Forming Trees with Treeformers

Nilay Patel
University of California, Santa Cruz
nilay@ucsc.edu

Jeffrey Flanigan
University of California, Santa Cruz
jmflanig@ucsc.edu

Abstract

Human language is known to exhibit a nested, hierarchical structure, allowing us to form complex sentences out of smaller pieces. However, many state-of-the-art neural networks models such as Transformers have no explicit hierarchical structure in their architecture—that is, they don't have an inductive bias toward hierarchical structure. Additionally, Transformers are known to perform poorly on compositional generalization tasks which require such structures. In this paper, we introduce Treeformer, a general-purpose encoder module inspired by the CKY algorithm which learns a composition operator and pooling function to construct hierarchical encodings for phrases and sentences. Our extensive experiments demonstrate the benefits of incorporating hierarchical structure into the Transformer and show significant improvements in compositional generalization as well as in downstream tasks such as machine translation, abstractive summarization, and various natural language understanding tasks.

1 Introduction

Human language is known to exhibit a nested or hierarchical structure (Chomsky, 1956; Montague, 1970). This structure allows humans to construct complex sentences from simple parts and is important for conveying meaning. For example, the phrase structure of the English sentence "The old man the boat." is critical for correctly determining its meaning (Figure 1).

Transformer models (Vaswani et al., 2017) are state-of-the-art across a wide variety of NLP tasks (Devlin et al., 2019), and pretrained Transformers have been shown to learn hierarchical structures after pretraining on large amounts of data (Lin et al., 2019; Rogers et al., 2020). However, Transformers do not have a hierarchical structure built into the architecture—that is, they don't have an inductive bias toward hierarchical structure (Tran et al., 2018). Additionally, Transformers are shown not to perform well on some compositional generalization tasks that require nested structure (Li et al., 2021).

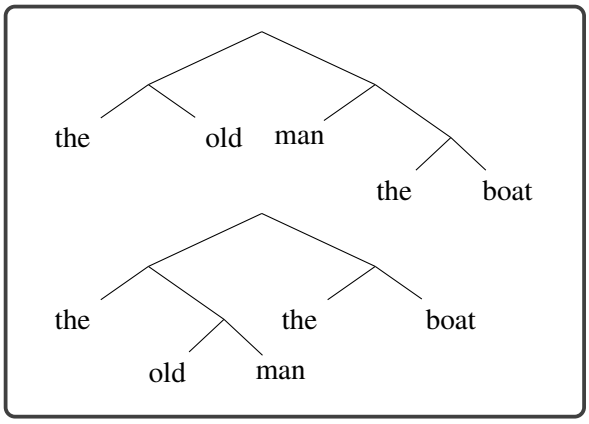

Figure 1: Two different parses of the text "the old man the boat" with significantly distinct meanings. While the top parse is a complete sentence (with "man" as a verb), the second is nonsense. Therefore, the encodings for the subphrase "the old man" (for example) in these parses should be significantly different.

We demonstrate that incorporating an inductive bias toward the hierarchical structure of language improves the performance of the Transformer on downstream tasks. We show that this improves compositional generalization and greatly improves the translation of predicated argument structure in machine translation. Specifically, we augment the Transformer to make it more compositional by adding a tree-encoder layer designed for modeling hierarchical phrases. Additionally, we show this layer improves downstream performance across a wide variety of tasks.

Our inductive bias layer, which we call **Treeformer**, is an encoder module that constructs hierarchical phrase encodings and is inspired by the CKY context-free-grammar parsing algorithm

Proceedings of Recent Advances in Natural Language Processing, pages 836–845
Varna, Sep 4–6, 2023
https://doi.org/10.26615/978-954-452-092-2_090

(Cocke, 1969; Younger, 1966; Kasami, 1965). To the best of our knowledge, this is the first study of adding a CKY-style phrase-structure inductive bias into a Transformer for compositional generalization and general-purpose supervised learning.

Prior work has used a similar CKY-style neural architecture for modeling unsupervised syntactic parsing (Drozdov et al., 2019; Xu et al., 2021b). These models are specific to unsupervised parsing and not directly applicable to supervised methods. In contrast, we focus on creating such an architecture for general-purpose supervised learning. Treeformer is also simpler than similar work such as DIORA (Drozdov et al., 2019), and faster due to two key optimizations which improve the complexity from cubic to linear time (see §4).

We demonstrate the effectiveness of adding a Treeformer module to the vanilla Transformer with experiments in compositional generalization (CG) on COGS (Kim and Linzen, 2020) and CoGnition, (Li et al., 2021), two challenging seq2seq datasets for testing CG. In addition, the addition of a Treeformer shows significant improvements in machine translation (Cettolo et al., 2012), abstractive summarization (Graff et al., 2003; Rush et al., 2015), and tasks in natural language understanding (Wang et al., 2018). Significantly, we find that the Treeformer is much better at correctly translating predicate-argument structures (subjects vs objects, etc). Predicate-argument structures require understanding the hierarchical structure of language and are very important for correctly conveying meaning. This demonstrates the benefits of the Treeformer architecture.

