Tree Transformer: Integrating Tree Structures into Self-Attention

Yau-Shian Wang Hung-Yi Lee Yun-Nung Chen
National Taiwan University, Taipei, Taiwan
king6101@gmail.com hungyilee@ntu.edu.tw y.v.chen@ieee.org

Abstract

Pre-training Transformer from large-scale raw texts and fine-tuning on the desired task have achieved state-of-the-art results on diverse NLP tasks. However, it is unclear what the learned attention captures. The attention computed by attention heads seems not to match human intuitions about hierarchical structures. This paper proposes Tree Transformer, which adds an extra constraint to attention heads of the bidirectional Transformer encoder in order to encourage the attention heads to follow tree structures. The tree structures can be automatically induced from raw texts by our proposed "*Constituent Attention*" module, which is simply implemented by self-attention between two adjacent words. With the same training procedure identical to BERT, the experiments demonstrate the effectiveness of Tree Transformer in terms of inducing tree structures, better language modeling, and further learning more explainable attention scores[1].

1 Introduction

Human languages exhibit a rich hierarchical structure which is currently not exploited nor mirrored by the self-attention mechanism that is the core of the now popular Transformer architecture. Prior work that integrated hierarchical structure into neural networks either used recursive neural networks (Tree-RNNs) (C.Goller and A.Kuchler, 1996; Socher et al., 2011; Tai et al., 2015) or simultaneously generated a syntax tree and language in RNN (Dyer et al., 2016), which have shown beneficial for many downstream tasks (Aharoni and Goldberg, 2017; Eriguchi et al., 2017; Strubell et al., 2018; Zaremoodi and Haffari, 2018). Considering the requirement of the annotated parse trees and the costly annotation effort, most prior work relied on the supervised syntactic parser. However, a supervised parser may be unavailable when the language is low-resourced or the target data has different distribution from the source domain.

Therefore, the task of learning latent tree structures without human-annotated data, called *grammar induction* (Carroll and Charniak, 1992; Klein and Manning, 2002; Smith and Eisner, 2005), has become an important problem and attractd more attention from researchers recently. Prior work mainly focused on inducing tree structures from recurrent neural networks (Shen et al., 2018a,b) or recursive neural networks (Yogatama et al., 2017; Drozdov et al., 2019), while integrating tree structures into Transformer remains an unexplored direction.

Pre-training Transformer from large-scale raw texts successfully learns high-quality language representations. By further fine-tuning pre-trained Transformer on desired tasks, wide range of NLP tasks obtain the state-of-the-art results (Radford et al., 2019; Devlin et al., 2018; Dong et al., 2019). However, what pre-trained Transformer self-attention heads capture remains unknown. Although an attention can be easily explained by observing how words attend to each other, only some distinct patterns such as attending previous words or named entities can be found informative (Vig, 2019). The attention matrices do not match our intuitions about hierarchical structures.

In order to make the attention learned by Transformer more interpretable and allow Transformer to comprehend language hierarchically, we propose Tree Transformer, which integrates tree structures into bidirectional Transformer encoder. At each layer, words are constrained to attend to other words in the same constituents. This constraint has been proven to be effective in prior work (Wu et al., 2018). Different from the prior

[1]The source code is publicly available at https://github.com/yaushian/Tree-Transformer.

Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, pages 1061–1070, Hong Kong, China, November 3–7, 2019.

work that required a supervised parser, in Tree Transformer, the constituency tree structures is automatically induced from raw texts by our proposed "*Constituent Attention*" module, which is simply implemented by self-attention. Motivated by Tree-RNNs, which compose each phrase and the sentence representation from its constituent sub-phrases, Tree Transformer gradually attaches several smaller constituents into larger ones from lower layers to higher layers.

The contributions of this paper are 3-fold:

- Our proposed Tree Transformer is easy to implement, which simply inserts an additional "Constituent Attention" module implemented by self-attention to the original Transformer encoder, and achieves good performance on the unsupervised parsing task.
- As the induced tree structures guide words to compose the meaning of longer phrases hierarchically, Tree Transformer improves the perplexity on masked language modeling compared to the original Transformer.
- The behavior of attention heads learned by Tree Transformer expresses better interpretability, because they are constrained to follow the induced tree structures. By visualizing the self-attention matrices, our model provides the information that better matchs the human intuition about hierarchical structures than the original Transformer.

2 Related Work

This section reviews the recent progress about grammar induction. Grammar induction is the task of inducing latent tree structures from raw texts without human-annotated data. The models for grammar induction are usually trained on other target tasks such as language modeling. To obtain better performance on the target tasks, the models have to induce reasonable tree structures and utilize the induced tree structures to guide text encoding in a hierarchical order. One prior attempt formulated this problem as a reinforcement learning (RL) problem (Yogatama et al., 2017), where the unsupervised parser is an actor in RL and the parsing operations are regarded as its actions. The actor manages to maximize total rewards, which are the performance of downstream tasks.

