# Second-Order Unsupervised Neural Dependency Parsing

**Songlin Yang**[1,2,3], **Yong Jiang**[4], **Wenjuan Han**[5], **Kewei Tu**[1*]
[1]School of Information Science and Technology, ShanghaiTech University
[2]Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences
[3]University of Chinese Academy of Sciences
[4]Alibaba DAMO Academy, Alibaba Group
[5]Department of Computer Science, National University of Singapore
{yangsl,tukw}@shanghaitech.edu.cn
yongjiang.jy@alibaba-inc.com dcshanw@nus.edu.sg

**Abstract**

Most of the unsupervised dependency parsers are based on first-order probabilistic generative models that only consider local parent-child information. Inspired by second-order supervised dependency parsing, we proposed a second-order extension of unsupervised neural dependency models that incorporate grandparent-child or sibling information. We also propose novel design of the neural parameterization and optimization methods of the dependency models. In second-order models, the number of grammar rules grows cubically with the increase of vocabulary size, making it difficult to train lexicalized models that may contain thousands of words. To circumvent this problem while still benefiting from both second-order parsing and lexicalization, we use the agreement-based learning framework to jointly train a second-order unlexicalized model and a first-order lexicalized model. Experiments on multiple datasets show the effectiveness of our second-order models compared with recent state-of-the-art methods. Our joint model achieves a 10% improvement over the previous state-of-the-art parser on the full WSJ test set.[1]

## 1 Introduction

Dependency parsing is a classical task in natural language processing. The head-dependent relations produced by dependency parsing can provide an approximation to the semantic relationship between words, which is useful in many downstream NLP tasks such as machine translation, information extraction and question answering. Nowadays, supervised dependency parsers can reach a very high accuracy (Dozat and Manning, 2017; Zhang et al., 2020). Unfortunately, supervised parsing requires treebanks (annotated parse trees) for training, which are very expensive and time-consuming to build. On the other hand, unsupervised dependency parsing requires only unannotated corpora for training, though the accuracy of unsupervised parsing still lags far behind that of supervised parsing. We focus on unsupervised dependency parsing in this paper.

Most methods in the literature of unsupervised dependency parsing are based on the Dependency Model with Valence (DMV) (Klein and Manning, 2004), which is a probabilistic generative model. A main disadvantage of DMV and many of its extensions is that they lack expressiveness. The generation of a dependent token is only conditioned on its parent, the relative direction of the token to its parent, and whether its parent has already generated any child in this direction, hence ignoring other contextual information. To improve model expressiveness, researchers often turn to discriminative methods, which can incorporate more contextual information into the scoring or prediction of dependency arcs. For example, Grave and Elhadad (2015) uses the idea of disrciminative clustering, Cai et al. (2017) uses a discriminative parser in the CRF-autoencoder framework, and Li et al. (2018) uses an encoder-decoder framework that contains a discriminative transitioned-based parser. For DMV, Han et al. (2019) proposes the discriminative neural DMV which uses a global sentence embedding to introduce contextual information into the calculation of grammar rule probabilities. In the literature of supervised graph-based dependency parsing, however, there exists another technique for incorporating contextual information and increasing expressiveness, namely high-order parsing (Koo and Collins, 2010; Ma and Hai, 2012).

[*]Corresponding Author
[1]Our source code is available at: https://github.com/sustcsonglin/second-order-neural-dmv

A first-order parser, such as the DMV, only considers local parent-children information. In comparison, a high-order parser takes into account the interaction between multiple dependency arcs.

In this work, we propose the second-order neural DMV model, which incorporates second-order information (e.g., sibling or grandparent) into the original (neural) DMV model. To achieve better learning accuracy, we design a new neural architecture for rule probability computation and promote direct marginal likelihood optimization (Salakhutdinov et al., 2003; Tran et al., 2016) over the widely used expectation-maximization algorithm for training. One particular challenge faced by second-order neural DMVs is that the number of grammar rules grows cubically to the vocabulary size, making it difficult to store and train a lexicalized model containing thousands of words. Therefore, instead of learning a second-order lexicalized model, we propose to jointly learn a second-order unlexicalized model (whose vocabulary consists of POS tags instead of words) and a first-order lexicalized model based on the agreement-based learning framework (Liang et al., 2007). The jointly learned models have a manageable number of grammar rules while still benefiting from both second-order parsing and lexicalization.

We conduct experiments on the Wall Street Journal (WSJ) dataset and seven languages on the Universal Dependencies (UD) dataset. The experimental results demonstrate that our models achieve state-of-the-art accuracies on unsupervised dependency parsing.

## 2 Background

### 2.1 Dependency Model With Valence

The Dependency Model with Valence (DMV) (Klein and Manning, 2004) is a probabilistic generative model of a sentence and its parse tree. It generates a dependency parse tree from the imaginary root node in a recursive top-down manner. There are three types of probabilistic grammar rules in a DMV, namely ROOT, CHILD and DECISION rules, each associated with a set of multinomial distributions $P_{\text{ROOT}}(c)$, $P_{\text{CHILD}}(c|p, dir, val)$ and $P_{\text{DECISION}}(dec|p, dir, val)$, where $p$ is the parent token, $c$ is the child token, $dec$ is the continue/stop decision, $dir$ indicates the direction of generation, and $val$ indicates whether parent $p$ has generated any child in direction $dir$. To generate a sequence of tokens along with its dependency parse tree, the DMV model generates a token $c$ from the ROOT distribution $P_{\text{ROOT}}(c)$ firstly. Then for each token $p$ that has already been generated, it generates a decision from the DECISION distribution $P_{\text{DECISION}}(dec|p, dir, val)$ to determine whether to generate a new child in direction $dir$. If $dec$ is CONTINUE, then a new child $p$ is generated from the CHILD distribution $P_{\text{CHILD}}(c|p, dir, val)$. If $dec$ is STOP, then $p$ stops generating children in direction $dir$. The joint probability of the sequence and its corresponding dependency parse tree can be calculated by taking product of the probabilities of all the generation steps.

### 2.2 Neuralized DMV Models

**Neural DMV** One limitation of the DMV model is that it does not consider the correlation between tokens. Jiang et al. (2016) proposed the Neural DMV (NDMV) model, which uses continuous POS embedding to represent discrete POS tags and calculate rule probabilities through neural networks based on the POS embedding. In this way, the model can learn the correlation between POS tags and smooth grammar rule probabilities accordingly.

**Lexicalized NDMV** Neural DMV is still an unlexicalized model which is based on POS tags and does not use word information. Han et al. (2017) proposed the Lexicalized NDMV (L-NDMV) in which each token is a POS/word pair. The neural network that computes rule probabilities takes both the POS embedding and the word embedding as input. To reduce the vocabulary size, they replace low-frequency words with their POS tags.

## 3 Method

### 3.1 Second-Order Parsing

In our proposed second-order NDMV, we calculate each rule probability based additionally on the information of the sibling or grandparent. We take sibling-NDMV for example to demonstrate the generative story.

- We start with the imaginary root token, generating its only child $c$ with probability $P_{\text{ROOT}}(c)$
- For each token $p$, we decide whether to generate a new child or not with probability $P_{\text{DECISION}}(dec|p, s, dir, val)$, where $s$ is the previous child token generated by $p$ in direction $dir$. If $p$ has not generated any child in direction $dir$ yet, we use a special symbol NULL to represent $s$.
- If decision $dec$ is CONTINUE, $p$ generates a new child $c$ with probability $P_{\text{CHILD}}(c|p, s, dir, val)$. If decision $a$ is STOP, $p$ stops generating children in direction $dir$.