We leave to future work large-scale pretraining with our architecture. While interesting and important for practical considerations, pretraining is not within our computing budget, and we consider it out of scope for this work. Our focus is on advancements purely in model architecture.

The paper is organized as follows. First, we discuss some related work (§2). Then we present our Treeformer module (§3). We analyze the computational complexity and propose two methods for optimizing the algorithm (§4). After describing our experimental setups (§5), we present our results (§6) and finally conclude (§7).

2 Related Work

There is much prior work that induces, operates over, or otherwise uses a tree structure in neural network models (Socher et al., 2013a; Tai et al., 2015; Le and Zuidema, 2015; Dyer et al., 2016; Bradbury and Socher, 2017; Choi et al., 2017, 2018; Drozdov et al., 2019; Ahmed et al., 2019; Wang et al., 2019; Mrini et al., 2021; Hu et al., 2021; Yogatama et al., 2017; Sartran et al., 2022). Such models are especially of interest due to the prevalence of trees in natural language.

Tai et al. (2015) introduced Tree-LSTMs, an LSTM model generalized to work on parse trees. They suggest specific instances of the general Tree-LSTM architecture for particular types of trees such as dependency and constituency trees. However, Tree-LSTMs and many other tree- or graph-structured models (Nguyen et al., 2020; Wang et al., 2022; Shiv and Quirk, 2019; Harer et al., 2019; Sartran et al., 2022) require a parse tree over the input text, making data expensive or difficult to obtain. Unsupervised parsing methods (Maillard et al., 2017; Wang et al., 2019; Li et al., 2020; Drozdov et al., 2019) have been of interest to solve this problem, but mostly focus on parsing rather than downstream tasks as we do in this paper. One exception is the Gumbel Tree-LSTM Choi et al. (2017), which uses an unsupervised method to generate tree structures for classification tasks. The authors showed improvement on two tasks (Bowman et al., 2015; Socher et al., 2013b) at the time of writing, but they fall short of modern methods such as finetuning pretrained language models.

Most similar to our architecture is the work of Drozdov et al. (2019), who introduced Deep Inside-Outside Recursive Autoencoders (DIORA). DIORA learns tree structures using a modified inside-outside algorithm. The inside pass recursively generates a single root node, and the outside pass regenerates the leaf nodes from a root.

DIORA focuses on unsupervised parse tree induction and demonstrates a number of trees that closely match traditionally labeled ones, suggesting the composition algorithm learns efficacious information—a fact we rely on in this paper. Our Treeformer layer is similar to DIORA's "inside" pass but simpler and faster (see §3.2). Treeformer also has no "outside" pass as it does not need to regenerate the leaf nodes, but instead uses the encoded tree structure from the inside pass directly for downstream tasks.

3 Treeformer

The Treeformer algorithm generates phrase encodings by the repeated composition of a given set of token encodings. We start with n tokens (i.e., phrases of length 1) and their representations. We recursively apply the algorithm to compute representations of phrases of length k for all lengths k where $k \leq n$. Our approach, shown in Figure 2 and Algorithm 1, is inspired by the CKY algorithm.

Algorithm 1 Treeformer algorithm

Input: $s_{i,j}, \{r_{k,k} : \forall k, i \leq k \leq j\}$ ▷ Token encodings
Output: $r_{i,j}$

```
function FormTree(s_{i,j})
    if i = j then                                  ▷ Base case
        return r_{i,j}
    for k ← i to j do
        r_{i,k} ← FormTree(s_{i,k})                ▷ Recurse
        r_{k+1,j} ← FormTree(s_{k+1,j})
        r_k ← Comp(r_{i,k}, r_{k+1,j})             ▷ Compose
    r_{i,j} ← Pool(r_i, ... r_j)                   ▷ Pool
    return r_{i,j}
```

3.1 Notation

We now define some notation used throughout the rest of this paper. For input text s, let $s_{i,j}$ indicate the span of tokens starting at index i and ending at index j (inclusive), and let $r_{i,j}$ be the constructed representation of the span $s_{i,j}$. Finally, we use "phrase" and "span" interchangeably.

3.2 Algorithm

At a high level, our algorithm works as follows. The representation of a phrase is constructed by pooling representations of pairs of sub-phrases (see Figure 2). To build the representation of the phrase $s_{i,j}$, we consider all possible pairs of sub-phrases (**Collect children**), build a representation for each pair using a composition function (**Compose**), and finally pool these representations into one using an attention-based pooling operation (**Pool**).