PRPN (Shen et al., 2018a) and On-LSTM (Shen et al., 2018b) induce tree structures by introducing a bias to recurrent neural networks. PRPN proposes a parsing network to compute the syntactic distance of all word pairs, and a reading network utilizes the syntactic structure to attend relevant memories. On-LSTM allows hidden neurons to learn long-term or short-term information by the proposed new gating mechanism and new activation function. In URNNG (Kim et al., 2019b), they applied amortized variational inference between a recurrent neural network grammar (RNNG) (Dyer et al., 2016) decoder and a tree structures 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 same sentences are optimized to be close to each other. 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) in a corpus.

3 Tree Transformer

Given a sentence as input, Tree Transformer induces a tree structure. A 3-layer Tree Transformer is illustrated in Figure 1(A). The building blocks of Tree Transformer is shown in Figure 1(B), which is the same as those used in bidirectional Transformer encoder, except the proposed Constituent Attention module. The blocks in Figure 1(A) are constituents induced from the input sentence. The red arrows indicate the self-attention. The words in different constituents are constrained to not attend to each other. In the 0-th layer, some neighboring words are merged into constituents; for example, given the sentence "*the cute dog is wagging its tail*", the tree Transformer automatically determines that "*cute*" and "*dog*" form a constituent, while "*its*" and "*tail*" also form one. The two neighboring constituents may merge together in the next layer, so the sizes of constituents gradually grow from layer to layer. In the top layer, the layer 2, all words are grouped into the same constituent. Because all words are into the same constituent, the attention heads freely attend to any other words, in this layer, Tree Transformer behaves the same as the typical Transformer encoder. Tree Transformer can be trained in an end-to-end fashion by using "masked LM",

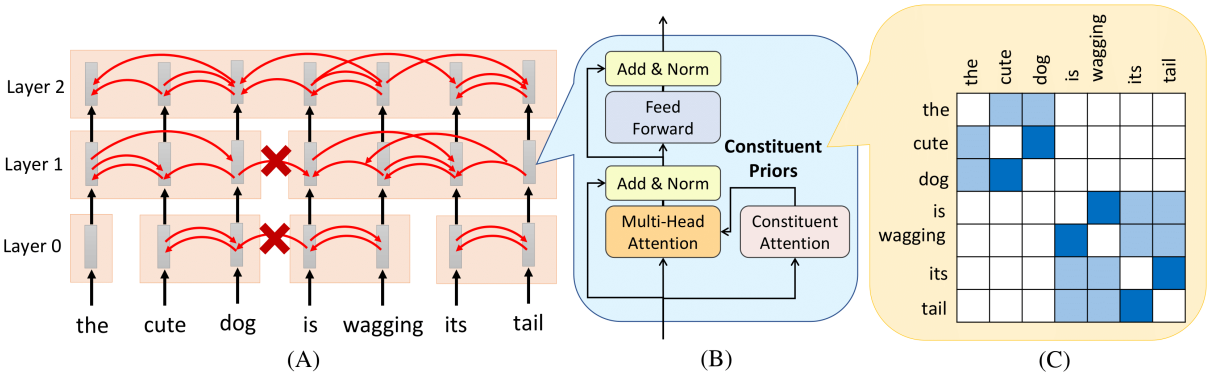


Figure 1: (A) A 3-layer Tree Transformer, where the blocks are constituents induced from the input sentence. The two neighboring constituents may merge together in the next layer, so the sizes of constituents gradually grow from layer to layer. The red arrows indicate the self-attention. (B) The building blocks of Tree Transformer. (C) Constituent prior C for the layer 1.

which is one of the unsupervised training task used for BERT training.

Whether two words belonging to the same constituent is determined by "*Constituent Prior*" that guides the self-attention. Constituent Prior is detailed in Section 4, which is computed by the proposed Constituent Attention module in Section 5. By using BERT masked language model as training, latent tree structures emerge from Constituent Prior and unsupervised parsing is thereby achieved. The method for extracting the constituency parse trees from Tree Transformer is described in Section 6.

4 Constituent Prior

In each layer of Transformer, there are a query matrix Q consisting of query vectors with dimension d_k and a key matrix K consisting of key vectors with dimension d_k. The attention probability matrix is denoted as E, which is an N by N matrix, where N is the number of words in an input sentence. $E_{i,j}$ is the probability that the position i attends to the position j. The Scaled Dot-Product Attention computes the E as:

$$E = \text{softmax}(\frac{QK^T}{d}), \tag{1}$$

where the dot-product is scaled by $1/d$. In Transformer, the scaling factor d is set to be $\sqrt{d_k}$.

