and thus incapable of modeling substructures that repeat a variable number of times. Chiang and Riley (2020) propose factor graph grammars (FGGs) to overcome this limitation, which are expressive enough to subsume HMMs and PCFGs.

2.1.1 Basics

We display necessary notations and concepts of FGGs (Chiang and Riley, 2020, Def. 1,2,5,6,8).

Definition 1. A hypergraph is a tuple $(V, E, att, lab^V, lab^E)$ where

- V and E are finite set of nodes and hyperedges.
- $att : E \to V^\star$ maps each hyperedge to zero or more (not necessarily distinct) endpoint nodes.
- $lab^V : V \to L^V$ assigns labels to nodes.
- $lab^E : E \to L^E$ assigns labels to edges.

Definition 2. A factor graph is a hypergraph with mappings Ω and F where

- Ω maps node labels to sets of possible values. $\Omega(v) \triangleq \Omega(lab^V(v))$.
- F maps edge labels to functions. $F(e) \triangleq F(lab^E(e))$ is of type $\Omega(v_1) \times \cdots \times \Omega(v_k)$ where $att(e) = v_1 \cdots v_k$.

In the terminology of factor graphs, a node v with its domain $\Omega(v)$ is a *variable*, and an hyperedge e with $F(e)$ is a *factor*. We typically use $\mathbf{T}, \mathbf{N}, \mathbf{O}$ to denote hidden state, nonterminal state and observation variables for HMMs and PCFGs.

Definition 3. A hypergraph fragment is a tuple $(V, E, att, lab^V, lab^E, ext)$ where

- $(V, E, att, lab^V, lab^E)$ is a hypergraph.
- $ext \in V^\star$ is a set of zero or more external nodes and each of which can be seen as a connecting point of this hypergraph fragment with another fragment.

Definition 4. A hyperedge replacement graph grammar (HRG) (Drewes et al., 1997) is a tuple (N, T, P, S) where

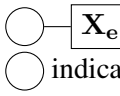
- $N, T \subset L^E$ is finite set of nonterminal and terminal symbols. $N \cap T = \emptyset$.
- P is a finite set of rules $(X \to R)$ where $X \in N$ and R is a hypergraph fragment with edge labels in $N \cup T$ [1].
- $S \in N$ is the start symbol.

Definition 5. A HRG with mapping Ω, F (Def. 2) is referred to as an FGG. In particular, F is defined on terminal edge labels T only.

Notations.

- $\mathbf{N}$: variable node. $\mathbf{N}$: external node.

[1] Note that, for the lhs of P, Chiang and Riley (2020) also draw their endpoint nodes using external node notations. We follow this practice.

- 

: hyperedge e with label $X \in N$. indicates zero or more endpoint nodes.

- $F(e)$: factor $F(e)$.

Fig. 1 illustrates HGG representations of HMM and PCFG.

Generative story. An FGG starts with $\boxed{\mathbf{S}}$, repeatedly selects $\boxed{\mathbf{X_e}}$ and uses rule $X \to R$ from P to replace e with R, until no $\boxed{\mathbf{X_e}}$ exists.

2.1.2 Conjunction

The conjunction operation (Chiang and Riley, 2020, Sec. 4) allows modularizing an FGG into two parts, one defining the model and the other defining a query. In this paper, we only consider querying the observed sentence $w_0, \cdots, w_{n-1}$, which is exemplified by the red part of Fig. 1. We sometimes omit the red part without further elaboration.

2.1.3 Inference

Denote ξ as an assignment of all variables, Ξ_D as the set of all assignments of factor graph D, and $\mathcal{D}(G)$ as the set of all derivations of an FGG G, i.e., all factor graphs generated by G. an FGG G assigns a score $w_G(D, \xi)$ to each $D \in \mathcal{D}(G)$ along with each $\xi \in \Xi_D$. A factor graph $D \in \mathcal{D}(G)$ assigns a score $w_D(\xi)$ to each $\xi \in \Xi_D$:

$$w_D(\xi) = \prod_{e \in D} F(e)(\xi(v_1), \ldots, \xi(v_k)) \quad (1)$$

with $att(e) = v_1 \cdots v_k$. Notably, $w_D(\xi) \triangleq w_G(D, \xi)$. The inference problem is to compute the sum-product of G:

$$Z_G = \sum_{D \in \mathcal{D}(G)} \sum_{\xi \in \Xi_D} w_G(D, \xi) \quad (2)$$

To obtain Z_G, the key difficulty is in the marginalization over all derivations, since $\sum_{\xi \in \Xi_D} w_D(\xi)$ can be obtained by running standard variable elimination (VE) on factor graph D. To tackle this, Chiang and Riley (2020, Thm. 15) propose an extended VE. For each $X \in N, \xi \in \Xi_X$ [2], define P^X as all rules in P with left-hand side X, and then define:

$$\psi_X(\xi) = \sum_{(X \to R) \in P^X} \tau_R(\xi). \quad (3)$$

[2] Ξ_X is defined as the set of assignments to the endpoints of an edge e labeled X, so $\Xi_X = \Omega(\ell_1) \times \cdots \times \Omega(\ell_k)$ where $att(e) = v_1 \cdots v_k, lab^V(v_i) = \ell_i$.

for each rhs $R = (V, E_N \cup E_T, att, lab^V, lab^E, ext)$, where E_N, E_T consist of nonterminal/terminal-labeled edges only, and $\tau_R(\xi)$ is given by:

$$\tau_R(\xi) = \sum_{\substack{\xi' \in \Xi_R \\ \xi'(ext) = \xi}} \prod_{e \in E_T} F(e)\left(\xi'(\mathrm{att}(e))\right) \prod_{e \in E_N} \psi_{lab^E(e)}\left(\xi'(\mathrm{att}(e))\right) \quad (4)$$

This defines a recursive formula for computing ψ_S, i.e., Z_G. Next, we will show how Eq. 3-4 recover the well-known inside algorithm.