More precisely, given a phrase $s_{i,j}$ of length $n = j - i$, we want to calculate the representation $r_{i,j}$ from its constituent subphrases. Figure 2 overviews our approach.

Collect children First, we gather each pair of complementary subphrases of $s_{i,j}$. For each index k such that $i \leq k < j$, we can split $s_{i,j}$ into a pair of subphrases $s_{i,k}$ (prefix) and $s_{k+1,j}$ (suffix). Let $R_{i,j}$ be the set containing the representations of each such pair:

$$R_{i,j} = \{(r_{i,k}, r_{k+1,j}) \ : \ i \leq k < j\}$$

Figure 3 shows the four such pairs of the input sentence $s_{1,5}$ = "I have the high ground". Note that these are exactly the set of pairs we would consider when parsing with the CKY algorithm.

Compose Next, we construct a set $C_{i,j}$ as the image of a *composition function* $\mathbf{Comp} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ on $R_{i,j}$. That is, it takes *pairs* of vectors and composes them into a single vector representing the concatenated span:

$$C_{i,j} = \{\mathbf{Comp}(r_k) \ : \ r_k \in R_{i,j}\}$$

Because the order of words and phrases in language matters, we want to retain non-commutativity, so this composition function should be non-commutative. A simple example would be concatenating the pair of vectors and feeding the result through a linear transformation. Indeed, Treeformer's composition function is exactly that:

$$\mathbf{Comp}(r_{i,k}, r_{k+1,j}) = \mathbf{W} \cdot [r_{i,k}, r_{k+1,j}] \quad (1)$$

where $\mathbf{W} \in \mathbb{R}^{2d \times d}$ and $[\cdot, \cdot]$ indicates concatenation. Thinking in terms of the CKY algorithm, composing two representations with **Comp** is the analogue of applying a grammatical rule.

Pool Finally, we pool the set $C_{i,j}$ into a single output vector $r_{i,j}$ via some *pooling function* **Pool**. A simple example would be an average or sum of the vectors, though these options treat all possible parses as equally valid. Treeformer's pooling function utilizes attention and a model parameter $w \in \mathbb{R}^d$. We calculate a weighted average of each $c_k \in C_{i,j}$ using scaled dot-product attention to w:

$$r_{i,j} = \sum_{c_k \in C_{i,j}} \text{softmax}\left(\frac{\mathbf{K}c_k \cdot \mathbf{Q}w}{\sqrt{d}}\right) c_k \quad (2)$$

At this point in the CKY algorithm, we'd be able to precisely determine our set of valid pairs and eliminate the others using the non-terminals and allowable grammar rules. However, it's not so straightforward to do so with untyped, approximate representations such as vectors. The pooling function is meant do so by extracting only pertinent information from each pair of nodes, each of which represents a possible parse.

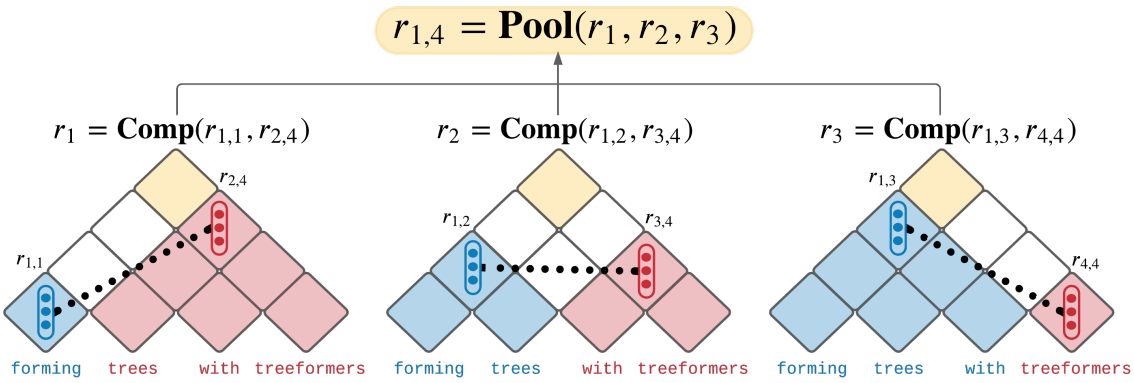

Figure 2: A demonstration of how the phrase "forming trees with treeformers" is encoded. First, we consider each pair of complementary subphrases (each chart represents a different pair). Next, for each pair, we compose their representations using a composition function **Comp** into an intermediate representation r_k. Finally, we pool the intermediate representations into a single vector via some function **Pool**.

I have the high ground
I have the high ground
I have the high ground
I have the high ground

Figure 3: All prefix and suffix pairs of the phrase "I have the high ground". We might guess the split "I have" and "the high ground" is the correct parse, but the model considers a weighted average of all parses.

