StructFormer: Joint Unsupervised Induction of Dependency and Constituency Structure from Masked Language Modeling

Yikang Shen[*]
Mila/Université de Montréal

Yi Tay
Google Research

Che Zheng
Google Research

Dara Bahri
Google Research

Donald Metzler
Google Research

Aaron Courville
Mila/Université de Montréal

Abstract

There are two major classes of natural language grammars — the dependency grammar that models one-to-one correspondences between words and the constituency grammar that models the assembly of one or several corresponded words. While previous unsupervised parsing methods mostly focus on only inducing one class of grammars, we introduce a novel model, StructFormer, that can simultaneously induce dependency and constituency structure. To achieve this, we propose a new parsing framework that can jointly generate a constituency tree and dependency graph. Then we integrate the induced dependency relations into the transformer, in a differentiable manner, through a novel dependency-constrained self-attention mechanism. Experimental results show that our model can achieve strong results on unsupervised constituency parsing, unsupervised dependency parsing, and masked language modeling at the same time.

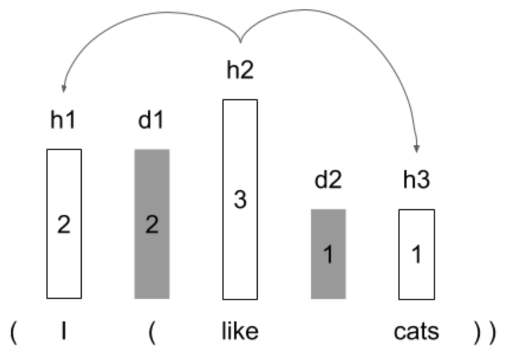
1 Introduction

Human languages have a rich latent structure. This structure is multifaceted, with the two major classes of grammar being dependency and constituency structures. There has been an exciting breath of recent work targeted at learning this structure in a data-driven unsupervised fashion (Klein and Manning, 2002; Klein, 2005; Le and Zuidema, 2015; Shen et al., 2018c; Kim et al., 2019a). The core principle behind recent methods that induce structure from data is simple - provide an inductive bias that is conducive for structure to emerge as a byproduct of some self-supervised training, e.g., language modeling. To this end, a wide range of models have been proposed that are able to successfully learn grammar structures (Shen et al., 2018a,c; Wang et al., 2019; Kim et al., 2019b,a). However, most of these works focus on inducing either constituency or dependency structures alone.

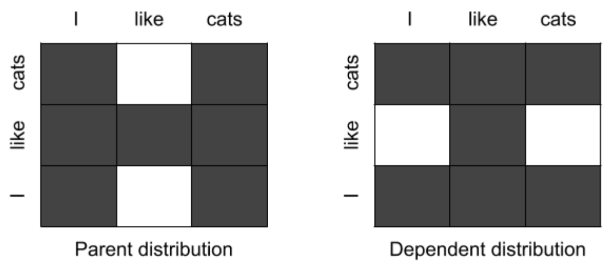
In this paper, we make two important technical contributions. First, we introduce a new neural model, StructFormer, that is able to simultaneously induce **both** dependency structure and constituency structure. Specifically, our approach aims to unify latent structure induction of different types of grammar within the same framework. Second, StructFormer is able to induce dependency structures from raw data in an end-to-end unsupervised fashion. Most existing approaches induce dependency structures from other syntactic information like gold POS tags (Klein and Manning, 2004; Cohen and Smith, 2009; Jiang et al., 2016). Previous works, having trained from words alone, often requires additional information, like pre-trained word clustering (Spitkovsky et al., 2011), pre-trained word embedding (He et al., 2018), acoustic cues (Pate and Goldwater, 2013), or annotated data from related languages (Cohen et al., 2011).

We introduce a new inductive bias that enables the Transformer models to induce a directed dependency graph in a fully unsupervised manner. To avoid the necessity of using grammar labels during training, we use a distance-based parsing mechanism. The parsing mechanism predicts a sequence of Syntactic Distances T (Shen et al., 2018b) and a sequence of Syntactic Heights Δ (Luo et al., 2019) to represent dependency graphs and constituency trees at the same time. Examples of Δ and T are illustrated in Figure 1a. Based on the syntactic distances (T) and syntactic heights (Δ), we provide a new dependency-constrained self-attention layer to replace the multi-head self-attention layer in standard transformer model. More specifically, the new attention head can only attend its parent (to avoid confusion with self-attention head, we use "parent" to denote "head" in dependency graph) or

[*] Corresponding author: `yikang.shn@gmail.ca`. Work done while interning at Google Reseach.



(a) An example of Syntactic Distances T (grey bars) and Syntactic Heights Δ (white bars). In this example, `like` is the parent (head) of constituent `(like cats)` and `(I like cats)`.



(b) Two types of dependency relations. The parent distribution allows each token to attend on its parent. The dependent distribution allows each token to attend on its dependents. For example the parent of `cats` is `like`. `Cats` and `I` are dependents of `like` Each attention head will receive a different weighted sum of these relations.

Figure 1: An example of our parsing mechanism and dependency-constrained self-attention mechanism. The parsing network first predicts the syntactic distance T and syntactic height Δ to represent the latent structure of the input sentence `I like cats`. Then the parent and dependent relations are computed in a differentiable manner from T and Δ.

its dependents in the predicted dependency structure, through a weighted sum of relations shown in Figure 1b. In this way, we replace the complete graph in the standard transformer model with a differentiable directed dependency graph. During the process of training on a downstream task (e.g. masked language model), the model will gradually converge to a reasonable dependency graph via gradient descent.

Incorporating the new parsing mechanism, the dependency-constrained self-attention, and the Transformer architecture, we introduce a new model named StructFormer. The proposed model can perform unsupervised dependency and constituency parsing at the same time, and can leverage the parsing results to achieve strong performance on masked language model tasks.

2 Related Work

Previous works on unsupervised dependency parsing are primarily based on the dependency model with valence (DMV) (Klein and Manning, 2004) and its extension (Daumé III, 2009; Gillenwater et al., 2010). To effectively learn the DMV model for better parsing accuracy, a variety of inductive biases and handcrafted features, such as correlations between parameters of grammar rules involving different part-of-speech (POS) tags, have been proposed to incorporate prior information into learning. The most recent progress is the neural DMV model (Jiang et al., 2016), which uses a neural network model to predict the grammar rule probabilities based on the distributed representation of POS tags. However, most existing unsupervised dependency parsing algorithms require the gold POS tags to ge provided as inputs. These gold POS tags are labeled by humans and can be potentially difficult (or prohibitively expensive) to obtain for large corpora. Spitkovsky et al. (2011) proposed to overcome this problem with unsupervised word clustering that can dynamically assign tags to each word considering its context. He et al. (2018) overcame the problem by combining DMV model with invertible neural network to jointly model discrete syntactic structure and continuous word representations.