Example: the inside algorithm. Consider π_6 in Fig. 2(b). All possible fragments R (rhs of π_6) differs in the value of k, i.e., the splitting point, so we use R_k to distinguish them. Then Eq. 3 becomes:

$$\psi_{X_{i,k}}(\xi) = \sum_{i<k<j} \tau_{R_k}(\xi) \quad (5)$$

Putting values into Eq. 4:

$$\tau_{R_k}(\xi) = \sum_{n_2, n_3} p(\xi, n_2, n_3) \psi_{X_{i,k}}(n_2) \psi_{X_{k,j}}(n_3) \quad (6)$$

where p denotes FGG rule probability $p(\mathbf{N}_1 \to \mathbf{N}_2\mathbf{N}_3)$. It is easy to see that $\psi_{X_{i,k}}$ is exactly the inside score of span $[i, k)$, and Eq. 5-6 recovers the recursive formula of the inside algorithm.

Remark. Eq. 4 can be viewed as unidirectional (from $e \in E_N$ to external nodes) belief propagation (BP) in the factor graph fragment R, where the incoming message is $\psi_{lab^E}(e)$ for $e \in E_N$, and the outcome of Eq. 4 can be viewed as the message passed to the external nodes. The time complexity of message updates grows exponentially with the number of variables in the factors. Therefore, to decrease inference complexity, one may decompose large factors into smaller factors connected by new nodes, as shown in the next subsection.

2.2 Tensor rank decomposition on factors

Consider a factor $F(e)$ (Def. 2), it can be represented as an order-k tensor in $\mathbb{R}^{N_1 \times \cdots \times N_k}$ where $N_i \triangleq |\Omega(v_i)|$. We can use tensor rank decomposition (aka. CPD) to decompose $F(e)$ into a weighted sum of outer products of vectors:

$$F(e) = \sum_{q=1}^{r} \lambda_q \mathbf{w}_{e_1}^q \otimes \mathbf{w}_{e_2}^q \otimes \cdots \otimes \mathbf{w}_{e_k}^q$$

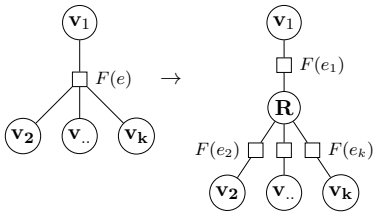


Figure 2: Using CPD to decompose a factor can be seen as adding a new node.

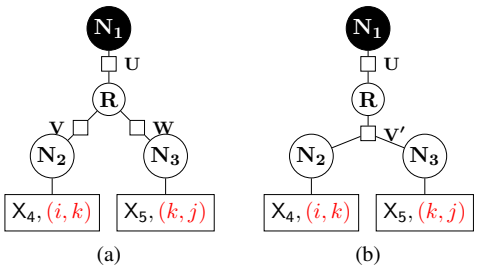


Figure 3: Representations of the rhs of π_6 (Fig. 1) after decomposition. (a): TD-PCFG (Cohen et al., 2013; Yang et al., 2021b). (b): LPCFG (Chiu et al., 2021).

where r is the rank size; $\mathbf{w}^q_{e_k} \in \mathbb{R}^{N_k}$; $\otimes$ is outer product; λ_q is weight, which can be absorbed into $\{\mathbf{w}^q_{e_k}\}$ and we omit it throughout the paper.

Dupty and Lee (2020, Sec. 4.1) show that BP can be written in the following matrix form when applying CPD on factors:

$$\mathbf{m}_{ei} = \mathbf{W}^T_{e_i}\left(\odot_{j\in N(e)\setminus \mathrm{i}} \mathbf{W}_{e_j}\mathbf{n}_{je}\right) \tag{7}$$

$$\mathbf{n}_{ie} = \odot_{c\in N(i)\setminus e}\mathbf{m}_{ci} \tag{8}$$

where $\mathbf{m}_{ei} \in \mathbb{R}^{N_i}$ is factor-to-node message; $\mathbf{n}_{ie} \in \mathbb{R}^{N_i}$ is node-to-factor message; $N(\cdot)$ indicates neighborhood ; $\mathbf{W}_{e_j} = [\mathbf{w}^1_{e_j}, \cdots, \mathbf{w}^r_{e_j}]^T \in \mathbb{R}^{r\times m}$; $\odot$ is element-wise product. We remark that this amounts to replacing the large factor $F(e)$ with smaller factors $\{F(e_i)\}$ connected by a new node $\mathbf{R}$ that represents rank, where each $F(e_i)$ can be represented as $\mathbf{W}_{e_i}$. Fig. 2 illustrates this intuition. We refer to $\mathbf{R}$ as rank nodes and others as state nodes thereafter.

3 Low-rank structured inference

In this section, we recover the accelerated inside algorithms of TD-PCFG (Cohen et al., 2013; Yang et al., 2021b) and LPCFG (Chiu et al., 2021) in an intuitive and unifying manner using the FGG notations. The accelerated forward algorithm of LHMM (Chiu et al., 2021) can be derived similarly.

Denote $\mathsf{T} \in \mathbb{R}^{m\times m\times m}$ as the tensor representation of $p(\mathbf{N}_1 \rightarrow \mathbf{N}_2\mathbf{N}_3)$, and $\boldsymbol{\alpha}_{i,j} \in \mathbb{R}^m$ as the inside score of span $[i,j)$. Cohen et al. (2013) and Yang et al. (2021b) use CPD to decompose T, i.e., let $\mathsf{T} = \sum_{q=1}^r \mathbf{u}_q \otimes \mathbf{v}_q \otimes \mathbf{w}_q$ where $\mathbf{u}_q, \mathbf{v}_q, \mathbf{w}_q \in \mathbb{R}^m$. Denote $\mathbf{U}, \mathbf{V}, \mathbf{W} \in \mathbb{R}^{r\times m}$ as the resulting matrices of stacking all $\mathbf{u}_q, \mathbf{v}_q, \mathbf{w}_q$, Cohen et al. (2013) derived the recursive form:

$$\boldsymbol{\alpha}_{i,j} = \sum_{k=i+1}^{j-1} \mathbf{U}^T\left((\mathbf{V}\boldsymbol{\alpha}_{i,k}) \odot (\mathbf{W}\boldsymbol{\alpha}_{k,j})\right) \tag{9}$$

$$= \mathbf{U}^T \sum_{k=i+1}^{j-1} \left((\mathbf{V}\boldsymbol{\alpha}_{i,k}) \odot (\mathbf{W}\boldsymbol{\alpha}_{k,j})\right) \tag{10}$$

Eq. 9 can be derived automatically by combining Eq. 7 (or Fig. 3 (a)) and Eq. 5-6. Cohen et al. (2013) note that $\mathbf{U}^\mathbf{T}$ can be extracted to the front of the summation (Eq. 10), and $\mathbf{V}\boldsymbol{\alpha}_{i,k}, \mathbf{W}\boldsymbol{\alpha}_{k,j}$ can be cached and reused, leading to further complexity reduction. The resulting inside algorithm time complexity is $O(n^3r + n^2mr)$.