In Tree Transformer, the E is not only determined by the query matrix Q and key matrix K, but also guided by Constituent Prior C generating from Constituent Attention module. Same as E, the constituent prior C is also a N by N matrix, where $C_{i,j}$ is the probability that word w_i and word w_j belong to the same constituency. This matrix is symmetric that $C_{i,j}$ is same as $C_{j,i}$. Each layer has its own Constituent Prior C. An example of Constituent Prior C is illustrated in Figure 1 (C), which indicates that in layer 1, "the cute dog" and "is wagging its tail" are two constituents.

To make each position not attend to the position in different constituents, Tree Transformer constrains the attention probability matrix E by constituent prior C as below,

$$E = C \odot \text{softmax}(\frac{QK^T}{d}), \tag{2}$$

where $\odot$ is the element-wise multiplication. Therefore, if $C_{i,j}$ has small value, it indicates that the positions i and j belong to different constituents, where the attention weight $E_{i,j}$ would be small. As Transformer uses multi-head attention with h different heads, there are h different query matrices Q and key matrices K at each position, but here in the same layer, all attention heads in multi-head attention share the same C. The multi-head attention module produces the output of dimension $d_{model} = h \times d_k$.

5 Constituent Attention

The proposed Constituent Attention module is to generate the constituent prior C. Instead of directly generating C, we decompose the problem into estimating the breakpoints between constituents, or the probability that two adjacent words belong to the same constituent. In each layer, the Constituent Attention module generates a sequence $a = \{a_1, ..., a_i, ..., a_N\}$, where a_i is the probability that the word w_i and its neighbor

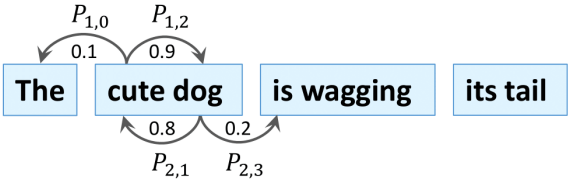


Figure 2: The example illustration about how neighboring attention works.

word w_{i+1} are in the same constituent. The small value of a_i implies that there is a breakpoint between w_i and w_{i+1}, so the constituent prior C is obtained from the sequence a as follows. $C_{i,j}$ is the multiplication of all $a_{i \leqslant k<j}$ between word w_i and word w_j:

$$C_{i,j} = \prod_{k=i}^{j-1} a_k. \tag{3}$$

In (3), we choose to use multiplication instead of summation, because if one of $a_{i \leqslant k<j}$ between two words w_i and w_j is small, the value of $C_{i,j}$ with multiplication also becomes small. In implementation, to avoid probability vanishing, we use log-sum instead of directly multiplying all a:

$$C_{i,j} = e^{\sum_{k=i}^{j-1} \log(a_k)}. \tag{4}$$

The sequence a is obtained based on the following two mechanisms: *Neighboring Attention* and *Hierarchical Constraint*.

5.1 Neighboring Attention

We compute the score $s_{i,i+1}$ indicating that w_i links to w_{i+1} by scaled dot-product attention:

$$s_{i,i+1} = \frac{q_i \cdot k_{i+1}}{d}, \tag{5}$$

where q_i is a link query vector of w_i with d_{model} dimensions, and k_{i+1} is a link key vector of w_{i+1} with d_{model} dimensions. We use $q_i \cdot k_{i+1}$ to represent the tendency that w_i and w_{i+1} belong to the same constituent. Here, we set the scaling factor d to be $\frac{d_{model}}{2}$. The query and key vectors in (5) are different from (1). They are computed by the same network architecture, but with different sets of network parameters.

For each word, we constrain it to either link to its *right neighbor* or *left neighbor* as illustrated in Figure 2. This constraint is implemented by applying a softmax function to two attention links of w_i:

$$p_{i,i+1}, p_{i,i-1} = \text{softmax}(s_{i,i+1}, s_{i,i-1}), \tag{6}$$

where $p_{i,i+1}$ is the probability that w_i attends to w_{i+1}, and $(p_{i,i+1} + p_{i,i-1}) = 1$. We find that without the constraint of the softmax operation in (6) the model prefers to link all words together and assign all words to the same constituency. That is, giving both $s_{i,i+1}$ and $s_{i,i-1}$ large values, so the attention head freely attends to any position without restriction of constituent prior, which is the same as the original Transformer. Therefore, the softmax function is to constraint the attention to be sparse.

As $p_{i,i+1}$ and $p_{i+1,i}$ may have different values, we average its two attention links:

$$\hat{a}_i = \sqrt{p_{i,i+1} \times p_{i+1,i}}. \tag{7}$$

The $\hat{a}_i$ links two adjacent words only if two words attend to each other. $\hat{a}_i$ is used in the next subsection to obtain a_i.