For parsing, we design dynamic programming algorithms adapted from Koo and Collins (2010). Since the grandparent token is deterministic for each token, the parsing algorithm of our grand-NDMV model is similar to theirs. There are two options for determining the sibling token since the generation process of child tokens can be either from the inside out or from the outside in. Koo and Collins (2010) make the inside-out assumption, but in this paper, we make the outside-in assumption because it makes implementation easier and can achieve better performance empirically. We provide the pseudo code of the second-order inside algorithm and the second-order parsing algorithm in the appendix.

## 3.2 Parameterization

In a neural DMV, we compute the probability of a grammar rule using a neural network. Below we formulate the computation of CHILD rule probabilities. The full architecture of the neural network is shown in Figure 1. ROOT and DECISION rule probabilities are computed in a similar way.

In our second-order neural DMV, each CHILD rule $P_{\text{CHILD}}(c|p, s, dir, val)$ involves three tokens: parent $p$, child $c$, and sibling (or grandparent) $s$. Denote the embedding of the parent, child and sibling (or grandparent) by $x_p, x_c, x_s \in \mathcal{R}^d$, which are retrieved from a shared token embedding layer. We use three different linear transformations to produce the representations of a token as a parent, child, and sibling (or grandparent).

$$e_c = W_c x_c \quad e_p = W_p x_p \quad e_s = W_s x_s$$

We feed $e_c, e_p, e_s$ to the same neural network that consists of three consecutive MLPs. The first and second MLPs are used respectively to insert valence and direction information into the representations, and the last MLP is used to produce final hidden representations $h_c, h_p, h_s$ (see the appendix for the complete formulation). We use different parameters of the first and second MLPs for different values of valence $val$ and direction $dir$. We add skip-connections to the first and second MLPs because skip-connections have been found very useful in unsupervised neural parsing (Kim et al., 2019).

We then follow Wang et al. (2019) and use a decomposed trilinear function to compute the unnormalized rule probability from the three vectors $h_c, h_p, h_s$.

$$S(p, s, c) = \sum_{i=1}^{q} o_{p,i} \times o_{s,i} \times o_{c,i}$$

$$o_p = C_p h_p \quad o_c = C_c h_c \quad o_s = C_s h_s$$

where $C_p, C_c, C_s \in R^{q \times d}$ are the parameters of the decomposed trilinear function and $\times$ is scalar multiplication. Then we apply a softmax function to produce the final rule probability.

$$P_{\text{CHILD}}(c|p, s, dir, val) = \frac{e^{S(p,s,c)}}{\sum\limits_{c' \in \mathcal{C}} e^{S(p,s,c')}}$$

where $\mathcal{C}$ is the vocabulary.

## 3.3 Learning

The learning objective function $L(\theta)$ is the log-likelihood of training sentences $X = \{x_1, ..., x_n\}$

$$L(\theta) = \sum_{i=1}^{n} \log p_\theta(x_i) \tag{1}$$

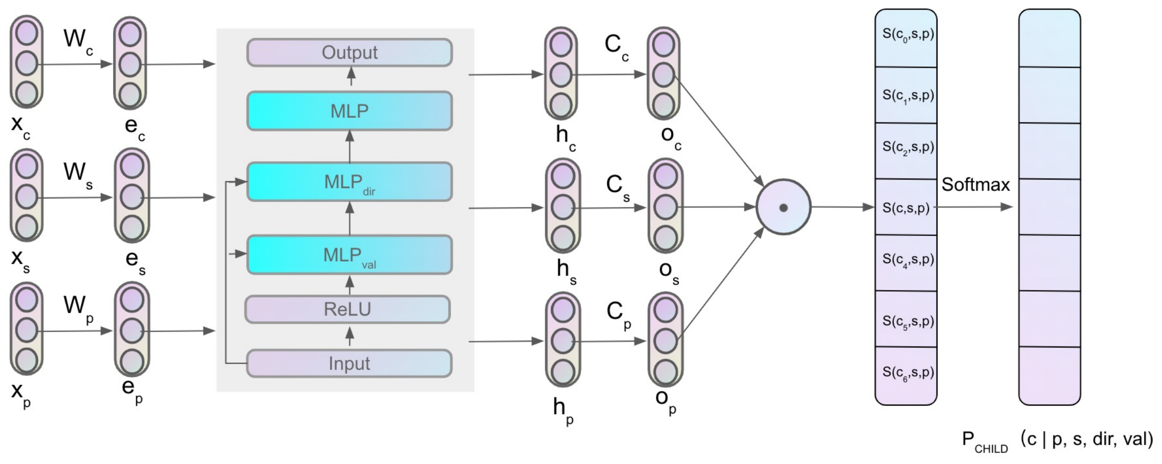

Figure 1: Illustration of our neural architecture.

where $\theta$ is the parameters of the neural networks. The probability of each sentence $x$ is defined as:

$$p_\theta(x) = \sum_{z \in \mathcal{T}(x)} p_\theta(x, z) \tag{2}$$

where $\mathcal{T}(x)$ is the set of all possible dependency parse trees for sentence $x$. We use $c(r, x, z)$ to represent the number of times rule $r$ is used in dependency parse tree $z$ of sentence $x$. Then we have

$$p_\theta(x, z) = \prod_{r \in \mathcal{R}} p_\theta(r)^{c(r,x,z)} \tag{3}$$

where $\mathcal{R}$ is the collection of all DECISION, CHILD and ROOT rules.

**Learning via EM algorithm** We can rewrite the log-likelihood of sentence $x$ as follows.

$$\log p_\theta(x) = E_{q(z)} \left[\log p_\theta(x, z)\right] + H\left[q(z)\right] + KL(q(z) \| p_\theta(z|x)) \tag{4}$$

where $q(z)$ is an arbitrary distribution and $H$ is the entropy function. In the E-step, we fix $\theta$ and set $q(z) = p_\theta(z|x)$. In the M-step, we fix $q(z)$ and update $\theta$ with the following objective:

$$Q(\theta) = E_{q(z)} \left[\log p_\theta(x, z)\right] = E_{q(z)} \left[\sum_{r \in \mathcal{R}} c(r, x, z) \log p_\theta(r)\right] = \sum_{r \in \mathcal{R}} e(r, x) \log p_\theta(r) \tag{5}$$

where $e(r, x)$ is the expected count of grammar rule $r$ in sentence $x$ based on $q(z)$, which can be obtained using the inside-outside algorithm. We can use gradient descent to update $\theta$.

$$\nabla_\theta Q(\theta) = \sum_{r \in \mathcal{R}} e(r, x) \nabla_\theta \log p_\theta(r) \tag{6}$$

**Learning via direct marginal likelihood optimization** We can also use gradient descent to maximize $\log p_\theta(x)$ directly. Based on the derivation of Salakhutdinov et al. (2003)[2], we have

$$\begin{aligned} \nabla_\theta(\log p_\theta(x)) &= \sum_{z \in \mathcal{T}(x)} p_\theta(z|x) \nabla_\theta \log p_\theta(x, z) \\ &= \sum_{z \in \mathcal{T}(x)} p_\theta(z|x) \sum_{r \in \mathcal{R}} c(r, x, z) \nabla_\theta \log p_\theta(r) \\ &= \sum_{r \in \mathcal{R}} e(r, x) \nabla_\theta \log p_\theta(r) \end{aligned} \tag{7}$$

[2]See Equation 8 in Salakhutdinov et al. (2003).

where $e(r,x)$ is the expected count of grammar rule $r$ in sentence $x$ based on $p_\theta(z|x)$. Traditionally, we use the inside-outside algorithm to obtain the expected count $e(r,x)$. Eisner (2016) points out that we can use back-propagation to calculate the expected count $e(r,x)$.

$$e(r,x) = \frac{\partial \log p_\theta(x)}{\partial \log p_\theta(r)} \tag{8}$$

So we only need to use the inside algorithm to calculate $\log p_\theta(x)$ and then use back-propagation to update the parameters directly, without the need for the outside algorithm.