Unsupervised constituency parsing has recently received more attention. PRPN (Shen et al., 2018a) and ON-LSTM (Shen et al., 2018c) induce tree structure by introducing an inductive bias to recurrent neural networks. PRPN proposes a parsing network to compute the syntactic distance of all word pairs, while a reading network uses the syntactic structure to attend to relevant memories. ON-LSTM allows hidden neurons to learn long-term or short-term information by a novel gating mechanism and activation function. In URNNG (Kim et al., 2019b), amortized variational inference was applied between a recurrent neural network grammar (RNNG) (Dyer et al., 2016) decoder and a tree structure inference network, which encourages the decoder to generate reasonable tree structures. DIORA (Drozdov et al., 2019) proposed using inside-outside dynamic programming to compose latent representations from all possible binary trees. The representations of inside and outside passes from the same sentences are optimized to be close to each other. The compound PCFG (Kim et al., 2019a) achieves grammar induction by maximizing the marginal likelihood of the sentences which are generated by a probabilistic context-free grammar (PCFG). Tree Transformer (Wang et al., 2019) adds extra locality constraints to the Transformer encoder's self-attention to encourage the attention heads to follow a tree structure such that each token can only attend on nearby neighbors in lower layers and gradually extend the attention field to further tokens when climbing to higher layers. Neural L-PCFG (Zhu et al., 2020) demonstrated that PCFG can benefit from modeling lexical dependencies. Similar to StructFormer, the Neural L-PCFG induces both constituents and dependencies within a single model.

Though large scale pre-trained models have dominated most natural language processing tasks, some recent work indicates that neural network models can see accuracy gains by leveraging syntactic information rather than ignoring it (Marcheggiani and Titov, 2017; Strubell et al., 2018). Strubell et al. (2018) introduces syntactically-informed self-attention that force one attention head to attend on the syntactic governor of the input token. Omote et al. (2019) and Deguchi et al. (2019) argue that dependency-informed self-attention can improve Transformer's performance on machine translation. Kuncoro et al. (2020) shows that syntactic biases help large scale pre-trained models, like BERT, to achieve better language understanding.

3 Syntactic Distance and Height

In this section, we first reintroduce the concepts of syntactic distance and height, then discuss their relations in the context of StructFormer.

3.1 Syntactic Distance

Syntactic distance is proposed in Shen et al. (2018b) to quantify the process of splitting sentences into smaller constituents.

Definition 3.1. Let **T** be a constituency tree for sentence $(w_1, ..., w_n)$. The height of the lowest common ancestor for consecutive words x_i and x_{i+1} is $\tilde{\tau}_i$. Syntactic distances $\mathrm{T} = (\tau_1, ..., \tau_{n-1})$ are defined as a sequence of $n-1$ real scalars that share the same rank as $(\tilde{\tau}_1, ..., \tilde{\tau}_{n-1})$.

In other words, each syntactic distance d_i is associated with a split point $(i, i+1)$ and specify the relative order in which the sentence will be split into smaller components. Thus, any sequence of $n-1$ real values can unambiguously map to an unlabeled binary constituency tree with n leaves through the Algorithm 1 (Shen et al., 2018b). As Shen et al. (2018c,a); Wang et al. (2019) pointed out, the syntactic distance reflects the information communication between constituents. More concretely, a large syntactic distance τ_i represents that short-term or local information should not be communicated between $(x_{\leq i})$ and $(x_{>i})$. While cooperating with appropriate neural network architectures, we can leverage this feature to build unsupervised dependency parsing models.

Algorithm 1 Distance to binary constituency tree

```
1: function CONSTITUENT($\mathbf{w}$, $\mathbf{d}$)
2:     if $\mathbf{d} = []$ then
3:         $\mathbf{T} \Leftarrow$ Leaf($\mathbf{w}$)
4:     else
5:         $i \Leftarrow \arg\max_i(\mathbf{d})$
6:         child$_l \Leftarrow$ Constituent($\mathbf{w}_{\leq i}$, $\mathbf{d}_{<i}$)
7:         child$_r \Leftarrow$ Constituent($\mathbf{w}_{>i}$, $\mathbf{d}_{>i}$)
8:         $\mathbf{T} \Leftarrow$ Node(child$_l$, child$_r$)
9:     return $\mathbf{T}$
```

Algorithm 2 Converting binary constituency tree to dependency graph

```
1: function DEPENDENT($\mathbf{T}$, $\Delta$)
2:     if $\mathbf{T} = w$ then
3:         $\mathbf{D} \Leftarrow []$, parent $\Leftarrow w$
4:     else
5:         child$_l$, child$_r \Leftarrow \mathbf{T}$
6:         $\mathbf{D}_l$, parent$_l \Leftarrow$ Dependent(child$_l$, $\Delta$)
7:         $\mathbf{D}_r$, parent$_r \Leftarrow$ Dependent(child$_r$, $\Delta$)
8:         $\mathbf{D} \Leftarrow$ Union($\mathbf{D}_l$, $\mathbf{D}_r$)
9:         if $\Delta$(parent$_l$) $>$ $\Delta$(parent$_r$) then
10:            $\mathbf{D}$.add(parent$_l \leftarrow$ parent$_r$)
11:            parent $\Leftarrow$ parent$_l$
12:        else
13:            $\mathbf{D}$.add(parent$_r \leftarrow$ parent$_l$)
14:            parent $\Leftarrow$ parent$_r$
15:    return $\mathbf{D}$, parent
```

3.2 Syntactic Height

Syntactic height is proposed in Luo et al. (2019), where it is used to capture the distance to the root node in a dependency graph. A word with high syntactic height means it is close to the root node. In this paper, to match the definition of syntactic distance, we redefine syntactic height as:

Definition 3.2. Let **D** be a dependency graph for sentence $(w_1, ..., w_n)$. The height of a token w_i in **D** is $\tilde{\delta}_i$. The syntactic heights of **D** can be any sequence of n real scalars $\Delta = (\delta_1, ..., \delta_n)$ that share the same rank as $(\tilde{\delta}_1, ..., \tilde{\delta}_n)$.

Although the syntactic height is defined based on the dependency structure, we cannot rebuild the original dependency structure by syntactic heights alone, since there is no information about whether a token should be attached to the left side or the right side. However, given an unlabelled constituent tree, we can convert it into a dependency graph with the help of syntactic distance. The converting process is similar to the standard process of converting constituency treebank to dependency treebank (Gelbukh et al., 2005). Instead of using the constituent labels and POS tags to identify the parent of each constituent, we simply assign the token with the largest syntactic height as the parent of each constituent. The conversion algorithm is described in Algorithm 2. In Appendix A.1, we also propose a joint algorithm, that takes T and Δ as inputs and jointly outputs a constituency tree and dependency graph.