Recently, Chiu et al. (2021) use low-rank matrix decomposition to accelerate PCFG inference. They first perform tensor matricization to flatten T to $\mathsf{T}' \in \mathbb{R}^{m\times m^2}$, and then let $\mathsf{T}' = \mathbf{U}^T\mathbf{V}$ where $\mathbf{U} \in \mathbb{R}^{r\times m}, \mathbf{V} \in \mathbb{R}^{r\times m^2}$. By un-flattening $\mathbf{V}$ to $\mathbf{V'} \in \mathbb{R}^{r\times m\times m}$, their accelerated inside algorithm has the following recursive form:

$$\boldsymbol{\alpha}_{i,j} = \sum_{k=i+1}^{j-1} \mathbf{U}^T\left(\mathbf{V'}\cdot\boldsymbol{\alpha}_{k,j}\cdot\boldsymbol{\alpha}_{i,k}\right) \tag{11}$$

$$= \mathbf{U}^T \sum_{k=i+1}^{j-1} \left(\mathbf{V'}\cdot\boldsymbol{\alpha}_{k,j}\cdot\boldsymbol{\alpha}_{i,k}\right) \tag{12}$$

Eq. 11 can be derived by combining Fig. 3 (b) and Eq. 5-6. The resulting inside time complexity is $O(n^3m^2r + n^2mr)$, which is higher than that of TD-PCFG.

Validity of probability. A remaining problem is how to ensure that T is a valid (non-negative and properly normalized) probability tensor. We discuss this in Appd. A.

4 Rank-space modeling and inference

4.1 Rank-space inference with B-FGGs

Interestingly, when applying CPD on factors and if the rank size is smaller than the state size, we can even obtain better inference time complexities for a subset of FGGs which we refer to as B-FGGs.

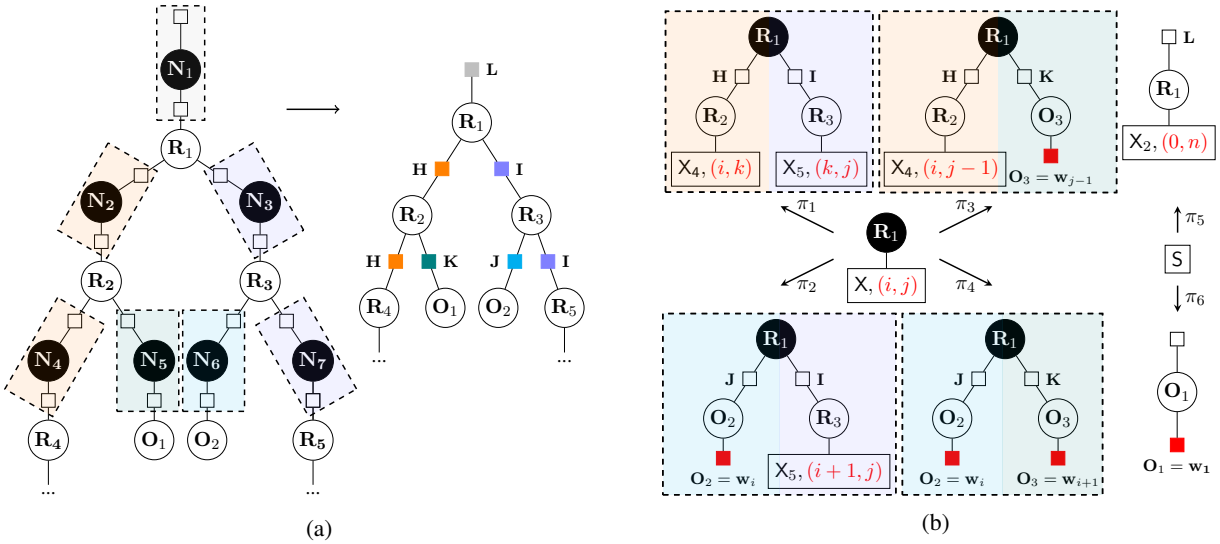


Figure 4: (a): illustration of marginalizing state nodes $\mathbf{N}$. (b): rule set of the new FGG. π_1 can be applied when $k \neq i+1$ and $k+1 \neq j$; π_2 and π_3 can be applied when $i \neq j-1$; π_4 can be applied when $j = i+2$.

We call a hyperedge a B-edge if its head contains exactly one node. B-graphs (Gallo et al., 1993) are a subset of directed hypergraphs whose hyperedges are all B-edges. Many dynamic programming algorithms can be formulated through B-graphs (Klein and Manning, 2001; Huang, 2008; Azuma et al., 2017; Chiu et al., 2021; Fu and Lapata, 2021), including the inference algorithms of many structured models, e.g., HMMs, Hidden Semi-Markov Models (HSMMs), and PCFGs. We follow the concept of B-graphs to define B-FGGs.

Definition 6. A hypergraph fragment is a B-hypergraph fragment iff. there is exactly one external node and there is no nonterminal-labeled hyperedge connecting to it. An FGG is a B-FGG iff. all rhs of its rules are B-hypergraph fragments.

It is easy to see that the aforementioned models are subsumed by B-FGGs. We can design a family of accelerated inference algorithms for B-FGGs based on the following strategy. (1) If there are multiple factors within a hypergraph fragment, merge them into a single factor. Then apply CPD on the single factor, thereby introducing rank nodes. (2) Find repeated substructures that take rank nodes as external nodes. Marginalize all state nodes to derive new rules. (3) Design new inference algorithms that can be carried out in the rank space based on the general-purpose FGG inference algorithm and the derived new rules.

We give two examples, the rank-space inside algorithm and the rank-space forward algorithm, in the following two subsections to help readers understand this strategy.