5.2 Hierarchical Constraint

As mentioned in Section 3, constituents in the lower layer merge into larger one in the higher layer. That is, once two words belong to the same constituent in the lower layer, they would still belong to the same constituent in the higher layer. To apply the hierarchical constraint to the tree Transformer, we restrict a_k^l to be always larger than a_k^{l-1} for the layer l and word index k. Hence, at the layer l, the link probability a_k^l is set as:

$$a_k^l = a_k^{l-1} + (1 - a_k^{l-1})\hat{a}_k^l, \tag{8}$$

where a_k^{l-1} is the link probability from the previous layer $l-1$, and $\hat{a}_k^l$ is obtained from Neighboring Attention (Section 5.1) of the current layer l. Finally, at the layer l, we apply (4) for computing C^l from a^l. Initially, different words are regarded as different constituents, and thus we initialize a_k^{-1} as zero.

6 Unsupervised Parsing from Tree Transformer

After training, the neighbor link probability a can be used for unsupervised parsing. The small value of a suggests this link be the breakpoint of two constituents. By top-down greedy parsing (Shen et al., 2018a), which recursively splits the sentence into two constituents with minimum a, a parse tree can be formed.

However, because each layer has a set of a^l, we have to decide to use which layer for parsing. Instead of using a from a specific layer for parsing

Algorithm 1 Unsupervised Parsing with Multiple Layers

$a \leftarrow$ link probabilities
$m \leftarrow minimum\ layer\ id$ ▷ Discard the a from layers below minimum layer
$thres \leftarrow 0.8$ ▷ Threshold of breakpoint
procedure BUILDTREE(l, s, e) ▷ l: layer index, s: start index, e: end index
  if $e - s < 2$ **then** ▷ The constituent cannot be split
    return (s, e)
  $span \leftarrow a^l_{s \le i < e}$
  $b \leftarrow$ **argmin**$(span)$ ▷ Get breakpoint
  $last \leftarrow$ **max**$(l-1, m)$ ▷ Get index of last layer
  if $a^l_b > thres$ **then**
    if $l = m$ **then**
      return (s, e)
    return BuildTree$(last, s, e)$
  $tree1 \leftarrow$ **BuildTree**$(last, s, b)$
  $tree2 \leftarrow$ **BuildTree**$(last, b+1, e)$
  return $(tree1, tree2)$ ▷ Return tree

(Shen et al., 2018b), we propose a new parsing algorithm, which utilizes a from all layers for unsupervised parsing. As mentioned in Section 5.2, the values of a are strictly increasing, which indicates that a directly learns the hierarchical structures from layer to layer. Algorithm 1 details how we utilize hierarchical information of a for unsupervised parsing.

The unsupervised parsing starts from the top layer, and recursively moves down to the last layer after finding a breakpoint until reaching the bottom layer m. The bottom layer m is a hyperparameter needed to be tuned, and is usually set to 2 or 3. We discard a from layers below m, because we find the lowest few layers do not learn good representations (Liu et al., 2019) and thus the parsing results are poor (Shen et al., 2018b). All values of a on top few layers are very close to 1, suggesting that those are not good breakpoints. Therefore, we set a threshold for deciding a breakpoint, where a minimum a will be viewed as a valid breakpoint only if its value is below the threshold. As we find that our model is not very sensitive to the threshold value, we set it to be 0.8 for all experiments.

7 Experiments

In order to evaluate the performance of our proposed model, we conduct the experiments detailed below.

7.1 Model Architecture

Our model is built upon a bidirectional Transformer encoder. The implementation of our Transformer encoder is identical to the original Transformer encoder. For all experiments, we set the hidden size d_{model} of Constituent Attention and Transformer as 512, the number of self-attention heads h as 8, the feed-forward size as 2048 and the dropout rate as 0.1. We analyze and discuss the sensitivity of the number of layers, denoted as L, in the following experiments.

7.2 Grammar Induction

In this section, we evaluate the performance of our model on unsupervised constituency parsing. Our model is trained on WSJ training set and WSJ-all (i.e. including testing and validation sets) by using BERT Masked LM (Devlin et al., 2018) as unsupervised training task. We use WordPiece (Wu et al., 2016) tokenizer from BERT to tokenize words with a 16k token vocabulary. Our best result is optimized by `adam` with a learning rate of 0.0001, $\beta_1 = 0.9$ and $\beta_2 = 0.98$. Following the evaluation settings of prior work (Htut et al., 2018; Shen et al., 2018b)[2], we evaluate F1 scores of our model on WSJ-test and WSJ-10 of Penn Treebank (PTB) (Marcus et al., 1993). The WSJ-10 has 7422 sentences from whole PTB with sentence length restricted to 10 after punctuation removal, while WSJ-test has 2416 sentences from the PTB testing set with unrestricted sentence length.