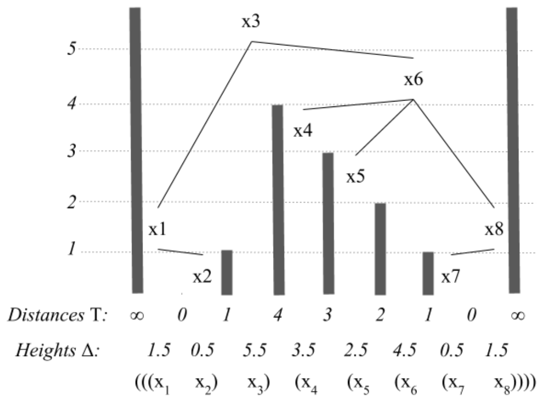
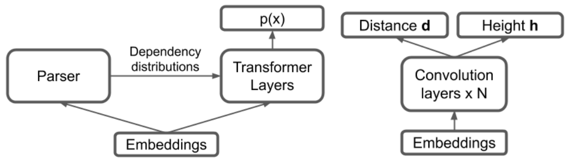

5
4
3
2
1
x3
x6
x4
x5
x1
x8
x2
x7
Distances T: ∞ 0 1 4 3 2 1 0 ∞
Heights Δ: 1.5 0.5 5.5 3.5 2.5 4.5 0.5 1.5
(((x1 x2) x3) (x4 (x5 (x6 (x7 x8))))

Figure 2: An example of T, Δ and respective dependency graph $\mathbf{D}$. Solid lines represent dependency relations between tokens. StructFormer only allow tokens with dependency relation to attend on each other.

3.3 The relation between Syntactic Distance and Height

As discussed previously, the syntactic distance controls information communication between the two sides of the split point. The syntactic height quantifies the centrality of each token in the dependency graph. A token with large syntactic height tends to have more long-term dependency relations to connect different parts of the sentence together. In StructFormer, we quantify the syntactic distance and height on the same scale. Given a split point $(i, i+1)$ and it's syntactic distance δ_i, only tokens x_j with $\tau_j > \delta_i$ can attend across the split point $(i, i+1)$. Thus tokens with small syntactic height are limited to attend to nearby tokens. Figure 2 provides an example of T, Δ and respective dependency graph $\mathbf{D}$.

However, if the left and right boundary syntactic distance of a constituent $[l, r]$ are too large, all words in $[l, r]$ will be forced to only attend to other words in $[l, r]$. Their contextual embedding will not be able to encode the full context. To avoid this phenomena, we propose calibrating T according to Δ in Appendix A.2



(a) Model Architecture (b) Parsing Network

Figure 3: The Architecture of StructFormer. The parser takes shared word embeddings as input, outputs syntactic distances T, syntactic heights Δ, and dependency distributions between tokens. The transformer layers take word embeddings and dependency distributions as input, output contextualized embeddings for input words.

4 StructFormer

In this section, we present the StructFormer model. Figure 3a shows the architecture of StructFormer, which includes a parser network and a Transformer module. The parser network predicts T and Δ, then passes them to a set of differentiable functions to generate dependency distributions. The Transformer module takes these distributions and the sentence as input to computes a contextual embedding for each position. The StructFormer can be trained in an end-to-end fashion on a Masked Language Model task. In this setting, the gradient back propagates through the relation distributions into the parser.

4.1 Parsing Network

As shown in Figure 3b, the parsing network takes word embeddings as input and feeds them into several convolution layers:

$$s_{l,i} = \tanh\left(\text{Conv}\left(s_{l-1,i-W}, ..., s_{l-1,i+W}\right)\right) \quad (1)$$

where $s_{l,i}$ is the output of l-th layer at i-th position, $s_{0,i}$ is the input embedding of token w_i, and $2W+1$ is the convolution kernel size.

Given the output of the convolution stack $s_{N,i}$, we parameterize the syntactic distance T as:

$$\tau_i = \begin{cases} \mathbf{W}_1^{\tau} \tanh\left(\mathbf{W}_2^{\tau} \begin{bmatrix} s_{N,i} \\ s_{N,i+1} \end{bmatrix}\right), \\ \qquad\qquad 1 \leq i \leq n-1 \\ \infty, \quad i = 0 \quad \text{or} \quad i = n \end{cases} \tag{2}$$

where τ_i is the contextualized distance for the i-th split point between token w_i and w_{i+1}. The syntactic height Δ is parameterized in a similar way:

$$\delta_i = \mathbf{W}_1^{\delta} \tanh\left(\mathbf{W}_2^{\delta} s_{N,i} + b_2^{\delta}\right) + b_1^{\delta} \tag{3}$$

4.2 Estimate the Dependency Distribution

Given T and Δ, we now explain how to estimate the probability $p(x_j|x_i)$ such that the j-th token is the parent of the i-th token. The first step is identifying the smallest legal constituent $\mathbf{C}(x_i)$, that contains x_i and x_i is not $\mathbf{C}(x_i)$'s parent. The second step is identifying the parent of the constituent $x_j = \mathbf{Pr}(\mathbf{C}(x_i))$. Given the discussion in section 3.2, the parent of $\mathbf{C}(x_i)$ must be the parent of x_i. Thus, the two-stages of identifying the parent of x_i can be formulated as:

$$\mathbf{D}(x_i) = \mathbf{Pr}(\mathbf{C}(x_i)) \tag{4}$$

In StructFormer, $\mathbf{C}(x_i)$ is represented as constituent $[l, r]$, where l is the starting index ($l \leq i$) of $\mathbf{C}(x_i)$ and r is the ending index ($r \geq i$) of $\mathbf{C}(x_i)$.

In a dependency graph, x_i is only connected to its parent and dependents. This means that x_i does not have direct connection to the outside of $\mathbf{C}(x_i)$. In other words, $\mathbf{C}(x_i) = [l, r]$ is the smallest constituent that satisfies:

$$\delta_i < \tau_{l-1}, \quad \delta_i < \tau_r \tag{5}$$

where τ_{l-1} is the first $\tau_{<i}$ that is larger then δ_i while looking backward, and τ_r is the first $\tau_{\geq i}$ that is larger then δ_i while looking forward. For example, in Figure 2, $\delta_4 = 3.5$, $\tau_3 = 4 > \delta_4$ and $\tau_8 = \infty > \delta_4$, thus $\mathbf{C}(x_4) = [4, 8]$. To make this process differentiable, we define τ_k as a real value and δ_i as a probability distribution $p(\tilde{\delta}_i)$. For the simplicity and efficiency of computation, we directly parameterize the cumulative distribution function $p(\tilde{\delta}_i > \tau_k)$ with sigmoid function:

$$p(\tilde{\delta}_i > \tau_k) = \sigma((\delta_i - \tau_k)/\mu_1) \tag{6}$$

where σ is the sigmoid function, δ_i is the mean of distribution $p(\tilde{\delta}_i)$ and μ_1 is a learnable temperature term. Thus the probability that the l-th ($l < i$) token is inside $\mathbf{C}(x_i)$ is equal to the probability that $\tilde{\delta}_i$ is larger then the maximum distance τ between l and i:

$$\begin{aligned} p(l \in \mathbf{C}(x_i)) &= p(\tilde{\delta}_i > \max(\tau_{i-1}, ..., \tau_l)) \quad (7) \\ &= \sigma((\delta_i - \max(\tau_l, ..., \tau_{i-1}))/\mu) \end{aligned}$$

Then we can compute the probability distribution for l:

$$\begin{aligned} p(l|i) &= p(l \in \mathbf{C}(x_i)) - p(l-1 \in \mathbf{C}(x_i)) \\ &= \sigma((\delta_i - \max(\tau_l, ..., \tau_{i-1}))/\mu) - \\ &\quad \sigma((\delta_i - \max(\tau_{l-1}, ..., \tau_{i-1}))/\mu) \quad (8) \end{aligned}$$