**Mini-batch gradient descent as online EM** In Equation 7, we note that the gradient contains the term $e(r,x)$. If we use mini-batch gradient descent to optimize $\log p_\theta(x)$, it is analogous to the online-EM algorithm (Liang and Klein, 2009). To compute the gradient for each mini-batch, we first need to compute the expected counts from the training sentences in the mini-batch, which is exactly what the online E-step does; we then use the expected counts to compute the gradient and update the model parameters, which is similar to the M-step, except that here we only perform one update step, while in the EM algorithm multiple update steps may be taken based on the same expected counts. According to Liang and Klein (2009), online-EM has a faster convergence speed and can even find a better solution. Empirically, we do find that direct marginal likelihood optimization outperforms the EM algorithm.

## 3.4 Agreement-Based Learning

In our second-order DMV model, the number of grammar rules is $4|V|^3 + 4|V|^2 + |V|$, which is cubic in the vocabulary size $|V|$. When our model is lexicalized, the vocabulary may contain thousands of words or more, making the model size less manageable. Instead of learning a second-order lexicalized model, we propose to jointly learn a second-order unlexicalized model (whose vocabulary consists of POS tags instead of words) and a first-order lexicalized model based on the agreement-based learning framework (Liang et al., 2007). The jointly learned models have a manageable number of grammar rules while still benefiting from both second-order parsing and lexicalization. Empirically, we do find that the jointly trained models outperform lexicalized second-order models.

Following Liang et al. (2007), we define the objective function for our jointly trained first-order L-NDMV and second-order NDMV as

$$\mathcal{O}_{agree}(\theta) \overset{def}{=} \log \sum_{z \in \mathcal{T}(x)} (p_{\theta_0}(x,z) \cdot p_{\theta_1}(x,z)) \tag{9}$$

where $\theta_0$ is parameters of L-NDMV and $\theta_1$ is parameters of second-order NDMV. Intuitively, the objective requires the two models to reach agreement on the probability distribution of dependency parse tree $z$. We use joint decoding (parsing) to predict dependency parse tree $z_{\text{predict}}$ for sentence $x$.

$$z_{\text{predict}} = \text{argmax}_{z \in \mathcal{T}(x)}\, p_{\theta_0}(x,z) \cdot p_{\theta_1}(x,z) \tag{10}$$

The inside and parsing algorithms for jointly trained models can be found in the appendix.

**Learning via product EM algorithm** Liang et al. (2007) propose to optimize the objective using the product EM algorithm based on the following lower bound of the objective.

$$\mathcal{O}_{agree} = \log \sum_z q(z) \frac{p_{\theta_0}(x,z) \cdot p_{\theta_1}(x,z)}{q(z)} \geq E_{q(z)}(\log \frac{p_{\theta_0}(x,z) \cdot p_{\theta_1}(x,z)}{q(z)}) \overset{def}{=} L(\theta, q) \tag{11}$$

The product EM algorithm performs coordinate-wise ascent on $L(\theta,q)$. In the product E-step, we optimize $L(\theta,q)$ with respect to $q$.

$$L(\theta,q) = -KL(q(z) \| p_{\theta_0}(x,z) \cdot p_{\theta_1}(x,z)) + const \tag{12}$$

where $const$ does not depend on $\theta$ and $q$. In the product E-step, the maximum can be obtained by setting $q(z) \propto p_{\theta_0}(x,z) \cdot p_{\theta_1}(x,z)$ to minimize the KL term. In the M-step, we optimize $L(\theta, q)$ with respect to $\theta$.

$$L(\theta, q) = E_q \log p_{\theta_0}(x,z) + E_q \log p_{\theta_1}(x,z) + const \tag{13}$$

where $const$ does not depend on $\theta$. It consists of one term for each model. We update the parameters of each model separately based on the expected counts obtained from the product E-step, which can be calculated through the inside-outside algorithm.

**Learning via direct marginal likelihood optimization** $\mathcal{O}_{agree}$ can be calculated through the inside algorithm. Similar to what we describe in Section 3.3, we can benefit from both agreement-based learning and the online-EM algorithm if we use gradient descent directly to optimize $\mathcal{O}_{agree}$ instead of using the product EM algorithm.

## 4 Experiment

### 4.1 Datasets and Setting

**English Penn Treebank** We conduct experiments on the Wall Street Journal (WSJ) corpus, with section 2-21 for training, section 22 for validation and section 23 for testing. We use sentences of length $\leq$ 10 in training and use sentences of length $\leq$ 10 (WSJ10) and all sentences (WSJ) in testing.

**Universal Dependency Treebank** Following the setting of Li et al. (2018) and Han et al. (2019), we conduct experiments on selected languages from the Universal Dependency Treebank 1.4 (Nivre et al., 2016). We use sentences of length $\leq$ 15 in training and sentences of length $\leq$ 15 and $\leq$ 40 in testing.

**Setting** On the WSJ dataset, for fair comparison, we follow Han et al. (2017) and Han et al. (2019) and use HDP-DEP (Naseem et al., 2010) to initialize our models. Specifically, we train the unsupervised HDP-DEP model on WSJ, use it to parse the training corpus, and then use the predicted parse trees to perform supervised learning of our model for several epochs. On the UD dataset, we use the K&M initialization (Klein and Manning, 2004). We use direct marginal likelihood optimization (DMO) as the training method and use Adam (Kingma and Ba, 2015) as the optimizer with learning rate 0.001. The batch size is set to 64 for WSJ and 100 for UD. The hyperparameters of the neural networks, the setting of L-NDMV and more details can be found in the appendix. We apply early stopping based on the log-likelihood of the development data and report the mean accuracy over 5 random restarts.

### 4.2 Result

**Result on WSJ** In Table 1, we compare our methods with previous unsupervised dependency parsers. Our sibling-NDMV model can outperform the previous state-of-the-art parser by 1.9 points on WSJ10 and 3.1 points on WSJ in the unlexicalized setting. Our lexicalized sibling-NDMV achieves further improvement over the unlexicalized sibling-NDMV. On the other hand, our grand-NDMV performs significantly worse than the sibling-NDMV and lexicalization hurts its performance. Why grandparent information is less useful than sibling information in unsupervised parsing is an intriguing question that we leave for feature research. Joint training with a first-order L-NDMV can increase the performance of unlexicalized sibling-NDMV from 77.5 to 79.9 and that of unlexicalized grand-NDMV from 71.4 to 76.0 on WSJ10. The jointly trained models also outperform the lexicalized second-order models.