The results on WSJ-test are in Table 1. We mainly compare our model to PRPN (Shen et al., 2018a), On-lstm (Shen et al., 2018b) and Compound PCFG(C-PCFG) (Kim et al., 2019a), in which the evaluation settings and the training data are identical to our model. DIORA (Drozdov et al., 2019) and URNNG (Kim et al., 2019b) use a relative larger training data and the evaluation settings are slightly different from our model. Our model performs much better than trivial trees (i.e. right and left-branching trees) and random trees, which suggests that our proposed model successfully learns meaningful trees. We also find that

[2] https://github.com/yikangshen/Ordered-Neurons

Model	Data	$F1_{median}$	$F1_{max}$
PRPN	WSJ-train	35.0	42.8
On-lstm	WSJ-train	47.7	49.4
C-PCFG	WSJ-train	**55.2**	**60.1**
Tree-T,L=12	WSJ-train	48.4	50.2
Tree-T,L=10	WSJ-train	49.5	51.1
Tree-T,L=8	WSJ-train	48.3	49.6
Tree-T,L=6	WSJ-train	47.4	48.8
DIORA	NLI	**55.7**	**56.2**
URNNG	Billion	-	52.4
Tree-T,L=10	WSJ-all	50.5	52.0
Random	-	21.6	21.8
LB	-	9.0	9.0
RB	-	39.8	39.8

Table 1: **The F1 scores on WSJ-test.** Tree Transformer is abbreviated as Tree-T, and L is the number of layers(blocks). DIORA is trained on multi-NLI dataset (Williams et al., 2018). URNNG is trained on the subset of one billion words (Chelba et al., 2013) with 1M training data. LB and RB are the left and right-brancing baselines.

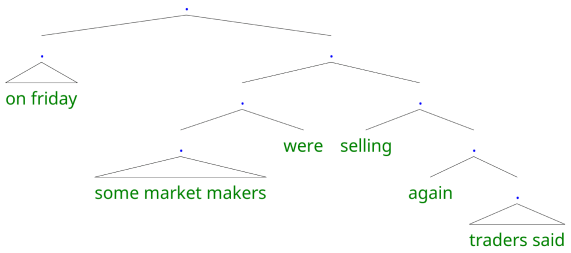


Figure 3: A parse tree induced by Tree Transformer. As shown in the figure, because we set a threshold in Algorithm 1, the leaf nodes are not strictly binary.

increasing the layer number results in better performance, because it allows the Tree Transformer to model deeper trees. However, the performance stops growing when the depth is above 10. The words in the layers above the certain layer are all grouped into the same constituent, and therefore increasing the layer number will no longer help model discover useful tree structures. In Table 2, we report the results on WSJ-10. Some of the baselines including CCM (Klein and Manning, 2002), DMV+CCM (Klein and Manning, 2005) and UML-DOP (Bod, 2006) are not directly comparable to our model, because they are trained using POS tags our model does not consider.

In addition, we further investigate what kinds of trees are induced by our model. Following URNNG, we evaluate the performance of constituents by its label in Table 3. The trees induced by different methods are quite different. Our model is inclined to discover noun phrases (NP) and adverb phrases (ADVP), but not easy to discover verb phrases (VP) or adjective phrases (ADJP). We show an induced parse tree in Figure 3 and more induced parse trees can be found in Appendix.

Model	Data	$F1_{median}$	$F1_{max}$
PRPN	WSJ-train	70.5	71.3
On-lstm	WSJ-train	65.1	66.8
C-PCFG	WSJ-train	70.5	-
Tree-T,L=10	WSJ-train	66.2	67.9
DIORA	NLI	67.7	68.5
Tree-T,L=10	WSJ-all	66.2	68.0
CCM	WSJ-10	-	71.9
DMV+CCM	WSJ-10	-	77.6
UML-DOP	WSJ-10	-	**82.9**
Random	-	31.9	32.6
LB	-	19.6	19.6
RB	-	56.6	56.6

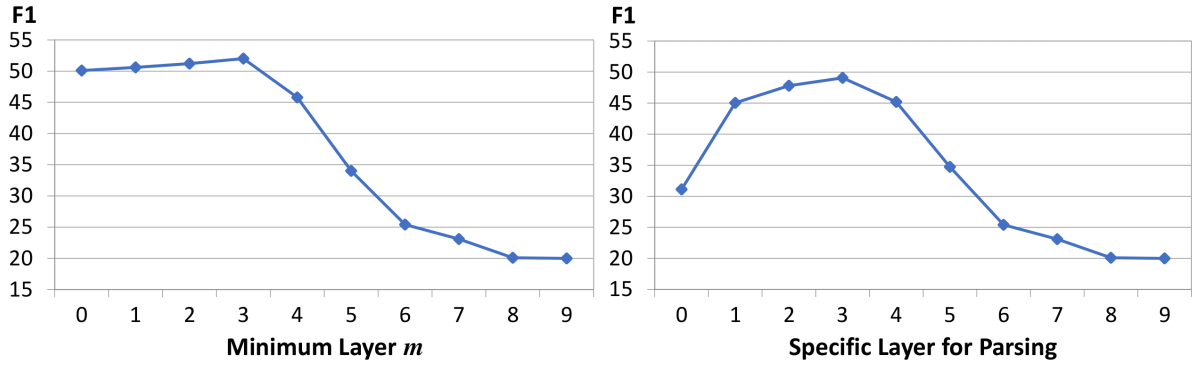
Table 2: **The F1 scores on WSJ-10.** Tree Transformer is abbreviated as Tree-T, and L is the number of layers (blocks).