Similarly, we can compute the probability distribution for r:

$$\begin{aligned} p(r|i) &= \sigma((\delta_i - \max(\tau_i, ..., \tau_{r-1}))/\mu) - \\ &\quad \sigma((\delta_i - \max(\tau_i, ..., \tau_r))/\mu) \quad (9) \end{aligned}$$

The probability distribution for $[l, r] = \mathbf{C}(x_i)$ can be computed as:

$$p_{\mathbf{C}}([l, r]|i) = \begin{cases} p(l|i)p(r|i), & l \leq i \leq r \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

The second step is to identify the parent of $[l, r]$. For any constituent $[l, r]$, we choose the $j = \text{argmax}_{k \in [l,r]}(\delta_k)$ as the parent of $[l, r]$. In the previous example, given constituent $[4, 8]$, the maximum syntactic height is $\delta_6 = 4.5$, thus $\mathbf{Pr}([4, 8]) = x_6$. We use softmax function to parameterize the probability $p_{\mathbf{Pr}}(j|[l, r])$:

$$p_{\mathbf{Pr}}(j|[l, r]) = \begin{cases} \frac{\exp(h_j/\mu_2)}{\sum_{l \leq k \leq r} \exp(h_k/\mu_2)}, & l \leq t \leq r \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

Given probability $p(j|[l, r])$ and $p([l, r]|i)$, we can compute the probability that x_j is the parent of x_i:

$$p_{\mathbf{D}}(j|i) = \begin{cases} \sum_{[l,r]} p_{\mathbf{Pr}}(j|[l, r]) p_{\mathbf{C}}([l, r]|i), & i \neq j \\ 0, & i = j \end{cases} \tag{12}$$

4.3 Dependency-Constrained Multi-head Self-Attention

The multi-head self-attention in the transformer can be seen as a information propagation mechanism on the complete graph $\mathbf{G} = (X, E)$, where the set of vertices X contains all n tokens in the sentence, and the set of edges E contains all possible word pairs (x_i, x_j). StructFormer replace the complete graph $\mathbf{G}$ with a soft dependency graph $\mathbf{D} = (X, A)$, where A is the matrix of $n \times n$ probabilities. $A_{ij} = p_{\mathbf{D}}(j|i)$ is the probability of the j-th token depending on the i-th token. The reason that we called it a directed edge is that each specific head is only allow to propagate information either from parent to dependent or from from dependent to parent. To do so, structformer associate each attention head with a probability distribution over parent or dependent relation.

$$p_{\text{parent}} = \frac{\exp(w_{\text{parent}})}{\exp(w_{\text{parent}}) + \exp(w_{\text{dep}})} \quad (13)$$

$$p_{\text{dep}} = \frac{\exp(w_{\text{dep}})}{\exp(w_{\text{parent}}) + \exp(w_{\text{dep}})} \quad (14)$$

where w_{parent} and w_{dep} are learnable parameters that associated with each attention head, p_{parent} is the probability that this head will propagate information from parent to dependent, vice versa. The model will learn to assign this association from the downstream task via gradient descent. Then we can compute the probability that information can be propagated from node j to node i via this head:

$$p_{i,j} = p_{\text{parent}} p_{\mathbf{D}}(j|i) + p_{\text{dep}} p_{\mathbf{D}}(i|j) \quad (15)$$

However, Htut et al. (2019) pointed out that different heads tend to associate with different type of universal dependency relations (including `nsubj`, `obj`, `advmod`, etc), but there is no generalist head can that work with all different relations. To accommodate this observation, we compute a individual probability for each head and pair of tokens (x_i, x_j):

$$q_{i,j} = \text{sigmoid}\left(\frac{QK^T}{\sqrt{d_k}}\right) \quad (16)$$

where Q and K are query and key matrix in a standard transformer model and d_k is the dimension of attention head. The equation is inspired by the scaled dot-product attention in transformer. We replace the original softmax function with a sigmoid function, so $q_{i,j}$ became an independent probability that indicates whether x_i should attend on x_j through the current attention head. In the end, we propose to replace transformer's scaled dot-product attention with our dependency-constrained self-attention:

$$\text{Attention}(Q_i, K_j, V_j, \mathbf{D}) = p_{i,j} q_{i,j} V_j \quad (17)$$

5 Experiments

We evaluate the proposed model on three tasks: Masked Language Modeling, Unsupervised Constituency Parsing and Unsupervised Dependency Parsing.

Our implementation of StructFormer is close to the original Transformer encoder (Vaswani et al., 2017). Except that we put the layer normalization in front of each layer, similar to the T5 model (Raffel et al., 2019). We found that this modification allows the model to converges faster. For all experiments, we set the number of layers $L = 8$, the embedding size and hidden size to be $d_{model} = 512$, the number of self-attention heads $h = 8$, the feed-forward size $d_{ff} = 2048$, dropout rate as 0.1, and the number of convolution layers in the parsing network as $L_p = 3$.

5.1 Masked Language Model

Masked Language Modeling (MLM) has been widely used as a pretraining object for larger-scale pretraining models. In BERT (Devlin et al., 2018) and RoBERTa (Liu et al., 2019), authors found that MLM perplexities on held-out evaluation set have a positive correlation with the end-task performance. We trained and evaluated our model on 2 different datasets: the Penn TreeBank (PTB) and BLLIP. In our MLM experiments, each token has an independent chance to be replaced by a mask token `<mask>`, except that we never replace $< unk >$ token. The training and evaluation object for Masked Language Model is to predict the replaced tokens. The performance of MLM is evaluated by measuring perplexity on masked words.

PTB is a standard dataset for language modeling (Mikolov et al., 2012) and unsupervised constituency parsing (Shen et al., 2018c; Kim et al., 2019a). Following the setting proposed in Shen et al. (2018c), we use Mikolov et al. (2012)'s prepossessing process, which removes all punctuations, and replaces low frequency tokens with `<unk>`. The preprocessing results in a vocabulary size of 10001 (including `<unk>`, `<pad>` and `<mask>`). For PTB, we use a 30% mask rate.

BLLIP is a large Penn Treebank-style parsed corpus of approximately 24 million sentences. We train and evaluate StructFormer on three splits of BLLIP: BLLIP-XS (40k sentences, 1M tokens), BLLIP-SM (200K sentences, 5M tokens), and BLLIP-MD (600K sentences, 14M tokens). They are obtained by randomly sampling sections from

Model	PTB	BLLIP -XS	BLLIP -SM	BLLIP -MD
Transformer	64.05	93.90	19.92	14.31
StructFormer	60.94	57.28	18.70	13.70

Table 1: Masked Language Model perplexities on different datasets.

BLLIP 1987-89 Corpus Release 1. All models are tested on a shared held-out test set (20k sentences, 500k tokens). Following the settings provided in (Hu et al., 2020), we use subword-level vocabulary extracted from the GPT-2 pre-trained model rather than the BLLIP training corpora. For BLLIP, we use a 15% mask rate.