**Result on UD** In Table 2, we first compare our models with models which do not use the universal linguistic prior (UP)[3]. The variational variant of D-NDMV (Han et al., 2019) is the recent state-of-the-art model without UP. Our method outperforms theirs on six of the eight languages and also on average. We then compare our second-order models with recent state-of-the-art discriminative models, which rely heavily on the universal linguistic prior to achieve good performance (for example, Li et al. (2018) reported bad results if they do not use the universal linguistic prior). We find that sibling-NDMV

[3]The universal linguistic prior is a set of syntactic dependencies that are common in many languages, proposed by Naseem et al. (2010)

| METHODS | WSJ10 | WSJ |
|---|---|---|
| DMV(Klein and Manning, 2004) | 58.3 | 39.4 |
| LN (Cohen et al., 2009) | 59.4 | 40.5 |
| Convex-MST (Grave and Elhadad, 2015) | 60.8 | 48.6 |
| Shared LN (Cohen and Smith, 2009) | 61.3 | 41.4 |
| Feature DMV (Berg-Kirkpatrick et al., 2010) | 63.0 | - |
| PR-S (Gillenwater et al., 2011) | 64.3 | 53.3 |
| E-DMV (Headden et al., 2009) | 65.0 | - |
| TSG-DMV (Blunsom and Cohn, 2010) | 65.9 | 53.1 |
| UR-A E-DMV (Tu and Honavar, 2012) | 71.4 | 57.0 |
| CRFAE (Cai et al., 2017) | 71.7 | 55.7 |
| Neural DMV (Jiang et al., 2016) | 72.5 | 57.6 |
| HDP-DEP (Naseem et al., 2010) | 73.8 | - |
| NVTP (Li et al., 2018) | 54.7 | 37.8 |
| Variational variant D-NDMV * (Han et al., 2019) | 75.5 | 60.4 |
| Deterministic variant D-NDMV * (Han et al., 2019) | 75.6 | 61.4 |
| L-NDMV * (Han et al., 2017) | 75.1 | 59.5 |
| grand-NDMV * | 71.4 | 57.3 |
| sibling-NDMV * | 77.5 | 64.5 |
| Lexicalized grand-NDMV * | 63.0 | 52.6 |
| Lexicalized sibling-NDMV * | 78.3 | 66.4 |
| Joint training: grand-NDMV + L-NDMV * | 76.0 | 64.3 |
| Joint training: sibling-NDMV + L-NDMV * | **79.9** | **67.5** |
| Systems with Additional Training Data (for reference) | | |
| CS (Spitkovsky et al., 2013) | 72.0 | 64.4 |
| MaxEnc (Le and Zuidema, 2015) | 73.2 | 65.8 |
| L-NDMV * (Han et al., 2017) | 77.2 | 63.2 |

Table 1: Comparison on WSJ. *: Approaches which use Naseem et al. (2010) for initialization.

can outperform these discriminative models while grand-NDMV can achieve comparable results, even though we do not utilize the universal linguistic prior.

## 5 Analysis

### 5.1 Effect of Skip-Connections

From Table 3 and 4, we find that using skip-connections can achieve higher log-likelihood and better parsing accuracy in most cases. On UD, the performance is much better when using skip-connections except on Basque.

### 5.2 Comparison of Training Methods

In Table 3, we find that the EM algorithm significantly underperforms DMO. On the other hand, Table 4 shows that the EM algorithm performs comparably to DMO on WSJ.

We also compare the learning curves of these two methods. For fair comparison, we use the same batch-size for both methods. First we conduct an experiment using the joint L-NDMV and sibling-NDMV model on WSJ. In Figure 2, we find that DMO converges to a higher log-likelihood compared with EM and the convergence speed is roughly the same. In Figure 3, we find DMO can find a slightly better model compared with EM. Second, we conduct an experiment using sibling-NDMV model on the UD French dataset. In Figure 4, we find DMO converges faster than EM and converges to a higher log-likelihood. In Figure 4, we find that the model accuracy of DMO is much higher than that of EM at the beginning, but it drops significantly after epoch 23, suggesting that early-stop is necessary. We also find similar phenomena for other languages on UD.

It should be noted that we use HDP-DEP (Naseem et al., 2010) for initialization on WSJ and use K&M initialization (Klein and Manning, 2004) on UD. We see that HDP-DEP initialization leads to a very high initial UAS of 75% (Figure 3), while K&M initialization leads to a low initial UAS of 38.5% (Figure 5). It can be seen that EM is more sensitive to the initialization while DMO can achieve good results even if the initialization is bad.

| | NO UP | | | | | | +UP | |
|---|---|---|---|---|---|---|---|---|
| | NDMV | LD | DV | VV | +sibling | +grand | NVTP | CM |
| | Length ≤ 15 | | | | | | | |
| Basque | 48.3 | 47.9 | 40.6 | 42.7 | 30.8 | 34.1 | **52.9** | 52.5 |
| Dutch | 44.1 | 35.5 | 42.1 | 43.0 | **46.5** | 42.4 | 39.6 | 43.4 |
| French | 59.5 | 52.1 | 59.0 | 61.7 | **65.7** | 62.9 | 59.9 | 61.6 |
| German | 56.2 | 51.9 | 56.4 | 58.5 | **63.6** | 48.9 | 57.5 | 66.7 |
| Italian | 72.7 | 73.1 | 59.6 | 63.5 | **76.3** | 76.3 | 59.7 | 73.2 |
| Polish | 72.7 | 66.2 | 70.5 | 75.8 | **77.1** | 63.7 | 57.1 | 66.7 |
| Portuguese | 45.5 | **70.5** | 68.8 | 69.1 | 67.8 | 62.5 | 52.7 | 60.7 |
| Spanish | 38.1 | 65.5 | 63.8 | 66.1 | 67.2 | **67.5** | 55.6 | 61.6 |
| **Average** | 53.3 | 57.8 | 57.6 | 60.1 | **61.9** | 57.2 | 54.4 | 59.3 |
| | Length ≤ 40 | | | | | | | |
| Basque | 47.8 | 45.4 | 39.9 | 42.4 | 30.5 | 33.0 | 48.9 | **50.0** |
| Dutch | 35.6 | 34.1 | 42.4 | 43.7 | 45.0 | 43.7 | 42.5 | **45.3** |
| French | 38.1 | 48.6 | 57.2 | 58.5 | **64.9** | 61.4 | 55.4 | 62.0 |
| German | 50.4 | 50.5 | 54.5 | 52.9 | **61.5** | 46.7 | 54.2 | 51.4 |
| Italian | 63.6 | 71.1 | 60.2 | 61.3 | **71.8** | 69.7 | 55.7 | 69.1 |
| Polish | 62.8 | 63.7 | 66.7 | 73.0 | **75.0** | 62.4 | 51.7 | 63.4 |
| Portuguese | 49.0 | **67.2** | 64.7 | 65.7 | 63.7 | 60.3 | 45.3 | 57.1 |
| Spanish | 58.0 | 61.9 | 64.3 | 64.4 | **66.8** | 65.0 | 52.5 | 61.9 |
| **Average** | 50.7 | 55.3 | 56.2 | 57.7 | **59.9** | 55.2 | 50.8 | 57.5 |

Table 2: Comparison on Universal Dependency Treebanks. No UP: System using the universal linguistic prior. +UP : Systems using the universal linguistic prior. LD: LC-DMV (Noji and Miyao, 2015). DV,VV: The deterministic and variational variants of D-NDMV (Han et al., 2019). +sibling: Our second-order sibling-NDMV. +grand: Our second-order grand-NDMV. NVTP: Neural variational transition-based parser (Li et al., 2018). CM: Convex-MST (Grave and Elhadad, 2015).