Use in Downstream Tasks For seq2seq tasks, inserting the Treeformer module is simple. We feed the output of the encoder into the Treeformer and use the result as the memory for cross-attention in the decoder. For sequence classification tasks, we average the top row of the Treeformer output and add the result to the `[CLS]` token representation from the pretrained Transformer (e.g., ALBERT).

Comparison to DIORA It is useful to compare the Treeformer architecture to DIORA's inside pass (Drozdov et al., 2019). DIORA uses a Tree-LSTM or MLP as the composition function, which we simplify to concatenation followed by a linear projection, which is equivalent to two linear projections added together. This is faster to compute because the linear projections can be precomputed in $O(n)$ and reused, rather than the $O(n^2)$ computations for DIORA. Additionally, our pooling function is simplified when compared to DIORA's bilinear compatibility function, which allows us to use linearity to precompute the majority of the computationally expensive operations in our pooling function in $O(n)$ time rather than $O(n^2)$ for DIORA's compatibility function.

4 Parallelization

The CKY algorithm, which uses a similar chart structure to Treeformer, has a worst-case runtime complexity of $\mathcal{O}(n^3|G|)$ where $|G|$ is the size of the context-free grammar. Similarly, the Treeformer encoding algorithm is also $\mathcal{O}(n^3)$ assuming constant model dimension and sequential operations. In this section, we show this calculation as well as two key optimizations which are necessary for tractable training and improve the time and space complexity to $\mathcal{O}(n)$ and $\mathcal{O}(nmH)$, respectively. See §6.8 for empirical results.

Sequential Algorithm Starting with a sequence of length n, we encode phrases of length h for $1 \leq h \leq n$. There are $n - h + 1$ phrases of length h, each having $h - 1$ pairs of children. Each pair will be composed together exactly once in the entire algorithm, giving us

$$\sum_{h=1}^{n}(n - h + 1)(h - 1) = \mathcal{O}(n^3) \qquad (3)$$

total compositions. As our composition function runs in constant time (with respect to n), our total complexity for compositions is $\mathcal{O}(n^3)$. For pooling, we have $\mathcal{O}(n^2)$ total nodes each with $\mathcal{O}(n)$ pairs of children each. Since the scaled dot-product attention scales linearly in its arguments, we again get a complexity of $\mathcal{O}(n^3)$ for pooling and thus for the entire algorithm as well.

Parallel Algorithm While encoding phrases of length h is dependent on the encodings for all lengths less than h, there is no dependency on other phrases of the same length, allowing us to compute them in parallel. Parallelization removes the factor of $n - h + 1$ in Equation 3, leaving

$$\sum_{h=1}^{n}(h-1) = \mathcal{O}(n^2) \tag{4}$$

total compositions. Likewise, we can pool $\mathcal{O}(n)$ sets of children in parallel, reducing the pooling (and thus overall) parallel complexity to $\mathcal{O}(n^2)$.

Limiting Tree Height In practice, the space complexity turns out to be a bottleneck. Decoding involves calculating and storing cross attention to $\mathcal{O}(n^2)$ vectors (compared to $\mathcal{O}(n)$ for Transformers) for each of the m tokens in the output, resulting in a space complexity of $\mathcal{O}(n^2m)$. To reduce this, we introduce a hyperparameter H which limits the maximum tree height (or phrase length). This results in $\mathcal{O}(n)$ and $\mathcal{O}(nmH)$ complexities, respectively. Surprisingly, this optimization is not harmful to the model's effectiveness and is possibly even beneficial (see appendix). We find a value of $H = 10$ gives the best performance in general, so we use that for all experiments.

5 Experiments

We conduct experiments in five settings: (1) English-Chinese machine translation for CG on CoGnition (Li et al., 2021), (2) semantic parsing for CG on COGS (Kim and Linzen, 2020), (3) machine translation on IWSLT'14 German-English and English-French (Cettolo et al., 2012), (4) abstractive summarization on GigaWord English abstractive summarization (Graff et al., 2003), and (5) five natural language understanding tasks selected from GLUE (Wang et al., 2018). For full experimental details, see appendix. Models referred to as "Treeformer" are a Transformer with a Treeformer module, as described in the last paragraph in § 3.2.

We test our models on two compositional generalization datasets: CoGnition (Li et al., 2021), an English-Chinese machine translation dataset designed to test CG abilities, and COGS (Kim and Linzen, 2020), a semantic parsing dataset. These datasets are specifically designed to test a model's ability to generalize compositionally by testing its ability to generalize to novel combinations of predicates and arguments.