The masked language model results are shown in Table 1. StructFormer consistently outperforms our Transformer baseline. This result aligns with previous observations that linguistically informed self-attention can help Transformers achieve stronger performance. We also observe that StructFormer converges much faster than the standard Transformer model.

5.2 Unsupervised Constituency Parsing

The unsupervised constituency parsing task compares the latent tree structure induced by the model with those annotated by human experts. We use the Algorithm 1 to predict the constituency trees from T predicted by StructFormer. Following the experiment settings proposed in Shen et al. (2018c), we take the model trained on PTB dataset and evaluate it on WSJ test set. The WSJ test set is section 23 of WSJ corpus, it contains 2416 human expert labeled sentences. Punctuation is ignored during the evaluation.

Methods	UF1
RANDOM	21.6
LBRANCH	9.0
RBRANCH	39.8
PRPN (Shen et al., 2018a)	37.4 (0.3)
ON-LSTM (Shen et al., 2018c)	47.7 (1.5)
Tree-T (Wang et al., 2019)	49.5
URNNG (Kim et al., 2019b)	52.4
C-PCFG (Kim et al., 2019a)	55.2
Neural L-PCFGs (Zhu et al., 2020)	55.31
StructFormer	54.0 (0.3)

Table 2: Unsupervised constituency parsing tesults. * results are from Kim et al. (2020). UF1 stands for Unlabeled F1.

Table 2 shows that our model achieves strong results on unsupervised constituency parsing. While the C-PCFG (Kim et al., 2019a) achieve a stronger parsing performance with its strong linguistic constraints (e.g. a finite set of production rules), StructFormer may have a border domain of application. For example, it can replace the standard transformer encoder in most of the popular large-scale pre-trained language models (e.g. BERT and ReBERTa) and transformer based machine translation models. Different from the transformer-based Tree-T (Wang et al., 2019), we did not directly use constituents to restrict the self-attention receptive field. But StructFormer achieves a stronger constituency parsing performance. This result may suggest that dependency relations are more suitable for grammar induction in transformer-based models. Table 3 shows that our model achieves strong accuracy while predicting Noun Phrase (NP), Preposition Phrase (PP), Adjective Phrase (ADJP), and Adverb Phrase (ADVP).

	PRPN	ON	C-PCFG	Tree-T	Ours
SBAR	50.0%	52.5%	**56.1%**	36.4%	48.7%
NP	59.2%	64.5%	**74.7%**	67.6%	72.1%
VP	**46.7%**	41.0%	41.7%	38.5%	43.0%
PP	57.2%	54.4%	68.8%	52.3%	**74.1%**
ADJP	44.3%	38.1%	40.4%	24.7%	**51.9%**
ADVP	32.8%	31.6%	52.5%	55.1%	**69.5%**

Table 3: Fraction of ground truth constituents that were predicted as a constituent by the models broken down by label (i.e. label recall)

5.3 Unsupervised Dependency Parsing

The unsupervised dependency parsing evaluation compares the induced dependency relations with those in the reference dependency graph. The most common metric is the Unlabeled Attachment Score (UAS), which measures the percentage that a token is correctly attached to its parent in the reference tree. Another widely used metric for unsupervised dependency parsing is Undirected Unlabeled Attachment Score (UUAS) measures the percentage that the reference undirected and unlabeled connections are recovered by the induced tree. Similar to the unsupervised constituency parsing, we take the model trained on PTB dataset and evaluate it on WSJ test set (section 23). For the WSJ test set, reference dependency graphs are converted from its human-annotated constituency trees. However, there are two different sets of rules for the conversion: the Stanford dependencies and the CoNLL dependencies. While Stanford dependencies are used as reference dependencies in previous unsupervised

Relations	MLM	Constituency	Stanford		Conll	
	PPL	UF1	UAS	UUAS	UAS	UUAS
parent+dep	60.9 (1.0)	54.0 (0.3)	46.2 (0.4)	61.6 (0.4)	36.2 (0.1)	56.3 (0.2)
parent	63.0 (1.2)	40.2 (3.5)	32.4 (5.6)	49.1 (5.7)	30.0 (3.7)	50.0 (5.3)
dep	63.2 (0.6)	51.8 (2.4)	15.2 (18.2)	41.6 (16.8)	20.2 (12.2)	44.7 (13.9)

Table 4: The performance of StructFormer with different combinations of attention masks. UAS stands for Unlabeled Attachment Score. UUAS stands for Undirected Unlabeled Attachment Score.

Methods	UAS
w/o gold POS tags	
DMV (Klein and Manning, 2004)	35.8
E-DMV (Headden III et al., 2009)	38.2
UR-A E-DMV (Tu and Honavar, 2012)	46.1
CS* (Spitkovsky et al., 2013)	64.4*
Neural E-DMV (Jiang et al., 2016)	42.7
Gaussian DMV (He et al., 2018)	43.1 (1.2)
INP (He et al., 2018)	47.9 (1.2)
Neural L-PCFGs (Zhu et al., 2020)	40.5 (2.9)
StructFormer	46.2 (0.4)
w/ gold POS tags (for reference only)	
DMV (Klein and Manning, 2004)	39.7
UR-A E-DMV (Tu and Honavar, 2012)	57.0
MaxEnc (Le and Zuidema, 2015)	65.8
Neural E-DMV (Jiang et al., 2016)	57.6
CRFAE (Cai et al., 2017)	55.7
L-NDMV† (Han et al., 2017)	63.2

Table 5: Dependency Parsing Results on WSJ testset. Starred entries (*) benefit from additional punctuation-based constraints. Daggered entries (†) benefit from larger additional training data. Baseline results are from He et al. (2018).

parsing papers, we noticed that our model sometimes output dependency structures that are closer to the CoNLL dependencies. Therefore, we report UAS and UUAS for both Stanford and CoNLL dependencies. Following the setting of previous papers (Jiang et al., 2016), we ignored the punctuation during evaluation. To obtain the dependency relation from our model, we compute the argmax for dependency distribution:

$$k = \mathrm{argmax}_{j \neq i} p_{\mathbf{D}}(j|i) \tag{18}$$

and assign the k-th token as the parent of i-th token.

Table 5 shows that our model achieves competitive dependency parsing performance while comparing to other models that do not require gold POS tags. While most of the baseline models still rely on some kind of latent POS tags or pre-trained word embeddings, StructFormer can be seen as an easy-to-use alternative that works in an end-to-end fashion. Table 6 shows that our model recovers 61.6% of undirected dependency relations. Given the strong performances on both dependency parsing and masked language modeling, we believe that the dependency graph schema could be a viable substitute for the complete graph schema used in the standard transformer. Appendix A.4 provides examples of parent distribution.

Since our model uses a mixture of the relation probability distribution for each self-attention head, we also studied how different combinations of relations affect the performance of our model. Table 6 shows that the model can achieve the best performance while using both parent and dependent relations. The model suffers more on dependency parsing if the parent relation is removed. And if the dependent relationship is removed, the model will suffer more on the constituency parsing. Appendix A.3 shows the weight for parent and dependent relations learnt from MLM tasks. It's interesting to observe that Structformer tends to focus on the parent relations in the first layer, and start to use both relations from the second layer.