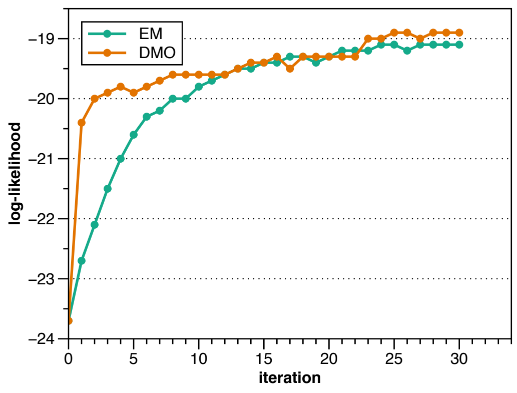

Figure 2: Comparison of training methods on log-likelihood for WSJ.

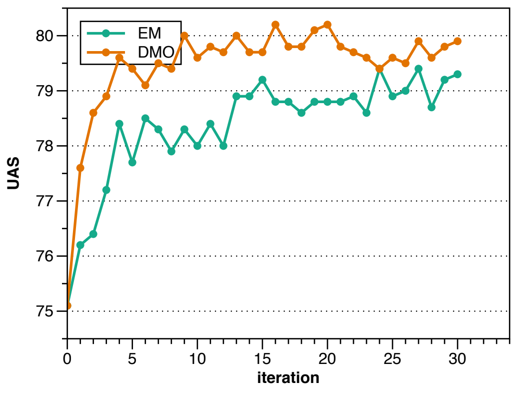

Figure 3: Comparison of training methods on UAS for WSJ.

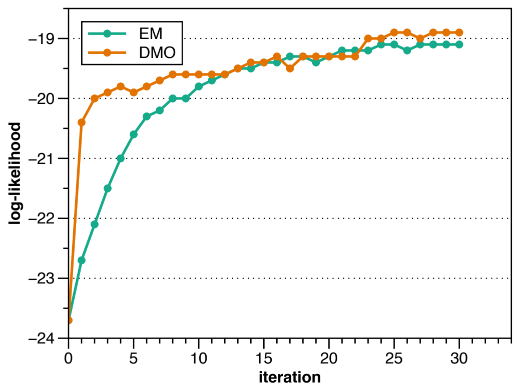

Figure 4: Comparison of training methods on log-likelihood for UD (French).

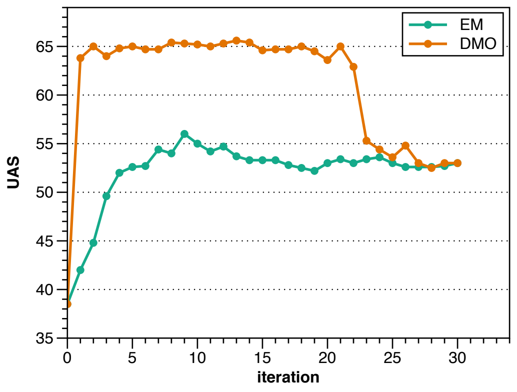

Figure 5: Comparison of training methods on UAS for UD (French).

| LANGUAGE | LOG-LIKELIHOOD | | | UAS ( $\leq 15$ / $\leq 40$) | | |
|---|---|---|---|---|---|---|
| | DMO | | EM | DMO | | EM |
| | w. SC | w.o. SC | w. SC | w. SC | w.o. SC | w. SC |
| Basque | -19.3 | -19.7 | -20.0 | 30.8/30.5 | 44.7/44.3 | 28.8/29.1 |
| Dutch | -19.1 | -19.4 | -19.3 | 46.5/45.0 | 34.9/34.8 | 43.6/41.8 |
| French | -19.8 | -20.9 | -19.4 | 65.7/64.9 | 47.2/50.1 | 55.0/54.5 |
| German | -20.9 | -22.4 | -21.2 | 63.6/61.5 | 36.4/37.8 | 57.9/54.5 |
| Italian | -16.7 | -16.9 | -16.7 | 76.3/71.8 | 65.6/58.1 | 71.0/65.2 |
| Polish | -16.2 | -16.9 | -17.0 | 77.1/75.0 | 66.2/63.7 | 61.7/60.4 |
| Portugese | -17.9 | -18.4 | -17.8 | 67.8/63.7 | 60.0/59.1 | 57.7/51.9 |
| Spanish | -20.8 | -21.7 | -20.9 | 67.2/66.8 | 49.6/48.8 | 61.4/58.1 |

Table 3: Effect of skip-connections and training methods on UD.

| MODELS | LOG-LIKELIHOOD | | | UAS ( WSJ10 / WSJ) | | |
|---|---|---|---|---|---|---|
| | DMO | | EM | DMO | | EM |
| | w. SC | w.o. SC | w. SC | w. SC | w.o. SC | w. SC |
| sibling-NDMV | -17.3 | -17.7 | -17.3 | 77.5/64.5 | 75.4/63.1 | 77.7/64.5 |
| grand-NDMV | -16.9 | -17.4 | -17.0 | 71.4/57.3 | 68.4/55.1 | 72.7/61.8 |
| sibling-NDMV + L-NDMV | -53.2 | -55.3 | -53.5 | 79.9/67.5 | 78.2/64.7 | 79.1/66.5 |
| grand-NDMV + L-NDMV | -53.8 | -57.5 | -54.2 | 76.0/64.3 | 73.7/60.7 | 75.2/65.3 |

Table 4: Effect of skip-connections and training methods on WSJ.

## 5.3 Effect of Joint Training and Parsing

In Table 5, we compare the performance with different training and parsing settings. We find that joint parsing is better than separate parsing in both training settings. With joint training, each individual model can achieve better performance compared with separate training, which shows the effectiveness of agreement-based joint learning.

## 5.4 Limitations

Our second-order NDMV model is more sensitive to the initialization compared with the first-order NDMV model. We fail to produce a good result under the K&M initialization on WSJ: only 58.5% UAS for sibling-NDMV on WSJ10, while the first-order NDMV model can achieve 69.7% UAS. We rely on the parsing result of HDP-DEP to initialize our model in order to reach the state-of-the-art result on WSJ. This is similar to the case of L-NDMV, which performs badly when using the K&M initialization according to Han et al. (2017). Because of the bad performance of L-NDMV with the K&M initialization as well as the time constraint that prevents us from running HDP-DEP on UD, we did not conduct experiments of agreement-based learning with L-NDMV on the UD datasets. We leave this for future work.

Our second-order model is also quite sensitive to the design of the neural architecture, which is similar to case of unsupervised constituency parsing reported by Kim et al. (2019). We also try the third-order NDMV model (grand-sibling or tri-sibling) but are not able to get better results compared with sibling-NDMV.