4.2 The rank-space inside algorithm

Consider an B-FGG G shown in Fig. 1(b) and replace the rhs of π_6 with Fig. 3(a), i.e., we use CPD to decompose binary rule probability tensor. Besides $\mathbf{U}, \mathbf{V}, \mathbf{W} \in \mathbb{R}^{r \times m}$ defined in Sec. 3, we define the start rule probability vector as $\mathbf{s} \in \mathbb{R}^{m \times 1}$, and the unary rule probability matrix as $\mathbf{E} \in \mathbb{R}^{o \times m}$ where o is the vocabulary size.

Fig. 4(a) is an example (partial) factor graph D generated by G. We highlight substructures of interest with dashed rectangles. Each substructure consists of a node $\mathbf{N}$ and two factors connecting to it. $\mathbf{N}$ is an external node connecting two hypergraph fragments which contain the two factors respectively. For each substructure, we can marginalize the state node $\mathbf{N}$ out, merging the two factors into a single one. After marginalizing all state nodes, we obtain a (partial) factor graph D' shown in the right of Fig. 4(a) where $\mathbf{H} = \mathbf{V}\mathbf{U}^T, \mathbf{I} = \mathbf{W}\mathbf{U}^T, \mathbf{J} = \mathbf{V}\mathbf{E}^T, \mathbf{K} = \mathbf{W}\mathbf{E}^T$, $\mathbf{L} = (\mathbf{U}\mathbf{s})^T$. We denote this transformation as $\mathcal{M}(D) = D'$. We define a new B-FGG G' with rules shown in Fig 4(b). It is easy to verify that for each $D \in \mathcal{D}(G)$, we have $\mathcal{M}(D) \in \mathcal{D}(G')$, and vice versa. Moreover, we have:

$$\sum_{\xi \in \Xi_D} w_G(D, \xi) = \sum_{\xi \in \Xi_{\mathcal{M}(D)}} w_{G'}(\mathcal{M}(D), \xi)$$

because marginalizing hidden variables does not affect the result of sum-product inference. Therefore, $Z_G = Z_{G'}$ (Eq. 2).

We can easily derive the inference (inside) algorithm of G' by following Eq. 3-4 and Fig. 4(b) [3]. Let $\boldsymbol{\alpha}_{i,j} \in \mathbb{R}^r$ denote the rank-space inside score for span $[i, j)$. When $j > i + 2$:

$$\boldsymbol{\alpha}_{i,j} = \overbrace{\sum_{i+1<k<j-1} (\mathbf{H}\boldsymbol{\alpha}_{i,k} \odot \mathbf{I}\boldsymbol{\alpha}_{k,j})}^{\text{from } \pi_1 \text{ of Fig. 4(b)}} + \underbrace{\mathbf{J}_{:,w_i} \odot \mathbf{I}\boldsymbol{\alpha}_{i+1,j}}_{\text{from } \pi_2} + \underbrace{\mathbf{H}\boldsymbol{\alpha}_{i,j-1} \odot \mathbf{K}_{:,w_{j-1}}}_{\text{from } \pi_3}$$

and when $j = i + 2$, $\boldsymbol{\alpha}_{i,j} = \mathbf{J}_{:,w_i} \odot \mathbf{K}_{:,w_{i+1}}$ (from π_4). w_j is the index of the j-th word of the input sentence in the vocabulary; $\mathbf{A}_{:,j}$ indicates the j-th column of $\mathbf{A}$.

We note that, similar to Cohen et al. (2013), we can cache $\mathbf{H}\boldsymbol{\alpha}_{i,k}, \mathbf{I}\boldsymbol{\alpha}_{k,j}$ and reuse them to further accelerate inference [4]. Denote $\boldsymbol{\alpha}^L_{i,j}, \boldsymbol{\alpha}^R_{i,j} \in \mathbb{R}^r$ as the inside scores of span $[i, j)$ serving as a left/right child of a larger span. Then we have:

$$\boldsymbol{\alpha}^L_{i,i+1} = \mathbf{K}_{:,i} \qquad \boldsymbol{\alpha}^R_{i,i+1} = \mathbf{J}_{:,i}$$
$$\boldsymbol{\alpha}^L_{i,j} = \mathbf{H}\boldsymbol{\alpha}_{i,j} \qquad \alpha^R_{i,j} = \mathbf{I}\boldsymbol{\alpha}_{i,j}$$
$$\boldsymbol{\alpha}_{i,j} = \sum_{i<k<j} (\boldsymbol{\alpha}^L_{i,k} \odot \boldsymbol{\alpha}^R_{k,j})$$

and finally, $Z_{G'} = \mathbf{L}\boldsymbol{\alpha}_{0,n}$. The resulting inference complexity is $O(n^3r + n^2r^2)$, which is lower than $O(n^3r + n^2mr)$ of TD-PCFG when $r < m$, enabling the use of a large state space for PCFGs in the low-rank setting.

The key difference between the rank-space inference and the original state-space inference is that they follow different variable elimination orders. The former marginalizes all state nodes before performing inference and marginalizes rank nodes from bottom up during inference; whereas the later marginalizes both state and rank nodes alternately from bottom up during inference.

Low-rank inference does not support the Viterbi semiring [5], inhibiting the use of CYK decoding. Therefore, we resort to Minimum Bayes-Risk decoding, similar to Yang et al. (2021b). Specifically, we estimate the span marginals using auto-differentiation (Eisner, 2016; Rush, 2020), which has the same complexity as the inside algorithm. Then we use the CYK algorithm to find the final parse with the maximum number of expected spans in cubic time. For unsupervised learning, we minimize $-\log Z_{G'}$ using gradient descent.