6 Conclusion

In this paper, we introduce a novel dependency and constituency joint parsing framework. Based on the framework, we propose StructFormer, a new unsupervised parsing algorithm that does unsupervised dependency and constituency parsing at the same time. We also introduced a novel dependency-constrained self-attention mechanism that allows each attention head to focus on a specific mixture of dependency relations. This brings Transformers closer to modeling a directed dependency graph. The experiments show promising results that StructFormer can induce meaningful dependency and constituency structures and achieve better performance on masked language model tasks. This research provides a new path to build more linguistic bias into a pre-trained language model.

References

Jiong Cai, Yong Jiang, and Kewei Tu. 2017. Crf autoencoder for unsupervised dependency parsing. *arXiv preprint arXiv:1708.01018*.

Shay B Cohen, Dipanjan Das, and Noah A Smith. 2011. Unsupervised structure prediction with non-parallel multilingual guidance. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 50–61.

Shay B Cohen and Noah A Smith. 2009. Shared logistic normal distributions for soft parameter tying in unsupervised grammar induction. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 74–82.

Hal Daumé III. 2009. Unsupervised search-based structured prediction. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 209–216.

Hiroyuki Deguchi, Akihiro Tamura, and Takashi Ninomiya. 2019. Dependency-based self-attention for transformer nmt. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pages 239–246.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Andrew Drozdov, Patrick Verga, Mohit Yadav, Mohit Iyyer, and Andrew McCallum. 2019. Unsupervised latent tree induction with deep inside-outside recursive auto-encoders. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1129–1141.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. 2016. Recurrent neural network grammars. In *Proceedings of NAACL-HLT*, pages 199–209.

Alexander Gelbukh, Sulema Torres, and Hiram Calvo. 2005. Transforming a constituency treebank into a dependency treebank. *Procesamiento del lenguaje natural*, (35):145–152.

Jennifer Gillenwater, Kuzman Ganchev, João Graça, Fernando Pereira, and Ben Taskar. 2010. Sparsity in dependency grammar induction. *ACL 2010*, page 194.

Wenjuan Han, Yong Jiang, and Kewei Tu. 2017. Dependency grammar induction with neural lexicalization and big training data. *arXiv preprint arXiv:1708.00801*.

Junxian He, Graham Neubig, and Taylor Berg-Kirkpatrick. 2018. Unsupervised learning of syntactic structure with invertible neural projections. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1292–1302.

William P Headden III, Mark Johnson, and David McClosky. 2009. Improving unsupervised dependency parsing with richer contexts and smoothing. In *Proceedings of human language technologies: the 2009 annual conference of the North American chapter of the association for computational linguistics*, pages 101–109.

Phu Mon Htut, Jason Phang, Shikha Bordia, and Samuel R Bowman. 2019. Do attention heads in bert track syntactic dependencies? *arXiv preprint arXiv:1911.12246*.

Jennifer Hu, Jon Gauthier, Peng Qian, Ethan Wilcox, and Roger P Levy. 2020. A systematic assessment of syntactic generalization in neural language models. *arXiv preprint arXiv:2005.03692*.

Yong Jiang, Wenjuan Han, Kewei Tu, et al. 2016. Unsupervised neural dependency parsing. Association for Computational Linguistics (ACL).

Taeuk Kim, Jihun Choi, Daniel Edmiston, and Sang-goo Lee. 2020. Are pre-trained language models aware of phrases? simple but strong baselines for grammar induction. *arXiv preprint arXiv:2002.00737*.

Yoon Kim, Chris Dyer, and Alexander M Rush. 2019a. Compound probabilistic context-free grammars for grammar induction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2369–2385.

Yoon Kim, Alexander M Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. 2019b. Unsupervised recurrent neural network grammars. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1105–1117.

Dan Klein. 2005. *The unsupervised learning of natural language structure*. Stanford University Stanford.

Dan Klein and Christopher D Manning. 2002. A generative constituent-context model for improved grammar induction. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 128–135.

Dan Klein and Christopher D Manning. 2004. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd annual meeting of the association for computational linguistics (ACL-04)*, pages 478–485.

Adhiguna Kuncoro, Lingpeng Kong, Daniel Fried, Dani Yogatama, Laura Rimell, Chris Dyer, and Phil Blunsom. 2020. Syntactic structure distillation pretraining for bidirectional encoders. *arXiv preprint arXiv:2005.13482*.

Phong Le and Willem Zuidema. 2015. Unsupervised dependency parsing: Let's use supervised parsers. *arXiv preprint arXiv:1504.04666*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Hongyin Luo, Lan Jiang, Yonatan Belinkov, and James Glass. 2019. Improving neural language models by segmenting, attending, and predicting the future. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1483–1493.

Diego Marcheggiani and Ivan Titov. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. *arXiv preprint arXiv:1703.04826*.

Tomáš Mikolov et al. 2012. Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*, 80:26.

Yutaro Omote, Akihiro Tamura, and Takashi Ninomiya. 2019. Dependency-based relative positional encoding for transformer nmt. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pages 854–861.

John K Pate and Sharon Goldwater. 2013. Unsupervised dependency parsing with acoustic cues. *Transactions of the Association for Computational Linguistics*, 1:63–74.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Yikang Shen, Zhouhan Lin, Chin-wei Huang, and Aaron Courville. 2018a. Neural language modeling by jointly learning syntax and lexicon. In *International Conference on Learning Representations*.

Yikang Shen, Zhouhan Lin, Athul Paul Jacob, Alessandro Sordoni, Aaron Courville, and Yoshua Bengio. 2018b. Straight to the tree: Constituency parsing with neural syntactic distance. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1171–1180.

Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron Courville. 2018c. Ordered neurons: Integrating tree structures into recurrent neural networks. In *International Conference on Learning Representations*.

Valentin I Spitkovsky, Hiyan Alshawi, Angel Chang, and Dan Jurafsky. 2011. Unsupervised dependency parsing without gold part-of-speech tags. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1281–1290.

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

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. Linguistically-informed self-attention for semantic role labeling. *arXiv preprint arXiv:1804.08199*.

Kewei Tu and Vasant Honavar. 2012. Unambiguity regularization for unsupervised learning of probabilistic grammars. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1324–1334.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Yaushian Wang, Hung-Yi Lee, and Yun-Nung Chen. 2019. Tree transformer: Integrating tree structures into self-attention. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1060–1070.