Our second-order parsing algorithm has a theoretical time complexity of $O(n^4)$, which is higher than the time complexity of $O(n)$ of transition-based unsupervised parsers (Li et al., 2018) and the time complexity of $O(n^3)$ of first-order NDMV models, where $n$ is the sentence length. However, transition-based parsers are hard to batchify, while our model can be parallelized efficiently following the methods introduced by Torch-Struct (Rush, 2020). In practice, our second-order parser runs very fast on GPU, requiring only several minutes to train.

| Training method | UAS (WSJ10 / WSJ) | | |
|---|---|---|---|
| | L-NDMV | sibling-NDMV | joint parsing |
| separate training | 76.6 / 62.7 | 77.5 / 64.8 | 78.4 / 65.8 |
| joint training | 79.2 / 65.4 | 78.7 / 65.6 | 79.9 / 67.5 |

Table 5: The effect of joint training and joint parsing

## 6 Conclusion

We propose second-order NDMV models, which incorporate sibling or grandparent information. We find that sibling information is very useful in unsupervised dependency parsing. We use agreement-based learning to combine the benefits of second-order parsing and lexicalization, achieving state-of-the-art results on the WSJ dataset. We also show the effectiveness of our neural parameterization architecture with skip-connections and the direct marginal likelihood optimization method.

## Acknowledgement

This work was supported by the National Natural Science Foundation of China (61976139).

## References

Taylor Berg-Kirkpatrick, Alexandre Bouchard-Côté, John DeNero, and Dan Klein. 2010. Painless unsupervised learning with features. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 582–590, Los Angeles, California, June. Association for Computational Linguistics.

Phil Blunsom and Trevor Cohn. 2010. Unsupervised induction of tree substitution grammars for dependency parsing. In *EMNLP*.

Jiong Cai, Yong Jiang, and Kewei Tu. 2017. Crf autoencoder for unsupervised dependency parsing. In *EMNLP*.

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

Shay B Cohen, Kevin Gimpel, and Noah A Smith. 2009. Logistic normal priors for unsupervised probabilistic grammar induction. In *Advances in Neural Information Processing Systems*, pages 321–328.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Jason Eisner. 2016. Inside-outside and forward-backward algorithms are just backprop (tutorial paper). In *SPNLP@EMNLP*.

Jennifer Gillenwater, Kuzman Ganchev, João Graça, Fernando C Pereira, and Ben Taskar. 2011. Posterior sparsity in unsupervised dependency parsing. *J. Mach. Learn. Res.*, 12:455–490.

Edouard Grave and Noémie Elhadad. 2015. A convex and feature-rich discriminative approach to dependency grammar induction. In *ACL*.

Wenjuan Han, Yong Jiang, and Kewei Tu. 2017. Dependency grammar induction with neural lexicalization and big training data. In *EMNLP*.

Wenjuan Han, Yong Jiang, and Kewei Tu. 2019. Enhancing unsupervised generative dependency parser with contextual information. In *ACL*.

William P. Headden, Mark Johnson, and David McClosky. 2009. Improving unsupervised dependency parsing with richer contexts and smoothing. In *HLT-NAACL*.

Yong Jiang, Wenjuan Han, and Kewei Tu. 2016. Unsupervised neural dependency parsing. In *EMNLP*.

Yoon Kim, Chris Dyer, and Alexander M. Rush. 2019. Compound probabilistic context-free grammars for grammar induction. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 2369–2385. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Dan Klein and Christopher D. Manning. 2004. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *ACL*.

Terry K Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *ACL*.

Phong Le and Willem Zuidema. 2015. Unsupervised dependency parsing: Let's use supervised parsers. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 651–661, Denver, Colorado, May–June. Association for Computational Linguistics.

Bowen Li, Jianpeng Cheng, Yang Liu, and Frank Keller. 2018. Dependency grammar induction with a neural variational transition-based parser. In *AAAI*.

Percy Liang and Dan Klein. 2009. Online em for unsupervised models. In *HLT-NAACL*.

Percy Liang, Dan Klein, and Michael I. Jordan. 2007. Agreement-based learning. In *NIPS*.

Xuezhe Ma and Zhao Hai. 2012. Fourth-order dependency parsing. In *COLING*.

Tahira Naseem, Harr Chen, Regina Barzilay, and Mark Johnson. 2010. Using universal linguistic knowledge to guide grammar induction. In *EMNLP*.

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 *LREC*.

Hiroshi Noji and Yusuke Miyao. 2015. Left-corner parsing for dependency grammar. *Journal of Information Processing*, 22:251–288.

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, July. Association for Computational Linguistics.

Ruslan Salakhutdinov, Sam T. Roweis, and Zoubin Ghahramani. 2003. Optimization with em and expectation-conjugate-gradient. In *ICML*.

Valentin I. Spitkovsky, Hiyan Alshawi, and Dan Jurafsky. 2013. Breaking out of local optima with count transforms and model recombination: A study in grammar induction. In *EMNLP*.

Ke M. Tran, Yonatan Bisk, Ashish Vaswani, Daniel Marcu, and Kevin Knight. 2016. Unsupervised neural hidden Markov models. In *Proceedings of the Workshop on Structured Prediction for NLP*, pages 63–71, Austin, TX, November. Association for Computational Linguistics.

Kewei Tu and Vasant G Honavar. 2012. Unambiguity regularization for unsupervised learning of probabilistic grammars. In *EMNLP-CoNLL*.

Xinyu Wang, Jingxian Huang, and Kewei Tu. 2019. Second-order semantic dependency parsing with end-to-end neural networks. In *ACL*.

Yu Zhang, Zhenghua Li, and Min Zhang. 2020. Efficient second-order treecrf for neural dependency parsing. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 3295–3305. Association for Computational Linguistics.

## A Appendix

### A.1 Inside Algorithm and Parsing Algorithm

We use the dynamic programming substructure proposed for second-order supervised dependency parsing. For grandparent-child model, Koo and Collins (2010) augment both complete and incomplete spans with grandparent indices. They called the augmented span g-spans. Formally, they denote a complete g-span as $C^{g}_{h,e}$, where $C_{h,e}$ is a normal complete span in the Eisner algorithm, g is the grandparent's index, with the implication that $(g, h)$ is a dependency. Incomplete g-span is defined similarly.

For second-order NDMV, we further augment incomplete and complete g-spans with valence information. We distinguish the direction of span explicitly, denoting our augmented complete v-span as $C^{g,v}_{h,e,d}$, where $d$ is the direction, $v$ is the valence, $h$ is the start index and $e$ is the end index of span compared with g-span. Incomplete v-span is defined similarly.

For grand-NDMV, given sentence $x$, we suppose that $x_0$ is the imaginary root token and $x_1, ..x_n$ are tokens. We denote $D[i, g, d, v, a] = \log(\mathrm{p}_{\mathrm{DECISION}}(\mathrm{decision} = \mathrm{a}|\mathrm{parent} = \mathrm{x_i}, \mathrm{grand} = \mathrm{x_g}, \mathrm{direction} =$

$\mathrm{d}, \mathrm{valence} = \mathrm{v})$), $S[i,c,g,d,v] = \log(\mathrm{p_{CHILD}}(\mathrm{child} = \mathrm{x_c}|\mathrm{parent} = \mathrm{x_i}, \mathrm{grand} = \mathrm{x_g}, \mathrm{direction} = \mathrm{d}, \mathrm{valence} = \mathrm{v}))$, and $R[i] = \log(\mathrm{p_{ROOT}}(\mathrm{child} = \mathrm{x_i}))$. Given these definitions, the inside algorithm of grand-NDMV is shown in Algorithm 1.