4.3 The rank-space forward algorithm

Consider an B-FGG G shown in Fig. 1 (a). We replace the rhs of π_2 by the hypergraph fragment in the right of Fig. 5(a), i.e., we merge the factor $p(\mathbf{T}_2 \mid \mathbf{T}_1)$ and $p(\mathbf{O}_3 \mid \mathbf{T}_2)$ into a single factor, which can be represented as $\mathsf{T} \in \mathbb{R}^{m \times m \times o}$ and can be decomposed into three matrices $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{r \times m}, \mathbf{W} \in \mathbb{R}^{r \times o}$ via CPD, where $m/o/r$ is the state/vocabulary/rank size. Fig. 5(b) gives an example factor graph of HMMs with sentences of length 3. Similar to previous subsection, we marginalize state nodes $\mathbf{T}$ to construct a new B-FGG G'. The rule set of G' can be obtained by replacing all variable nodes $\mathbf{T}$ with $\mathbf{R}$ and modifying all factors accordingly, as one can easily infer from Fig. 5(c). Inference with G' simply coincides with the forward algorithm, which has a $O(nr^2)$ time complexity and is lower than $O(nmr)$ of LHMM (Chiu et al., 2021) when $r < m$.

4.4 Neural parameterization

We use neural networks to produce probabilities for all factors, which has been shown to benefit learning in previous work (Kim et al., 2019; Yang et al., 2021b; Chiu and Rush, 2020; Chiu et al., 2021). We use the neural parameterization of Yang et al. (2021b) with slight modifications. We show the details in Appd. B and Appd. C.

5 Experiments

5.1 Unsupervised parsing with PCFGs

Setting. We evaluate our model on Penn Treebank (PTB) (Marcus et al., 1994). Our implementation is based on the open-sourced code of Yang et al. (2021b)[6] and we use the same setting as theirs. For all experiments, we set the ratio of nonterminal number to the preterminal number to 1:2 [7]. We set the rank size to 1000. We show other details in Appd. D and E.

Main result. Table 1 shows the result on PTB. Among previous unsupervised PCFG models, TN-

[3] π_6 is used for generating sentences of length 1, we do not consider this in the following derivation of the inside algorithm to reduce clutter.

[4] In fact, this is a typical application of the unfold-refold transformation (Eisner and Blatz, 2007; Vieira et al., 2021).

[5] The Viterbi semiring is also known as the max-product semiring. Chiu et al. (2021, Appd. C) and Yang et al. (2021b, Sec. 6) have discussed this issue.

[6] github.com/sustcsonglin/TN-PCFG

[7] Although we did not explicitly distinguish between nonterminal and preterminal symbols before, in our implementation, we follow Kim et al. (2019) to make such distinction.

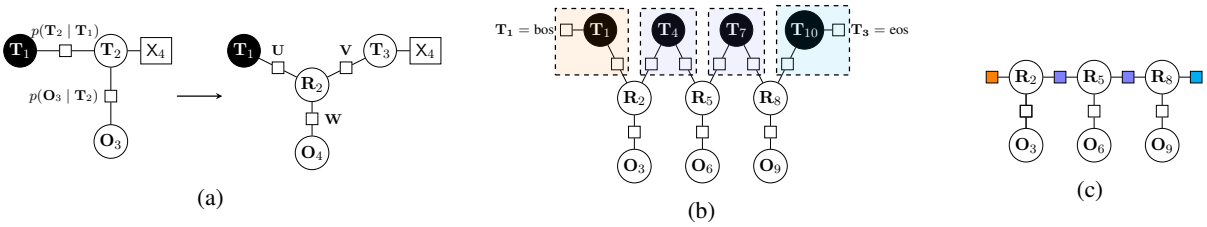


Figure 5: (a): merge the two factors into a single one, and apply CPD on the resulting factor. (b): factor graph of a HMM for sentences of length 3. (c): the resulting factor graph after marginalizing the state nodes.

Model	S-F1
N-PCFG (Kim et al., 2019)	50.8
C-PCFG (Kim et al., 2019)	55.2
NL-PCFG (Zhu et al., 2020)	55.3
TN-PCFG (Yang et al., 2021b)	57.7
NBL-PCFG (Yang et al., 2021a)	60.4
Ours with 9000 PTs and 4500 NTs	**64.1**
For reference	
Constituency test (Cao et al., 2020)	62.8
S-DIORA (Drozdov et al., 2020)	57.6
StructFormer (Shen et al., 2021)	54.0
DIORA+span constraint (Xu et al., 2021)	61.2

Table 1: Results on PTB. S-F1: sentence-level F1. PTs: preterminals. NTs: nonterminals.

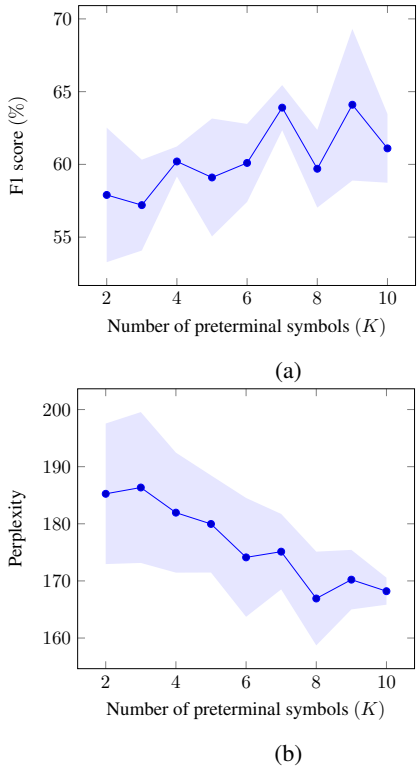


Figure 6: The change of F1 scores and perplexities with the change of number of perterminal symbols.

PCFG (Yang et al., 2021b) uses the largest number of states (500 perterminals and 250 nonterminals). Our model is able to use much more states thanks to our new inside algorithm with lower time complexity, surpassing all previous PCFG-based models by a large margin and achieving a new state-of-the-art in unsupervised constituency parsing in terms of sentence-level F1 score on PTB.

Ablation study. Fig. 6 shows the change of the sentence-level F1 scores and perplexity with the change of the number of preterminals. As we can see, when increasing the state, the perplexity tends to decrease while the F1 score tends to increase, validating the effectiveness of using large state spaces for neural PCFG induction.

5.2 HMM language modeling

Setting. We conduct the language modeling experiment also on PTB. Our implementation is based on the open-sourced code of Chiu et al. (2021)[8]. We set the rank size to 4096. See Appd. D and E for more details.