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 Appendix

A.1 Joint Dependency and Constituency Parsing

Algorithm 3 The joint dependency and constituency parsing algorithm. Inputs are a sequence of words $\mathbf{w}$, syntactic distances $\mathbf{d}$, syntactic heights $\mathbf{h}$. Outputs are a binary constituency tree $\mathbf{T}$, a dependency graph $\mathbf{D}$ that is represented as a set of dependency relations, the parent of dependency graph $\mathbf{D}$, and the syntactic height of parent.

```
1: function BuildTree($\mathbf{w}, \mathbf{d}, \mathbf{h}$)
2:     if $\mathbf{d} = []$ and $\mathbf{w} = [w]$ and $\mathbf{h} = [h]$ then
3:         $\mathbf{T} \Leftarrow \mathrm{Leaf}(w)$, $\mathbf{D} \Leftarrow []$, parent $\Leftarrow w$, height $\Leftarrow h$
4:     else
5:         $i \Leftarrow \arg\max(\mathbf{d})$
6:         $\mathbf{T}_l, \mathbf{D}_l, \mathrm{parent}_l, \mathrm{height}_l \Leftarrow$ BuildTree($\mathbf{d}_{<i}, \mathbf{w}_{\leq i}, \mathbf{h}_{\leq i}$)
7:         $\mathbf{T}_r, \mathbf{D}_r, \mathrm{parent}_r, \mathrm{height}_r \Leftarrow$ BuildTree($\mathbf{d}_{>i}, \mathbf{w}_{>i}, \mathbf{h}_{>i}$)
8:         $\mathbf{T} \Leftarrow \mathrm{Node}(\mathrm{child}_l \Leftarrow \mathbf{T}_l, \mathrm{child}_r \Leftarrow \mathbf{T}_r)$
9:         $\mathbf{D} \Leftarrow \mathrm{Union}(\mathbf{D}_l, \mathbf{D}_r)$
10:        if $\mathrm{height}_l > \mathrm{height}_r$ then
11:            $\mathbf{D}.\mathrm{add}(\mathrm{parent}_l \leftarrow \mathrm{parent}_r)$
12:            parent $\Leftarrow \mathrm{parent}_l$, height $\Leftarrow \mathrm{height}_l$
13:        else
14:            $\mathbf{D}.\mathrm{add}(\mathrm{parent}_r \leftarrow \mathrm{parent}_l)$
15:            parent $\Leftarrow \mathrm{parent}_r$, height $\Leftarrow \mathrm{height}_r$
16:    return $\mathbf{T}$, $\mathbf{D}$, parent, height
```

A.2 Calibrating the Syntactic Distance and Height

In Section 3.3, we explained the relation between Δ and T, that if $\delta_i < \tau_j$, the i-th word won't be able to attend beyond the j-th split point. However, in a specific case, the constraint will isolate a constituent $[l, r]$ from the rest of the sentence. If τ_{l-1} and τ_r are larger then all height $\delta_{l,...,r}$ in the constituent, then all words in $[l, r]$ won't be able to attend on the outside of the constituent. This phenomenon will prevent their output contextual embedding from encoding the full context. To avoid this phenomenon, we propose to calibrate the syntactic distance T according to the syntactic height Δ. First, we compute the maximum syntactic height for each constituent:

$$\delta_{[l,r]} = \max\left(\delta_l, ..., \delta_r\right), \quad l < r \tag{19}$$

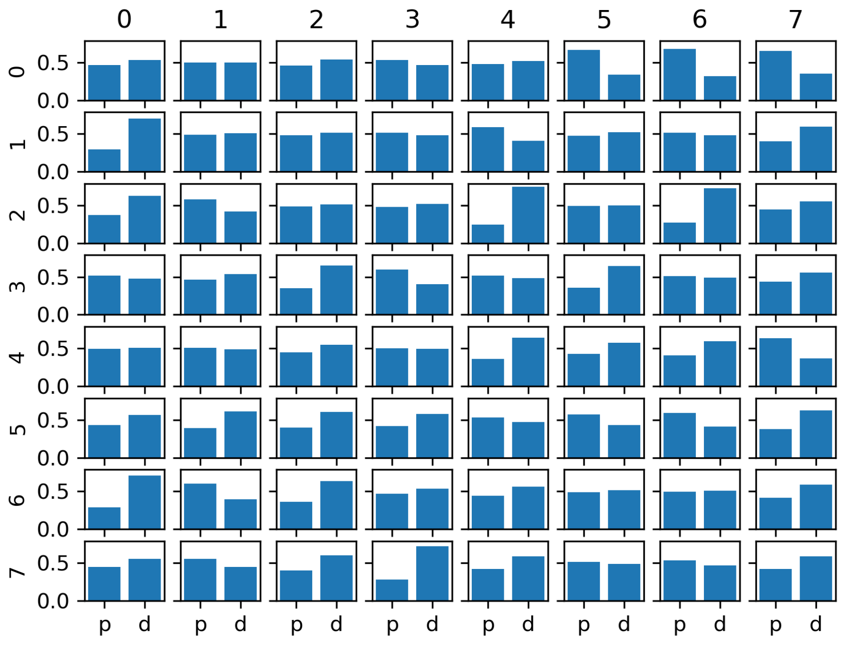
Then we compute the minimum difference between $\delta_{[l,r]}$ and $[l, r]$'s left and right boundary distance. Since we only care about constituents that the boundary distance is larger than its maximum height, we use a ReLU activation function to keep only the positive values:

$$\epsilon_{[l,r]} = \mathrm{ReLU}\left(\min\left(\tau_{l-1} - \delta_{[l,r]}, \tau_r - \delta_{[l,r]}\right)\right) \tag{20}$$

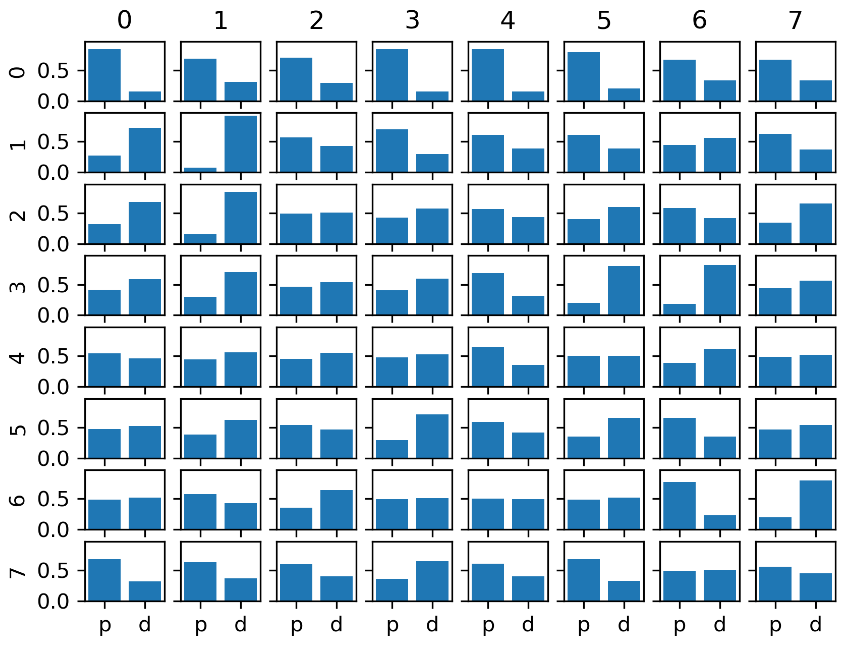
To make sure all constituent are not isolated and maintain the rank of T, we subtract all T by the maximum of ϵ:

$$\hat{\delta_i} = \delta_i - \max_{\{[l,r]\}/[1,n]}\left(\epsilon_{[l,r]}\right) \tag{21}$$