For sibling-NDMV, $g$ in $C^{g}_{h,e}$ stands for the index of sibling instead of the index of grandparent. Given sentence $x$, we suppose that $x_0$ is a special NULL token which stands for no sibling and $x_1, ..x_n$ are tokens. We denote $D[i,g,d,v,a] = \log(\mathrm{p_{DECISION}}(\mathrm{decision} = \mathrm{a}|\mathrm{parent} = \mathrm{x_i}, \mathrm{sibling} = \mathrm{x_g}, \mathrm{direction} = \mathrm{d}, \mathrm{valence} = \mathrm{v}))$, $S[i,c,g,d,v] = \log(\mathrm{p_{CHILD}}(\mathrm{child} = \mathrm{x_c}|\mathrm{parent} = \mathrm{x_i}, \mathrm{sibling} = \mathrm{x_g}, \mathrm{direction} = \mathrm{d}, \mathrm{valence} = \mathrm{v}))$, and $R[i] = \log(\mathrm{p_{ROOT}}(\mathrm{child} = \mathrm{x_i}))$. Given these definitions, the inside algorithm of sibling-NDMV is shown in Algorithm 2.

For jointly trained L-NDMV and second-order NDMV model, we take jointly trained L-NDMV sibing-NDMV for example. We denote $D'[i,d,v,a] = \log(\mathrm{p_{DECISION}}(\mathrm{decision} = \mathrm{a}|\mathrm{parent} = \mathrm{x'_i}, \mathrm{direction} = \mathrm{d}, \mathrm{valence} = \mathrm{v})), \mathrm{S'[i,c,d,v]} = \log(\mathrm{p_{CHILD}}(\mathrm{child} = \mathrm{x'_c}|\mathrm{parent} = \mathrm{x'_i}, \mathrm{direction} = \mathrm{d}, \mathrm{valence} = \mathrm{v})), \mathrm{R'[i]} = \log(\mathrm{p_{ROOT}}(\mathrm{child} = \mathrm{x'_i}))$ for L-NDMV where $x'$ is the sequence of word/POS pairs which starts indexing at 1. The inside algorithm of jointly trained L-NDMV and sibling-NDMV model is shown in Algorithm 3.

Following Eisner (2016), we use back-propagation to obtain expected counts of grammar rules. For the parsing algorithm, we can replace logsumexp with max in Algorithm 1, 2 and 3 to get the Viterbi log-likelihood of the sentence, then use back-propagation to get grammar rules which are used in the Viterbi parse tree, and finally reconstruct the parse tree based on these rules.

### A.2 Full Parameterization

Denote the embedding of the parent, child and sibling (or grandparent) by $x_p, x_c, x_s \in \mathcal{R}^d$. We use three different linear transformations to produce the representations of each token as a parent, child, and sibling (or grandparent).

$$e_c = W_c x_c \quad e_p = W_p x_p \quad e_s = W_s x_s$$

We feed $e_s, e_c, e_p$ to the same neural network which consists of three MLP with skip-connection layer. The first MLP aims at encoding valence information:

$$v = ReLU(W_1(ReLU(W_{val}e + e)))$$

where $val \in [\mathrm{HASCHILD}, \mathrm{NOCHILD}]$.

The second MLP aims at encoding direction information:

$$d = W_3(ReLU(W_2(W_{dir}v + e))$$

where $dir \in [\mathrm{LEFT}, \mathrm{RIGHT}]$

We use the final MLP to get final hidden representation $h$:

$$h = ReLU(W_4 d)$$

For the UD dataset, we use a more expressive MLP to get the final hidden representation $h$ since we find that the UD dataset is more difficult to train.

$$h = ReLU(W_6(ReLU(W_5(ReLU(W_4 d + e)))))$$

For decision rules, we introduce two embedding $x_{\text{stop}}$ and $x_{\text{continue}}$. We feed $x_{\text{stop}}$ and $x_{\text{continue}}$ to a fully connected layer $W_{\text{dec}}$ to get $e_{\text{stop}}$ and $e_{\text{continue}}$. We use the same neural network to get hidden representation $h_{\text{stop}}$ and $h_{\text{continue}}$. For root rules, we introduce $x_{\text{root}}$. We feed $x_{\text{root}}$ to a fully connected layer $W_{\text{root}}$ to get $e_{\text{root}}$. Also, we use the same neural network to get hidden representation $h_{\text{root}}$. We use different decomposed trilinear function parameters for different types of rules. The calculation of the decision rule probability and root rule probability is similar to that of the child rule probability.

**Algorithm 1:** Inside algorithm for grand-NDMV

```
notation LEFT=0, RIGHT=0, HASCHILD=0, NOCHILD=1
  initialization $\forall i, g \quad C^{g,0}_{i,i,0} = D[i,g,0,0,0]$ $C^{g,1}_{i,i,0} = D[i,g,0,1,0]$ $C^{g,0}_{i,i,1} = D[i,g,1,0,0]$ $C^{g,1}_{i,i,1} = D[i,g,1,1,0]$
  for $w = 1...(n-1)$
      for $i = 1...(n-w)$
        $j = i + w$
        for $g < i$ or $g > j$
          $I^{g,0}_{i,j,0} = \mathbf{logsumexp}_{i \le r < j} \left\{ C^{j,1}_{i,r,1} + C^{g,0}_{r+1,j,0} + D[j,g,0,0,1] + S[j,i,g,0,0] \right\}$
          $I^{g,1}_{i,j,0} = \mathbf{logsumexp}_{i \le r < j} \left\{ C^{j,1}_{i,r,1} + C^{g,0}_{r+1,j,0} + D[j,g,0,1,1] + S[j,i,g,0,1] \right\}$
          $I^{g,0}_{i,j,1} = \mathbf{logsumexp}_{i \le r < j} \left\{ C^{g,0}_{i,r,1} + C^{i,1}_{r+1,j,0} + D[i,g,1,0,1] + S[i,j,g,1,0] \right\}$
          $I^{g,1}_{i,j,1} = \mathbf{logsumexp}_{i \le r < j} \left\{ C^{g,0}_{i,r,1} + C^{i,1}_{r+1,j,0} + D[i,g,1,1,1] + S[i,j,g,1,1] \right\}$
          $C^{g,0}_{i,j,0} = \mathbf{logsumexp}_{i \le r \le j} \left\{ C^{j,1}_{i,r,0} + I^{g,0}_{r,j,0} \right\}$
          $C^{g,1}_{i,j,0} = \mathbf{logsumexp}_{i \le r \le j} \left\{ C^{j,1}_{i,r,0} + I^{g,1}_{r,j,0} \right\}$
          $C^{g,0}_{i,j,1} = \mathbf{logsumexp}_{i \le r \le j} \left\{ I^{g,0}_{i,r,1} + C^{i,1}_{r,j,1} \right\}$
          $C^{g,1}_{i,j,1} = \mathbf{logsumexp}_{i \le r \le j} \left\{ I^{g,1}_{i,r,1} + C^{i,1}_{r,j,1} \right\}$
  for $i = 1..(n)$
      $P[i] = R[i] + C^{0,1}_{1,i,0}$
$P = \mathbf{logsumexp}_{1 \le i \le n} P[i] + C^{0,1}_{i,n,1}$
  return P
```

## A.3 Hyperparameters

We set the dimension of POS embedding to 100. The dimension of all linear layers to calculate hidden representation is set to 100. We set the size of decomposed trilinear function parameters to 30 for child and root rules and 10 for decision rules in the unlexicalized setting.