Label	Tree-T	URNNG	PRPN
NP	**67.6**	39.5	63.9
VP	38.5	**76.6**	27.3
PP	52.3	**55.8**	55.1
ADJP	24.7	33.9	**42.5**
SBAR	36.4	**74.8**	28.9
ADVP	**55.1**	50.4	45.1

Table 3: **Recall of constituents by labels.** The results of URNNG and PRPN are taken from Kim et al. (2019b).

7.3 Analysis of Induced Structures

In this section, we study whether Tree Transformer learns hierarchical structures from layers to layers. First, we analyze the influence of the hyperparameter minimum layer m in Algorithm. 1 given the model trained on WSJ-all in Table 1. As illustrated in Figure 4(a), setting m to be 3 yields the best performance. Prior work discovered that the representations from the lower layers of Transformer are not informative (Liu et al., 2019). Therefore, using syntactic structures from lower layers decreases the quality of parse trees. On the other hand, most syntactic information is



(a) F1 in terms of different minimum layers.

(b) F1 of parsing via a specific layer.

Figure 4: Performance of unsupervised parsing.

missing when a from top few layers are close to 1, so too large m also decreases the performance.

To further analyze which layer contains richer information of syntactic structures, we evaluate the performance on obtaining parse trees from a specific layer. We use a^l from the layer l for parsing with the top-down greedy parsing algorithm (Shen et al., 2018a). As shown in Figure 4(b), using a^3 from the layer 3 for parsing yields the best F1 score, which is 49.07. The result is consistent to the best value of m. However, compared to our best result (52.0) obtained by Algorithm 1, the F1-score decreases by 3 ($52.0 \rightarrow 49.07$). This demonstrates the effectiveness of Tree Transformer in terms of learning hierarchical structures. The higher layers indeed capture the higher-level syntactic structures such as clause patterns.

7.4 Interpretable Self-Attention

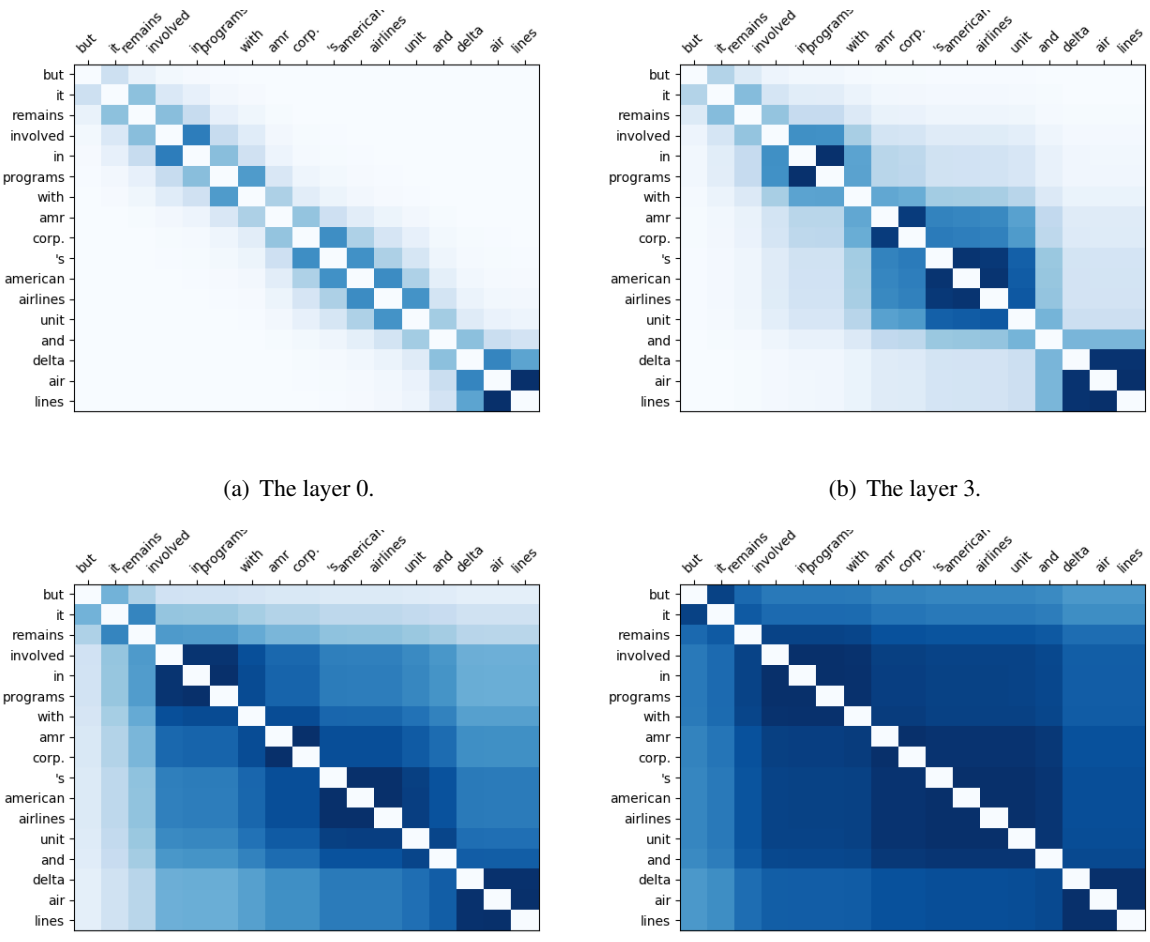
This section discusses whether the attention heads in Tree Transformer learn hierarchical structures. Considering that the most straightforward way of interpreting what attention heads learn is to visualize the attention scores, we plot the heat maps of Constituent Attention prior C from each layer in Figure 5.