A Note About Baselines Although there is much prior work on tree structures in deep learning, we are not aware of any prior work using tree structures that is suitable as a baseline for our tasks beyond the Transformer. Models such as DIORA (Drozdov et al., 2019) and related models are for unsupervised parsing but not for classification or seq2seq tasks such as the ones we consider here. Gumbel Tree-LSTMs (Choi et al., 2017) similarly are only for classification and not for seq2seq. Transformer Grammars (Sartran et al., 2022) and RNNGs are for parsing or language modeling (Dyer et al., 2016), or for classification (Yogatama et al., 2017). All the above architectures would require significant changes for seq2seq tasks.

6 Results

6.1 Translation

Table 1 shows the results on IWSLT'14 German-English and English-French translation. Compared to the baseline Transformer, our model improves by 0.9 and 0.5 BLEU points over a 6-layer Transformer, and by 0.5 and 0.3 over a Transformer with a 7-layer encoder (which notably has more parameters than the Treeformer). For German-English, we also report scores from DynamicConv (Wu et al., 2019) and their reported baseline (also a Transformer), compared to which our model improves by 0.2 and 1.0 points respectively.

6.2 Abstractive Summarization

For the summarization task, Treeformer improves by a significant 1.6, 0.9, and 0.6 points in ROUGE-1, ROUGE-2, and ROUGE-L, respectively, compared to the baseline (Table 2).

6.3 GLUE

Treeformer matches or improves performance on four of five selected GLUE tasks, notably making a significant improvement on CoLA with a 5.1 point increase (Table 3). Intuitively, we expect Treeformer to perform well on single-sentence tasks more so than sentence pair tasks since phrases that span both sentences would likely be meaningless. This is reflected in our results as Treeformer performs well on both CoLA and SST-2. These results indicate despite rich contextual token encodings, Transformers are not capturing beneficial phrase-level information.

Model	Parameters	De-En	En-Fr
Transformer (Wu et al., 2019)	37M	34.4	-
DynamicConv (Wu et al., 2019)	-	35.2	-
Transformer	37M	34.5	41.0
Transformer (7-layer encoder)	42M	34.9	41.2
Treeformer ($H = 10$)	40M	**35.4**	**41.5**
BiBERT (state of the art)		38.6	-

Table 1: Model performance (BLEU) on the IWSLT'14 German-English and English-French translation tasks. Models we trained (highlighted in grey) used six layer encoders and decoders and dimensions $d_{model} = 512$ and $d_{ffn} = 1024$. For comparison, we also report the (to the best of our knowledge) state-of-the-art for De-En (Xu et al., 2021a).

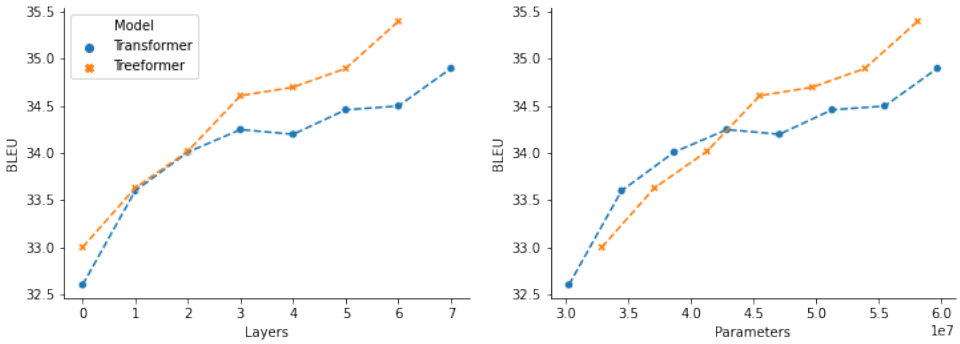

Figure 4: Effects of including a Treeformer module on-top of a Transformer with respect to the number of layers (left) and parameters (right). Although the Treeformer module is less efficient in shallower models, its efficacy grows as the underlying encoder grows larger. With more layers, it becomes more parameter-efficient to add a Treeformer module than adding more Transformer layers. Models are trained and evaluated on IWSLT'14 De-En.

6.4 Compositional Generalization

On the CoGnition CG test set (Table 4), Treeformer attains a significant 4.2% and 5.9% decrease in instance-level and aggregate-level compound error rates respectively (averaged over three runs).

On COGS, Treeformer improves over the Transformer by 1.6% percentage points. Our results on both datasets indicate the hierarchical structure is especially useful for generalization tasks while simultaneously improving other downstream tasks.

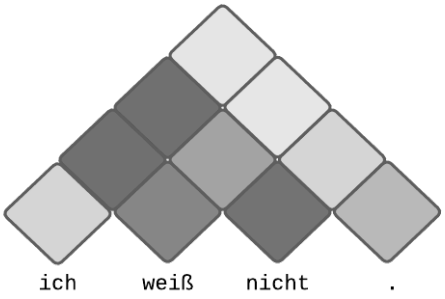

Figure 5: Heat-map of the cross-attention weights for the Treeformer averaged over each layer, head, and output position (darker is higher).