For the lexicalized model, we set the dimension of word embedding to 100. We concatenate the POS embedding and word embedding as input. The dimension of all linear layers to calculate hidden representation is set to 200. We set the size of decomposed trilinear function parameters to 150 for child and root rules and 50 for decision rules. We use an additional dropout layer after the embedding layer to avoid over-fitting since the vocabulary size of the lexicalized model is much larger compared to the unlexicalized model. The dropout rate is set to 0.5.

## A.4 Setting of L-NDMV

The vocabulary consists of word/POS pairs that appear for at least two times in the WSJ10 dataset. We use random embedding to initialize the POS embedding and FastText embedding to initialize the word embedding, which is different from the setting in the original paper (Han et al., 2017). We train FastText on the whole WSJ dataset for 100 epochs with window size 3 and embedding dimension 100.

**Algorithm 2:** Inside algorithm for sibling-NDMV

```
notation LEFT=0, RIGHT=0, HASCHILD=0, NOCHILD=1
  initialization $\forall i, g \quad C^{g,0}_{i,i,0} = D[i,g,0,0,0]$ $C^{g,1}_{i,i,0} = D[i,g,0,1,0]$ $C^{g,0}_{i,i,1} = D[i,g,1,0,0]$ $C^{g,1}_{i,i,1} = D[i,g,1,1,0]$
  for $w = 1...(n-1)$
      for $i = 1...(n-w)$
        $j = i + w$
        for $g < i$ or $g > j$
          $I^{g,0}_{i,j,0} = \mathbf{logsumexp}_{i \le r < j}\left\{C^{0,1}_{i,r,1} + C^{i,0}_{r+1,j,0} + D[j,g,0,0,1] + S[j,i,g,0,0]\right\}$
          $I^{g,1}_{i,j,0} = \mathbf{logsumexp}_{i \le r < j}\left\{C^{0,1}_{i,r,1} + C^{i,0}_{r+1,j,0} + D[j,g,0,1,1] + S[j,i,g,0,1]\right\}$
          $I^{g,0}_{i,j,1} = \mathbf{logsumexp}_{i \le r < j}\left\{C^{j,0}_{i,r,1} + C^{0,1}_{r+1,j,0} + D[i,g,1,0,1] + S[i,j,g,1,0]\right\}$
          $I^{g,1}_{i,j,1} = \mathbf{logsumexp}_{i \le r < j}\left\{C^{j,0}_{i,r,1} + C^{0,1}_{r+1,j,0} + D[i,g,1,1,1] + S[i,j,g,1,1]\right\}$
          $C^{g,0}_{i,j,0} = \mathbf{logsumexp}_{i \le r \le j}\left\{C^{0,1}_{i,r,0} + I^{i,0}_{r,j,0}\right\}$
          $C^{g,1}_{i,j,0} = \mathbf{logsumexp}_{i \le r \le j}\left\{C^{0,1}_{i,r,0} + I^{i,1}_{r,j,0}\right\}$
          $C^{g,0}_{i,j,1} = \mathbf{logsumexp}_{i \le r \le j}\left\{I^{j,0}_{i,r,1} + C^{0,1}_{r,j,1}\right\}$
          $C^{g,1}_{i,j,1} = \mathbf{logsumexp}_{i \le r \le j}\left\{I^{j,1}_{i,r,1} + C^{0,1}_{r,j,1}\right\}$
  for $i = 1..(n)$
      $P[i] = R[i] + C^{0,1}_{1,i,0}$
  $P = \mathbf{logsumexp}_{1 \le i \le n} P[i] + C^{0,1}_{i,n,1}$
  return P
```

**Algorithm 3:** Inside algorithm for joint L-NDMV and sibling-NDMV

```
notation LEFT=0, RIGHT=0, HASCHILD=0, NOCHILD=1
  initialization $\forall i, g \quad C^{g,0}_{i,i,0} = D[i,g,0,0,0] + D'[i,0,0,0]$ $C^{g,1}_{i,i,0} = D[i,g,0,1,0] + D'[i,0,1,0]$
  $C^{g,0}_{i,i,1} = D[i,g,1,0,0] + D'[i,1,0,0]$ $C^{g,1}_{i,i,1} = D[i,g,1,1,0] + D'[i,1,1,0]$
  for $w = 1...(n-1)$
      for $i = 1...(n-w)$
        $j = i + w$
        for $g < i$ or $g > j$
          $I^{g,0}_{i,j,0} =$
  $\mathbf{logsumexp}_{i \le r < j}\left\{C^{0,1}_{i,r,1} + C^{i,0}_{r+1,j,0} + D[j,g,0,0,1] + D'[j,0,0,1] + S[j,i,g,0,0] + S'[j,i,0,0]\right\}$
          $I^{g,1}_{i,j,0} =$
  $\mathbf{logsumexp}_{i \le r < j}\left\{C^{0,1}_{i,r,1} + C^{i,0}_{r+1,j,0} + D[j,g,0,1,1] + D'[j,0,1,1] + S[j,i,g,0,1] + S'[j,i,0,1]\right\}$
          $I^{g,0}_{i,j,1} =$
  $\mathbf{logsumexp}_{i \le r < j}\left\{C^{j,0}_{i,r,1} + C^{0,1}_{r+1,j,0} + D[i,g,1,0,1] + D'[i,1,0,1] + S[i,j,g,1,0] + S'[i,j,1,0]\right\}$
          $I^{g,1}_{i,j,1} =$
  $\mathbf{logsumexp}_{i \le r < j}\left\{C^{j,0}_{i,r,1} + C^{0,1}_{r+1,j,0} + D[i,g,1,1,1] + D'[i,1,1,1] + S[i,j,g,1,1] + S'[i,j,1,1]\right\}$
          $C^{g,0}_{i,j,0} = \mathbf{logsumexp}_{i \le r \le j}\left\{C^{0,1}_{i,r,0} + I^{i,0}_{r,j,0}\right\}$
          $C^{g,1}_{i,j,0} = \mathbf{logsumexp}_{i \le r \le j}\left\{C^{0,1}_{i,r,0} + I^{i,1}_{r,j,0}\right\}$
          $C^{g,0}_{i,j,1} = \mathbf{logsumexp}_{i \le r \le j}\left\{I^{j,0}_{i,r,1} + C^{0,1}_{r,j,1}\right\}$
          $C^{g,1}_{i,j,1} = \mathbf{logsumexp}_{i \le r \le j}\left\{I^{j,1}_{i,r,1} + C^{0,1}_{r,j,1}\right\}$
  for $i = 1..(n)$
      $P[i] = R[i] + R'[i] + C^{0,1}_{1,i,0}$
  $P = \mathbf{logsumexp}_{1 \le i \le n} P[i] + C^{0,1}_{i,n,1}$
  return P
```