A.3 Dependency Relation Weights for Self-attention Heads



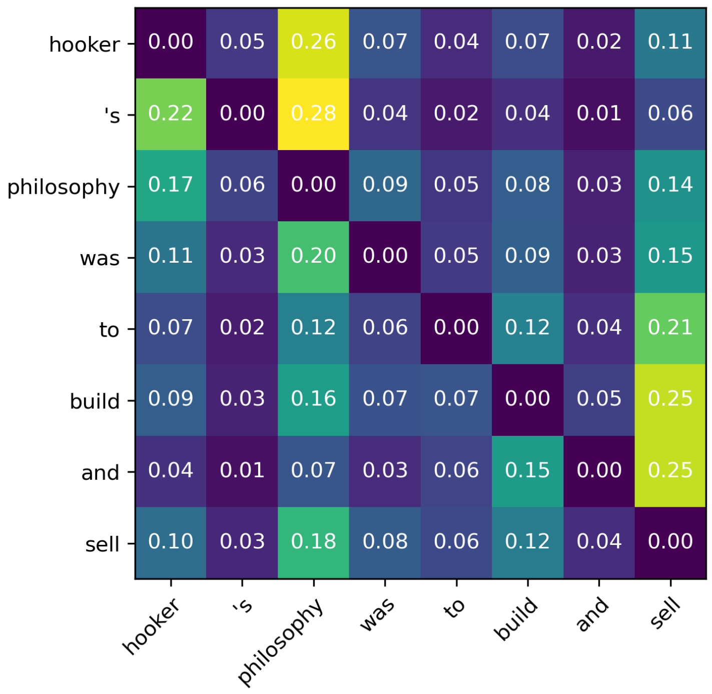
(a) Dependency relation weights learnt on PTB



(b) Dependency relation weights learnt on BLLIP-SM

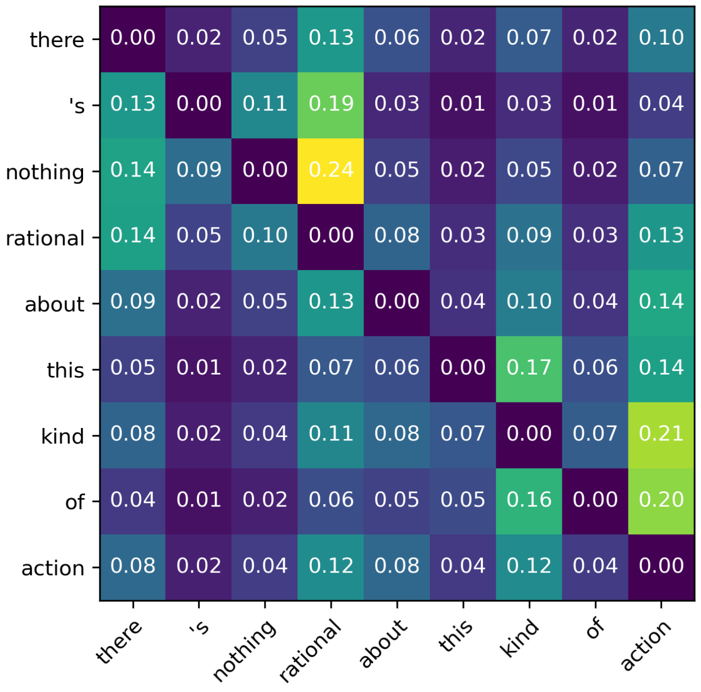
Figure 4: Dependency relation weights learnt on different datasets. Row i constains relation weights for all attention heads in the i-th transformer layer. p represents the parent relation. d represents the dependent relation. We observe a clearer preference for each attention head in the model trained on BLLIP-SM. This probably due to BLLIP-SM has signficantly more training data. It's also interesting to notice that the first layer tend to focus on parent relations.

A.4 Dependency Distribution Examples



	hooker	's	philosophy	was	to	build	and	sell
hooker	0.00	0.05	0.26	0.07	0.04	0.07	0.02	0.11
's	0.22	0.00	0.28	0.04	0.02	0.04	0.01	0.06
philosophy	0.17	0.06	0.00	0.09	0.05	0.08	0.03	0.14
was	0.11	0.03	0.20	0.00	0.05	0.09	0.03	0.15
to	0.07	0.02	0.12	0.06	0.00	0.12	0.04	0.21
build	0.09	0.03	0.16	0.07	0.07	0.00	0.05	0.25
and	0.04	0.01	0.07	0.03	0.06	0.15	0.00	0.25
sell	0.10	0.03	0.18	0.08	0.06	0.12	0.04	0.00

(a)



	there	's	nothing	rational	about	this	kind	of	action
there	0.00	0.02	0.05	0.13	0.06	0.02	0.07	0.02	0.10
's	0.13	0.00	0.11	0.19	0.03	0.01	0.03	0.01	0.04
nothing	0.14	0.09	0.00	0.24	0.05	0.02	0.05	0.02	0.07
rational	0.14	0.05	0.10	0.00	0.08	0.03	0.09	0.03	0.13
about	0.09	0.02	0.05	0.13	0.00	0.04	0.10	0.04	0.14
this	0.05	0.01	0.02	0.07	0.06	0.00	0.17	0.06	0.14
kind	0.08	0.02	0.04	0.11	0.08	0.07	0.00	0.07	0.21
of	0.04	0.01	0.02	0.06	0.05	0.05	0.16	0.00	0.20
action	0.08	0.02	0.04	0.12	0.08	0.04	0.12	0.04	0.00

(b)

Figure 5: Dependency distribution examples from WSJ test set. Each row is the parent distribution for the respective word. The sum of each distribution may not equal to 1. Against our intuition, the distribution is not very sharp. This is partially due to the ambiguous nature of the dependency graph. As we previously discussed, at least two styles of dependency rules (Conll and Stanford) exist. And without extra constraint or supervision, the model seems trying to model both of them at the same time. One interesting future work will be finding an inductive bias that can encourage the model to converge to a specific style of dependency graph.

A.5 The Performance of StructFormer with different mask rates

Mask rate	MLM	Constituency	Stanford		Conll	
	PPL	UF1	UAS	UUAS	UAS	UUAS
0.1	45.3 (1.2)	51.45 (2.7)	31.4 (11.9)	51.2 (8.1)	32.3 (5.2)	52.4 (4.5)
0.2	50.4 (1.3)	54.0 (0.6)	37.4 (12.6)	55.6 (8.8)	33.0 (5.7)	53.5 (4.7)
0.3	60.9 (1.0)	54.0 (0.3)	46.2 (0.4)	61.6 (0.4)	36.2 (0.1)	56.3 (0.2)
0.4	76.9 (1.2)	53.5 (1.5)	34.0 (10.3)	52.0 (7.4)	29.5 (5.4)	50.6 (4.1)
0.5	100.3 (1.4)	53.2 (0.9)	36.3 (9.8)	53.6 (6.8)	30.6 (4.2)	51.3 (3.2)

Table 6: The performance of StructFormer on PTB dataset with different mask rates. Dependency parsing is especially affected by the masks. Mask rate 0.3 provides the best and the most stable performance.