6.5 Effects of Model Size

Figure 4 shows a comparison of the Transformer with and without a Treeformer module at various encoder depths (left) and their respective parameter counts (right). In each case, simply adding a Treeformer module is beneficial, especially in deeper models. Importantly, the Treeformer module becomes more parameter-efficient than further encoder layers as the base model deepens. This fact implies it is not simply extra parameters improving performance, but rather that the Treeformer module is capturing useful information otherwise lost.

6.6 Analysis of Treeformer Attention

In some cases, despite no supervision for parsing, we see the decoder cross-attends to constituent phrases identified by linguists (Figure 5). Similarly, we can generate "parse trees" by choosing the pair with the highest attention weight at each step in

Model	Parameters	ROUGE-1	ROUGE-2	ROUGE-L
Transformer	73M	37.1	17.7	34.8
Treeformer ($H = 10$)	75M	**38.7**	**18.6**	**35.4**
Pegasus+DotProd (state of the art)	568M	40.45	20.69	36.56

Table 2: Model performance (ROUGE) on the Gigaword abstractive summarization task. Bold values indicate the highest performance for each metric. We include the current (to the best of our knowledge) state-of-the-art (Kedia et al., 2021).

GLUE Task	CoLA	MNLI (m/mm)	MRPC	SST-2	STS-B	Avg.
ALBERT	56.4	84.9 / 85.1	**88.9 / 92.0**	91.9	**90.4 / 90.7**	85.0
ALBERT+Treeformer	**61.5**	**85.4 / 85.5**	88.4 / 91.6	**92.4**	**90.4 / 90.7**	**85.7**

Table 3: Model performance on selected GLUE tasks. ALBERT is the `albert-base-v2` pretrained model from Huggingface's `Transformers` library, fine-tuned on these five tasks. We add a Treeformer as described in §3.2

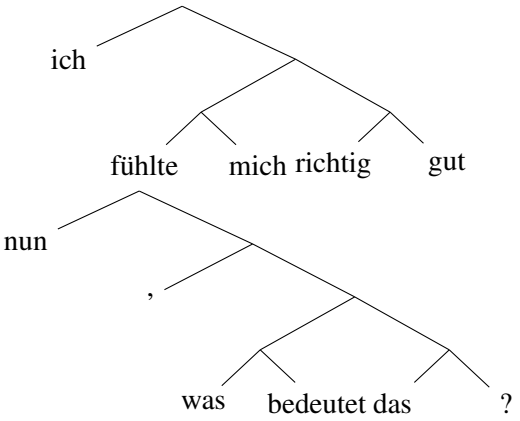

Figure 6: Example German parses from the model trained on IWSLT'14. Despite no explicit training, the resulting trees are visually plausible.

the algorithm. In Figure 6, we see two such parses which seem visually plausible despite no explicit supervision. However, we find in most cases the generated trees are not linguistically plausible, and do not have high parsing accuracy when evaluated as parse trees. Nevertheless, the improvement in performance we see across tasks, especially for CG and predicate-argument structure in MT, suggests that the information in the phrase-level vectors is useful for understanding the hierarchical structure of language.

6.7 Treeformer Captures Predicate-Argument Structure

To better understand where Treeformer improves over a vanilla Transformer, we conduct a human analysis on 50 randomly selected examples from the IWSLT'14 De/En validation set (Table 5). We find the Treeformer greatly reduces the frequency of errors in predicate-argument structure (e.g., swapping subject and object, or the example in Table 6). Of the categories of errors we analyzed, correctly translating predicate-argument structure requires the most understanding of the hierarchical structure and is very important for correctly conveying the meaning. This demonstrates the benefit of the Treeformer approach.

6.8 Speed Comparison

Our optimizations (§4) make training Treeformer tractable, but the architecture is slower than the vanilla Transformer due to the sequential nature of the algorithm and the increase in total encoded vectors. We measure the encoder-only speed at various sequence lengths for both models (Figure 7).

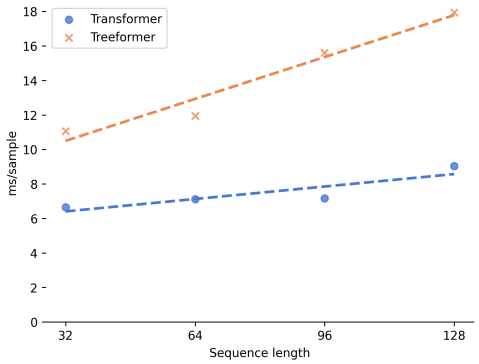

Figure 7: A comparison of training speed (ms/sample) by sequence length. The Treeformer is about 50%-60% as fast as the Transformer. For shorter sequences, Treeformer is about 60% as fast, which decreases to about 50% for longer sequences.