In the heat map of constituent prior from first layer (Figure 5(a)), as the size of constituent is small, the words only attend to its adjacent words. We can observe that the model captures some subphrase structures, such as the noun phrase "*delta air line*" or "*american airlines unit*". In Figure 5(b)-(d), the constituents attach to each other and become larger. In the layer 6, the words from "*involved*" to last word "*lines*" form a high-level adjective phrase (ADJP). In the layer 9, all words are grouped into a large constituent except the first word "*but*". By visualizing the heat maps of the constituent prior from each layer, we can easily know what types of syntactic structures are learned in each layer. The parse tree of this example can be found in Figure 3 of Appendix. We also visualize the heat maps of self-attention from the original Transformer layers and one from the Tree Transformer layers in Appendix A. As the self-attention heads from our model are constrained by the constituent prior, compared to the original Transformer, we can discover hierarchical structures more easily.

Those attention heat maps demonstrate that: (1) the size of constituents gradually grows from layer to layer, and (2) at each layer, the attention heads tend to attend to other words within constituents posited in that layer. Those two evidences support the success of the proposed Tree Transformer in terms of learning tree-like structures.

Model	L	Params	Perplexity
Transformer	8	40M	48.8
Transformer	10	46M	48.5
Transformer	10-B	67M	49.2
Transformer	12	52M	48.1
Tree-T	8	44M	46.1
Tree-T	10	51M	45.7
Tree-T	12	58M	45.6

Table 4: **The perplexity of masked words.** Params is the number of parameters. We denote the number of layers as L. In Transformer $L = 10 - B$, the increased hidden size results in more parameters.



(a) The layer 0.

(b) The layer 3.

(c) The layer 6.

(d) The layer 9.

Figure 5: The constituent prior heat maps.

7.5 Masked Language Modeling

To investigate the capability of Tree Transformer in terms of capturing abstract concepts and syntactic knowledge, we evaluate the performance on language modeling. As our model is a bidirectional encoder, in which the model can see its subsequent words, we cannot evaluate the language model in a left-to-right manner. We evaluate the performance on masked language modeling by measuring the perplexity on masked words[3]. To perform the inference without randomness, for each sentence in the testing set, we mask all words in the sentence, but not at once. In each masked testing data, only one word is replaced with a "[MASK]" token. Therefore, each sentence creates the number of testing samples equal to its length.

In Table 4, the models are trained on WSJ-train with BERT masked LM and evaluated on WSJ-test. All hyperparameters except the number of layers in Tree Transformer and Transformer are set to be the same and optimized by the same optimizer. We use `adam` as our optimizer with learning rate of 0.0001, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Our proposed Constituent Attention module increases about 10% hyperparameters to the original Transformer encoder and the computational speed is 1.2 times slower. The results with best performance on validation set are reported. Compared to the original Transformer, Tree Transformer achieves better performance on masked language modeling. As the performance gain is possibly due to more parameters, we adjust the number of layers or increase the number of hidden layers in Transformer $L = 10 - B$. Even with fewer parameters than Transformer, Tree Transformer still performs bet-

[3]The perplexity of masked words is $e^{\frac{-\sum \log(p)}{n_{mask}}}$, where p is the probability of correct masked word to be predicted and n_{mask} is the total number of masked words.

ter.