Main result. Table 2 shows the perplexity on the PTB validation and test sets. As discussed earlier, VL-HMM (Chiu and Rush, 2020) imposes strong sparsity constraint to decrease the time complexity of the forward algorithm and requires pre-clustering of terminal symbols. Specifically, VL-HMM uses Brown clustering (Brown et al., 1992), introducing external information to improve performance. Replacing Brown clustering with uniform clustering leads to a 10 point increase in perplexity on the PTB validation set. LHMM (Chiu et al., 2021) and our model only impose low-rank constraint without using any external information and are thus more comparable. Our method outperforms LHMM by 4.8 point when using the same state number (i.e., 2^{14}), and it can use more states thanks to our lower inference time complexity.

[8] github.com/justinchiu/low-rank-models

Model	Val	Test
VL-HMM (2^{15} states, Brown)	125.0	116.0
VL-HMM (2^{14} states, Brown)†	136	-
VL-HMM (2^{14} states, Uniform)†	146	-
LHMM (2^{14} states)	141.4	131.8
Ours (2^{14} states)	135.6	127.0
Ours (2^{15} states)	137.0	126.4
For reference		
HMM+RNN (Buys et al., 2018)	142.3	-
AWD-LSTM (Merity et al., 2018)	60.0	57.3

Table 2: Resulting perplexity on PTB validate set and test set. VL-HMM: (Chiu and Rush, 2020). LHMM: (Chiu et al., 2021). † denotes results reported by ablation study of Chiu and Rush (2020).

#States	Val	Test
2^{12}	149.8	139.1
2^{13}	143.8	133.4
2^{14}	149.5	137.4
2^{15}	141.1	131.1

Table 3: Perplexity with varying numbers of states. Following Chiu et al. (2021), we fix the rank to 2048 for faster ablation studies.

Ablation study. As we can see in Table 3, the perplexity tends to decrease when increasing the state number, validating the effectiveness of using more states for neural HMM language modeling.

6 Related work

Tensor and matrix decomposition have been used to decrease time and space complexities of probabilistic inference algorithms. Siddiqi et al. (2010) propose a reduced-rank HMM whereby the forward algorithm can be carried out in the rank space, which is similar to our model, but our method is more general. Cohen and Collins (2012); Cohen et al. (2013) use CPD for fast (latent-variable) PCFG parsing, but they do not leverage CPD for fast learning and they need to actually perform CPD on existing probability tensors. Rabusseau et al. (2016) use low-rank approximation method to learn weighted tree automata, which subsumes PCFGs and latent-variable PCFGs. Our method can subsume more models. Yang et al. (2021b,a) propose CPD-based neural parameterizations for (lexicalized) PCFGs. Yang et al. (2021b) aim at scaling PCFG inference. We achieve better time complexity than theirs and hence can use much more hidden states. Yang et al. (2021a) aims to decrease the complexity of lexicalized PCFG parsing, which can also be described within our framework. Chiu et al. (2021) use low-rank matrix decomposition, which can be viewed as CPD on order-2 tensors, to accelerate inference on chain and tree structure models including HMMs and PCFGs. However, their method is only efficient when the parameter tensors are of order 2, e.g., in HMMs and HSMMs. Our method leverages full CPD, thus enabling efficient inference with higher-order factors, e.g., in PCFGs. Our method can be applied to all models considered by Chiu et al. (2021), performing inference in the rank-space with lower complexities.

Besides HMMs and PCFGs, Wrigley et al. (2017) propose an efficient sampling-based junction-tree algorithm using CPD to decompose high-order factors. Dupty and Lee (2020) also use CPD to decompose high-order factors for fast belief propagation. Ducamp et al. (2020) use tensor train decomposition for fast and scalable message passing in Bayesian networks. Bonnevie and Schmidt (2021) leverage matrix product states (i.e., tensor trains) for scalable discrete probabilistic inference. Miller et al. (2021) leverage tensor networks for fast sequential probabilistic inference.

7 Conclusion and future work

In this work, we leveraged tensor rank decomposition (CPD) for low-rank scaling of structured inference. We showed that CPD amounts to decomposing a large factor into several smaller factors connected by a new rank node, and gave a unifying perspective towards previous low-rank structured models (Yang et al., 2021b; Chiu et al., 2021). We also presented a novel framework to design a family of rank-space inference algorithms for B-FGGs, a subset of FGGs which subsume most structured models of interest to the NLP community. We have shown the application of our method in scaling PCFG and HMM inference, and experiments on unsupervised parsing and language modeling validate the effectiveness of using large state spaces facilitated by our method.

We believe our framework can be applied to many other models which have high inference time complexity and are subsumed by B-FGGs, including lexicalized PCFGs, quasi-synchronous context-free grammars (QCFGs), etc. A direct application of our method is to decrease the inference complexity of the neural QCFG model (Kim, 2021), which has a very large grammar constant and can be improved easily under our framework.

References

Ai Azuma, Masashi Shimbo, and Yuji Matsumoto. 2017. An algebraic formalization of forward and forward-backward algorithms. *CoRR*, abs/1702.06941.

Rasmus Bonnevie and Mikkel N. Schmidt. 2021. Matrix product states for inference in discrete probabilistic models. *Journal of Machine Learning Research*, 22(187):1–48.

Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jenifer C. Lai, and Robert L. Mercer. 1992. Class-based *n*-gram models of natural language. *Computational Linguistics*, 18(4):467–480.

Jan Buys, Yonatan Bisk, and Yejin Choi. 2018. Bridging hmms and rnns through architectural transformations.

Steven Cao, Nikita Kitaev, and Dan Klein. 2020. Unsupervised parsing via constituency tests. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4798–4808, Online. Association for Computational Linguistics.

David Chiang and Darcey Riley. 2020. Factor graph grammars. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Justin Chiu, Yuntian Deng, and Alexander Rush. 2021. Low-rank constraints for fast inference in structured models. *Advances in Neural Information Processing Systems*, 34.

Justin Chiu and Alexander Rush. 2020. Scaling hidden Markov language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1341–1349, Online. Association for Computational Linguistics.

Shay B. Cohen and Michael Collins. 2012. Tensor decomposition for fast parsing with latent-variable pcfgs. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 2528–2536.

Shay B. Cohen, Giorgio Satta, and Michael Collins. 2013. Approximate PCFG parsing using tensor decomposition. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 487–496, Atlanta, Georgia. Association for Computational Linguistics.