Model	CoGnition (Inst/Agg. ER) ↓	COGS (Acc.) ↑
Transformer	29.0/64.3%	78.5%
Treeformer	**24.8/58.5%**	**80.1%**
T5+CSL-Aug (Qiu et al., 2021)	-	99.5%
R-Dangle (Zheng and Lapata, 2022)	16.0/42.1%	-

Table 4: Results on the CoGnition COGS datasets. In both cases, the Treeformer makes significant improvements in generalization ability. For comparison, we also report state-of-the-art for both tasks.

Model	Transformer	Treeformer
Correct	22	**23**
Lexical	25	**22**
Pred-Arg	7	**2**
Morphosyntax	9	**7**
Drop/Add	**3**	4
Other	1	1
Total Errors	42	**34**

Table 5: Counts from a human analysis of 50 randomly sampled sentences from IWSLT'14 De/En, categorized by translation error type. The Treeformer greatly reduces errors in predicate-argument structures, demonstrating the benefit of modeling hierarchical structure. The error types are: correct = correct translation, lexical = incorrect lexical choice, pred-arg = incorrect predicate-argument structure (e.g., swapping subjects and objects), morphosyntax = morphosyntactic errors (e.g., incorrect inflections, tense, number, or determiners), drop/add = missing or incorrectly added tokens, other = other errors. Note: sentences can have multiple errors.

Input	also ging ich von da an weiter.
Transformer	so i went from there to further.
Treeformer	so i went on from there.
Gold	so i moved on from there.

Table 6: An example from the IWSLT'14 validation set in which the vanilla Transformer makes a predicate-argument error which the addition of the Treeformer avoids.

7 Conclusion

This paper presents Treeformer, a CKY-inspired neural network algorithm for composing tokens into phrases and sentences. We showed that, in many cases, standard Transformers are unable to effectively capture the phrase-level or hierarchical information which the Treeformer module helps exploit. This information allows the Treeformer to outperform a vanilla Transformer in compositional generalization and many downstream tasks, including machine translation, abstractive summarization, and natural language understanding.

We believe hierarchical structure is an important feature for models to have due to the prevalence of tree structures in natural language, and we are further convinced by the performance increase shown with our Treeformer module across a variety of settings. While this paper and many previous works modify algorithms such as CKY to induce tree structures, this approach can be slow and resource intensive due to the number of parses which must be computed. We believe improving speed, memory, and performance in tree-level neural models is possible and an important avenue for future research.

References

Mahtab Ahmed, Muhammad Rifayat Samee, and Robert E. Mercer. 2019. You only need attention to traverse trees. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 316–322, Florence, Italy. Association for Computational Linguistics.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv: Arxiv-1508.05326*.

James Bradbury and Richard Socher. 2017. Towards neural machine translation with latent tree attention. In *Proceedings of the 2nd Workshop on Structured Prediction for Natural Language Processing*, pages 12–16, Copenhagen, Denmark. Association for Computational Linguistics.

Mauro Cettolo, Christian Girardi, and Marcello Federico. 2012. WIT3: Web inventory of transcribed and translated talks. In *Proceedings of the 16th Annual conference of the European Association for Machine Translation*, pages 261–268, Trento, Italy. European Association for Machine Translation.

Jihun Choi, Kang Min Yoo, and Sang goo Lee. 2017. Learning to compose task-specific tree structures. *Aaai Conference On Artificial Intelligence*.

Jihun Choi, Kang Min Yoo, and Sang-goo Lee. 2018. Learning to compose task-specific tree structures. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

N. Chomsky. 1956. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124.

John Cocke. 1969. *Programming Languages and Their Compilers: Preliminary Notes*. New York University, USA.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *North American Chapter Of The Association For Computational Linguistics*.

Andrew Drozdov, Pat Verga, Mohit Yadav, Mohit Iyyer, and Andrew McCallum. 2019. Unsupervised latent tree induction with deep inside-outside recursive autoencoders. *arXiv preprint arXiv: Arxiv-1904.02142*.

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

David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. 2003. English gigaword. *Linguistic Data Consortium, Philadelphia*, 4(1):34.

Jacob Harer, Chris Reale, and Peter Chin. 2019. Tree-transformer: A transformer-based method for correction of tree-structured data. *arXiv preprint arXiv: Arxiv-1908.00449*.

Xiang Hu, Haitao Mi, Zujie Wen, Yafang Wang, Yi Su, Jing Zheng, and Gerard de Melo. 2021. R2d2: Recursive transformer based on differentiable tree for interpretable hierarchical language modeling. *arXiv preprint arXiv: Arxiv-2107.00967*.