The performance gain is because the induced tree structures guide the self-attention processes language in a more straightforward and human-like manner, and thus the knowledge can be better generalized from training data to testing data. Also, Tree Transformer acquires positional information not only from positional encoding but also from the induced tree structures, where the words attend to other words from near to distant (lower layers to higher layers)[4].

7.6 Limitations and Discussion

It is worth mentioning that we have tried to initialize our Transformer model with pre-trained BERT, and then fine-tuning on WSJ-train. However, in this setting, even when the training loss becomes lower than the loss of training from scratch, the parsing result is still far from our best results. This suggests that the attention heads in pre-trained BERT learn quite different structures from the tree-like structures in Tree Transformer. In addition, with a well-trained Transformer, it is not necessary for the Constituency Attention module to induce reasonable tree structures, because the training loss decreases anyway.

8 Conclusion

This paper proposes Tree Transformer, a first attempt of integrating tree structures into Transformer by constraining the attention heads to attend within constituents. The tree structures are automatically induced from the raw texts by our proposed Constituent Attention module, which attaches the constituents to each other by self-attention. The performance on unsupervised parsing demonstrates the effectiveness of our model in terms of inducing tree structures coherent to human expert annotations. We believe that incorporating tree structures into Transformer is an important and worth exploring direction, because it allows Transformer to learn more interpretable attention heads and achieve better language modeling. The interpretable attention can better explain how the model processes the natural language and guide the future improvement.

[4]We do not remove the positional encoding from the Transformer and we find that without positional encoding the quality of induced parse trees drops.

Acknowledgements

We would like to thank reviewers for their insightful comments. This work was financially supported from the Young Scholar Fellowship Program by Ministry of Science and Technology (MOST) in Taiwan, under Grant 108-2636-E002-003.

References

Roee Aharoni and Yoav Goldberg. 2017. Towards string-to-tree neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*.

Rens Bod. 2006. An all-subtrees approach to unsupervised parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*.

Glenn Carroll and Eugene Charniak. 1992. Two experiments on learning probabilistic dependency grammars from corpora. In *WORKING NOTES OF THE WORKSHOP STATISTICALLY-BASED NLP TECHNIQUES*.

C.Goller and A.Kuchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of International Conference on Neural Networks (ICNN'96)*.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*.

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*.

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. *arXiv preprint arXiv:1905.03197*.

Andrew Drozdov, Pat Verga, Mohit Yadav, Mohit Iyyer, and Andrew McCallum. 2019. Unsupervised latent tree induction with deep inside-outside recursive autoencoders. In *North American Association for Computational Linguistics*.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proc. of NAACL*.

Akiko Eriguchi, Yoshimasa Tsuruoka, and Kyunghyun Cho. 2017. Learning to parse and translate improves neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*.

Phu Mon Htut, Kyunghyun Cho, and Samuel Bowman. 2018. Grammar induction with neural language models: An unusual replication. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Association for Computational Linguistics.

Yoon Kim, Chris Dyer, and Alexander 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. *arXiv preprint arXiv:1904.03746*.

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

Dan Klein and Christopher D. Manning. 2005. Natural language grammar induction with a generative constituent-context model. *Pattern Recogn.*

Nelson F. Liu, Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith. 2019. Linguistic knowledge and transferability of contextual representations. *arXiv preprint arXiv:1903.08855*.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

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, Shawn Tan, Alessandro Sordoni, and Aaron Courville. 2018b. Ordered neurons: Integrating tree structures into recurrent neural networks. *arXiv preprint arXiv:1810.09536*.

Noah A. Smith and Jason Eisner. 2005. Guiding unsupervised grammar induction using contrastive estimation. In *In Proc. of IJCAI Workshop on Grammatical Inference Applications*.

Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng, and Christopher D. Manning. 2011. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*.

Emma Strubell, Patrick Verga, Daniel Andor, David I Weiss, and Andrew McCallum. 2018. Linguistically-informed self-attention for semantic role labeling. In *EMNLP*.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.

Jesse Vig. 2019. Visualizing attention in transformer-based language representation models. *arXiv preprint arXiv:1904.02679*.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*.

Wei Wu, Houfeng Wang, Tianyu Liu, and Shuming Ma. 2018. Phrase-level self-attention networks for universal sentence encoding. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3729–3738.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, and et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. 2017. Learning to compose words into sentences with reinforcement learning. In *International Conference on Learning Representations*.

Poorya Zaremoodi and Gholamreza Haffari. 2018. Incorporating syntactic uncertainty in neural machine translation with a forest-to-sequence model. In *Proceedings of the 27th International Conference on Computational Linguistics*.