Frank Drewes, Hans-Jörg Kreowski, and Annegret Habel. 1997. Hyperedge replacement, graph grammars. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pages 95–162. World Scientific.

Andrew Drozdov, Subendhu Rongali, Yi-Pei Chen, Tim O'Gorman, Mohit Iyyer, and Andrew McCallum. 2020. Unsupervised parsing with S-DIORA: Single tree encoding for deep inside-outside recursive autoencoders. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4832–4845, Online. Association for Computational Linguistics.

Gaspard Ducamp, Philippe Bonnard, Anthony Nouy, and Pierre-Henri Wuillemin. 2020. An efficient low-rank tensors representation for algorithms in complex probabilistic graphical models. In *International Conference on Probabilistic Graphical Models, PGM 2020, 23-25 September 2020, Aalborg, Hotel Comwell Rebild Bakker, Skørping, Denmark*, volume 138 of *Proceedings of Machine Learning Research*, pages 173–184. PMLR.

Mohammed Haroon Dupty and Wee Sun Lee. 2020. Neuralizing efficient higher-order belief propagation. *CoRR*, abs/2010.09283.

Jason Eisner. 2016. Inside-outside and forward-backward algorithms are just backprop (tutorial paper). In *Proceedings of the Workshop on Structured Prediction for NLP*, pages 1–17, Austin, TX. Association for Computational Linguistics.

Jason Eisner and John Blatz. 2007. Program transformations for optimization of parsing algorithms and other weighted logic programs. In *Proceedings of FG 2006: The 11th Conference on Formal Grammar*, pages 45–85. CSLI Publications.

Yao Fu and Mirella Lapata. 2021. Scaling structured inference with randomization. *CoRR*, abs/2112.03638.

Giorgio Gallo, Giustino Longo, and Stefano Pallottino. 1993. Directed hypergraphs and applications. *Discret. Appl. Math.*, 42(2):177–201.

Liang Huang. 2008. Advanced dynamic programming in semiring and hypergraph frameworks. In *Coling 2008: Advanced Dynamic Programming in Computational Linguistics: Theory, Algorithms and Applications - Tutorial notes*, pages 1–18, Manchester, UK. Coling 2008 Organizing Committee.

Yoon Kim. 2021. Sequence-to-sequence learning with latent neural grammars. *Advances in Neural Information Processing Systems*, 34.

Yoon Kim, Chris Dyer, and Alexander 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, Florence, Italy. Association for Computational Linguistics.

Dan Klein and Christopher D. Manning. 2001. Parsing and hypergraphs. In *Proceedings of the Seventh International Workshop on Parsing Technologies (IWPT-2001), 17-19 October 2001, Beijing, China*. Tsinghua University Press.

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 *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. Regularizing and optimizing LSTM language models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Jacob Miller, Guillaume Rabusseau, and John Terilla. 2021. Tensor networks for probabilistic sequence modeling. In *The 24th International Conference on Artificial Intelligence and Statistics, AISTATS 2021, April 13-15, 2021, Virtual Event*, volume 130 of *Proceedings of Machine Learning Research*, pages 3079–3087. PMLR.

Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. 2017. Introduction to tensor decompositions and their applications in machine learning. *CoRR*, abs/1711.10781.

Guillaume Rabusseau, Borja Balle, and Shay B. Cohen. 2016. Low-rank approximation of weighted tree automata. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016*, volume 51 of *JMLR Workshop and Conference Proceedings*, pages 839–847. JMLR.org.

Alexander Rush. 2020. Torch-struct: Deep structured prediction library. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 335–342, Online. Association for Computational Linguistics.

Yikang Shen, Yi Tay, Che Zheng, Dara Bahri, Donald Metzler, and Aaron Courville. 2021. StructFormer: Joint unsupervised induction of dependency and constituency structure from masked language modeling. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7196–7209, Online. Association for Computational Linguistics.

Sajid M. Siddiqi, Byron Boots, and Geoffrey J. Gordon. 2010. Reduced-rank hidden markov models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pages 741–748. JMLR.org.

Tim Vieira, Ryan Cotterell, and Jason Eisner. 2021. Searching for more efficient dynamic programs. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3812–3830, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Andrew Wrigley, Wee Sun Lee, and Nan Ye. 2017. Tensor belief propagation. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3771–3779. PMLR.

Zhiyang Xu, Andrew Drozdov, Jay Yoon Lee, Tim O'Gorman, Subendhu Rongali, Dylan Finkbeiner, Shilpa Suresh, Mohit Iyyer, and Andrew McCallum. 2021. Improved latent tree induction with distant supervision via span constraints. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4818–4831, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Songlin Yang, Yanpeng Zhao, and Kewei Tu. 2021a. Neural bi-lexicalized PCFG induction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2688–2699, Online. Association for Computational Linguistics.

Songlin Yang, Yanpeng Zhao, and Kewei Tu. 2021b. PCFGs can do better: Inducing probabilistic context-free grammars with many symbols. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1487–1498, Online. Association for Computational Linguistics.

Hao Zhu, Yonatan Bisk, and Graham Neubig. 2020. The return of lexical dependencies: Neural lexicalized PCFGs. *Transactions of the Association for Computational Linguistics*, 8:647–661.

A Validity of probability

When learning a PCFG and a HMM, there is no need to first learn T and then perform decomposition on T. Instead, one can learn the decomposed matrices (e.g., $\mathbf{U}, \mathbf{V}$) to learn T implicitly. During inference, one can follow Eq. 10 or 12 without the need to reconstruct T. The remaining problem is to ensure T to be a valid probability tensor (i.e., nonnegative and properly normalized) when learning it implicitly. The solution of Yang et al. (2021b) is to transform Fig. 3(a) into a Bayesian network, adding directed arrows $\mathbf{N_1} \rightarrow \mathbf{R}, \mathbf{R} \rightarrow \mathbf{N_2}, \mathbf{R} \rightarrow \mathbf{N_3}$. This is equivalent to requiring that $\mathbf{V}, \mathbf{W}$ are nonnegative and column-wise normalized and $\mathbf{U}$ is nonnegative and row-wise normalized, as described in Yang et al. (2021b, Thm. 1).