Tadao Kasami. 1965. An efficient recognition and syntax-analysis algorithm for context-free languages.

Akhil Kedia, Sai Chetan Chinthakindi, and Wonho Ryu. 2021. Beyond reptile: Meta-learned dot-product maximization between gradients for improved single-task regularization. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 407–420, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Najoung Kim and Tal Linzen. 2020. COGS: A compositional generalization challenge based on semantic interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9087–9105, Online. Association for Computational Linguistics.

Phong Le and Willem Zuidema. 2015. Compositional distributional semantics with long short term memory. In *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics*, pages 10–19, Denver, Colorado. Association for Computational Linguistics.

Bowen Li, Taeuk Kim, Reinald Kim Amplayo, and Frank Keller. 2020. Heads-up! unsupervised constituency parsing via self-attention heads. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 409–424, Suzhou, China. Association for Computational Linguistics.

Yafu Li, Yongjing Yin, Yulong Chen, and Yue Zhang. 2021. On compositional generalization of neural machine translation. *Annual Meeting Of The Association For Computational Linguistics*.

Yongjie Lin, Yi Chern Tan, and Robert Frank. 2019. Open sesame: Getting inside bert's linguistic knowledge. *arXiv preprint arXiv:1906.01698*.

Jean Maillard, Stephen Clark, and Dani Yogatama. 2017. Jointly learning sentence embeddings and syntax with unsupervised tree-lstms. *arXiv preprint arXiv: Arxiv-1705.09189*.

Richard Montague. 1970. Universal grammar. *Theoria*, 36(3):373–398.

Khalil Mrini, Emilia Farcas, and Ndapa Nakashole. 2021. Recursive tree-structured self-attention for answer sentence selection. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4651–4661, Online. Association for Computational Linguistics.

Xuan-Phi Nguyen, Shafiq Joty, Steven Hoi, and Richard Socher. 2020. Tree-structured attention with hierarchical accumulation. In *International Conference on Learning Representations*.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Pawel Krzysztof Nowak, Tal Linzen, Fei Sha, and Kristina Toutanova. 2021. Improving compositional generalization with latent structure and data augmentation. *North American Chapter Of The Association For Computational Linguistics*.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in bertology: What we know about how bert works. *Transactions of the Association for Computational Linguistics*, 8:842–866.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Laurent Sartran, Samuel Barrett, Adhiguna Kuncoro, Miloš Stanojević, Phil Blunsom, and Chris Dyer. 2022. Transformer grammars: Augmenting transformer language models with syntactic inductive biases at scale. *Transactions of the Association for Computational Linguistics*, 10:1423–1439.

Vighnesh Shiv and Chris Quirk. 2019. Novel positional encodings to enable tree-based transformers. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013a. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013b. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

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

Ke Tran, Arianna Bisazza, and Christof Monz. 2018. The importance of being recurrent for modeling hierarchical structure. *arXiv preprint arXiv: Arxiv-1803.03585*.

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*. Curran Associates, Inc.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv: Arxiv-1804.07461*.

Wenhan Wang, Kechi Zhang, Ge Li, Shangqing Liu, Anran Li, Zhi Jin, and Yang Liu. 2022. Learning program representations with a tree-structured transformer. *arXiv preprint arXiv: Arxiv-2208.08643*.

Yau-Shian Wang, Hung-Yi Lee, and Yun-Nung Chen. 2019. Tree transformer: Integrating tree structures into self-attention. *arXiv preprint arXiv: Arxiv-1909.06639*.

Felix Wu, Angela Fan, Alexei Baevski, Yann N. Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. *arXiv preprint arXiv: Arxiv-1901.10430*.

Haoran Xu, Benjamin Van Durme, and Kenton Murray. 2021a. Bert, mbert, or bibert? a study on contextualized embeddings for neural machine translation. *arXiv preprint arXiv: Arxiv-2109.04588*.

Zhiyang Xu, Andrew Drozdov, Jay Yoon Lee, Timothy J. O'Gorman, Subendhu Rongali, Dylan Finkbeiner, S. Suresh, Mohit Iyyer, and A. McCallum. 2021b. Improved latent tree induction with distant supervision via span constraints. *EMNLP*.

Dani Yogatama, Chris Dyer, Wang Ling, and Phil Blunsom. 2017. Generative and discriminative text classification with recurrent neural networks. *arXiv preprint arXiv: Arxiv-1703.01898*.

Daniel H. Younger. 1966. Context-free language processing in time n3. In *7th Annual Symposium on Switching and Automata Theory (swat 1966)*, pages 7–20.

Hao Zheng and Mirella Lapata. 2022. Real-world compositional generalization with disentangled sequence-to-sequence learning. *ArXiv*, abs/2212.05982.