B Neural parameterization of PCFGs

In this section, we give the full parameterization of PCFGs. We follow Yang et al. (2021b) with slight modifications for generations of $\mathbf{U}, \mathbf{V}, \mathbf{W} \in \mathbb{R}^{m \times r}$ in 4.2. We use the same MLPs with two residual layers as Yang et al. (2021b):

$$\mathbf{s} = \frac{\exp(\mathbf{u}_S^T h_1(\mathbf{w}_A)}{\sum_{A' \in \mathcal{N}} \exp(\mathbf{u}_S^T h_1(\mathbf{w}_{A'}))}$$

$$\mathbf{E} = \frac{\exp(\mathbf{u}_E^T h_2(\mathbf{w}_t)}{\sum_{E' \in \Sigma} \exp(\mathbf{u}_{E'}^T h_2(\mathbf{w}_t))}$$

$$\mathbf{U} = \frac{\exp(\mathbf{u}_H^T f_1(\mathbf{w}_n)}{\sum_{n' \in \mathcal{N}} \exp(\mathbf{u}_H^T f_1(\mathbf{w}_{n'}))}$$

$$\mathbf{V} = \frac{\exp(\mathbf{u}_H^T f_2(\mathbf{w}_l)}{\sum_{H' \in \mathcal{H}} \exp(\mathbf{u}_{H'}^T f_2(\mathbf{w}_l))}$$

$$\mathbf{W} = \frac{\exp(\mathbf{u}_H^T f_3(\mathbf{w}_l)}{\sum_{H' \in \mathcal{H}} \exp(\mathbf{u}_{H'}^T f_3(\mathbf{w}_l))}$$

$$h_i(\mathbf{x}) = g_{i,1}(g_{i,2}(\tilde{\mathbf{W}}_i \mathbf{x}))$$

$$g_{i,j}(\mathbf{y}) = ReLU(\tilde{\mathbf{V}}_{i,j} ReLU(\tilde{\mathbf{U}}_{i,j} \mathbf{y})) + \mathbf{y}$$

where Σ is the vocabulary set, $\mathcal{H}$ is the set of rank, $\mathcal{N}$ is a finite set of nonterminals, $\mathbf{W}_l = [\mathbf{W}_n; \mathbf{W}_t], \mathbf{w}_l, \mathbf{w}_n, \mathbf{w}_t \in \mathbf{W}_l, \mathbf{W}_n, \mathbf{W}_t$. The main differences of neural parameterization between ours and previous work are that we make the projection parameter $\mathbf{u}_H$ shared among $\mathbf{U}, \mathbf{V}$, and $\mathbf{U}$.

C Neural parameterization of HMMs

In this section, we give the full parameterization of HMMs, which is similar to PCFGs' parameterization. Define $\mathbf{s}$ as start probability for HMMs. And the definitions of $\mathbf{U}, \mathbf{V}, \mathbf{W}$ are same as definitions in 4.3:

$$\mathbf{s} = \frac{\exp(\mathbf{u}_P^T h_1(\mathbf{w}_s))}{\sum_{s' \in \mathcal{S}} \exp(\mathbf{u}_P^T h_1(\mathbf{w}_{s'}))}$$

$$\mathbf{U} = \frac{\exp(\mathbf{u}_H^T \mathbf{w}_u)}{\sum_{H' \in \mathcal{H}} \exp(\mathbf{u}_{H'}^T \mathbf{w}_u)}$$

$$\mathbf{V} = \frac{\exp(\mathbf{u}_H^T \mathbf{w}_v)}{\sum_{v' \in \mathcal{S}} \exp(\mathbf{u}_H^T \mathbf{w}_{v'})}$$

$$\mathbf{W} = \frac{\exp(\mathbf{u}_W^T h_2(\mathbf{w}_w)}{\sum_{w' \in \Sigma} \exp(\mathbf{u}_W^T h_2(\mathbf{w}_{w'}))}$$

$$h_i(\mathbf{x}) = g_{i,1}(g_{i,2}(\tilde{\mathbf{W}}_i \mathbf{x}))$$

$$g_{i,j}(\mathbf{y}) = ReLU(\tilde{\mathbf{V}}_{i,j} ReLU(\tilde{\mathbf{U}}_{i,j} \mathbf{y})) + \mathbf{y}$$

where $\mathcal{S}$ is a finite set of states, $\mathcal{H}$ is the set of rank, Σ is vocabulary set.

D Data details

Penn Treebank (PTB) (Marcus et al., 1994)[9] consists of 929k training words, 73k validation words, and 82k test words, with a vocabulary of size 10k.

For PCFGs, we follow Yang et al. (2021b) and use their code to preprocess dataset. This processing discards punctuation and lowercases all tokens with 10k most frequent words as the vocabulary. The splits of the dataset are: 2-21 for training, 22 for validation and 23 for test.

For HMMs, we follow Chiu et al. (2021) and use their code to preprocess dataset. We lowercase all words and substitutes OOV words with UNKs. EOS tokens have been inserted after each sentence.

E Experimental details

For PCFGs, we use Xavier normal initialization to initialize the weights in h_i and f_i. We optimize our model using Adam optimizer with $\beta_1 = 0.75, \beta_2 = 0.999$, and the learning rate 0.002, setting the dimension of all embeddings to 256.

For HMMs, we initialize all parameters by Xavier normal initialization except for $\mathbf{w}_s$ and $\mathbf{w}_w$. We use AdamW optimizer with $\beta_1 = 0.99, \beta_2 = 0.999$, and the learning rate 0.001, and a max grad norm of 5. We use dropout rate of 0.1 to dropout $\mathbf{w}_s$ and $\mathbf{U}, \mathbf{V}$ in HMMs. We train for 30 epochs with a max batch size of 256 tokens, and reduce the learning by multiplying $\frac{1}{2}$ if the validation perplexity fails to improve after 2 evaluations. Evaluations are performed one time per epoch. We follow Chiu et al. (2021) to shuffle sentences and leverage bucket iterator, where batch of sentences are drawn from buckets containing sentences of similar lengths to minizing padding.

We run all experiments on NVIDIA TITAN RTX and NVIDIA RTX 2080ti and all experimental results are averaged from four runs.

[9] The licence of PTB dataset is LDC User Agreement for Non-Members, which can be seen on https://catalog.ldc.upenn.edu/LDC99T42