Unifying Parsing and Tree-Structured Models for Generating Sentence Semantic Representations

Antoine Simoulin[1,2] **Benoît Crabbé**[1]
[1]University of Paris, LLF [2]Quantmetry
asimoulin@quantmetry.com
benoit.crabbe@u-paris.fr

Abstract

We introduce a novel tree-based model that learns its composition function together with its structure. The architecture produces sentence embeddings by composing words according to an induced syntactic tree. The parsing and the composition functions are explicitly connected and, therefore, learned jointly. As a result, the sentence embedding is computed according to an interpretable linguistic pattern and may be used on any downstream task. We evaluate our encoder on downstream tasks, and we observe that it outperforms tree-based models relying on external parsers. In some configurations, it is even competitive with BERT base model. Our model is capable of supporting multiple parser architectures. We exploit this property to conduct an ablation study by comparing different parser initializations. We explore to which extent the trees produced by our model compare with linguistic structures and how this initialization impacts downstream performance. We empirically observe that downstream supervision troubles producing stable parses and preserving linguistically relevant structures.

1 Introduction

Computing sentence semantic representations traditionally calls for a recursive compositional function whose structure is tree-shaped. There is a strong intuition in natural language processing that language has a recursive structure (Chomsky, 1956; Shen et al., 2019). Tree-based models should thus mimic the compositional effect of language and enable better generalization and abstraction.

Yet, tree-based models need carefully hand-annotated data to be trained. Alternative methods such as recurrent neural network (Hochreiter and Schmidhuber, 1997; Cho et al., 2014) or BERT (Devlin et al., 2019) have gained increased popularity as they only require raw text as input. On the other hand, as many results suggest (Linzen et al., 2016; Jawahar et al., 2019; Clark et al., 2019), these new models acquire some sort of tree structure.

Another line of work, called *latent tree learning*, induces trees from raw text and computes semantic representations along with the inferred structure (Socher et al., 2011; Bowman et al., 2016; Dyer et al., 2016; Maillard et al., 2019; Yogatama et al., 2017; Kim et al., 2019). Such methods preserve explicit recursive computation and produce intelligible tree structures. Moreover, the parser and composition function are learned jointly and are specific to a given task or domain. Choi et al. (2018) propose the closest approach to ours by composing a tree using the Gumbel-Softmax estimator. The method is fully differentiable, produces a discrete tree, and does not require training the parser using an auxiliary task. However, Williams et al. (2018) show the method does not produce meaningful syntactic representations and that trees are inconsistent across initializations. Moreover, Choi et al. (2018) produces trees by selecting and merging adjacent nodes. Therefore, it cannot directly use architectures designed for standard parsing formalisms such as dependency structures.

We propose a unified architecture, which infers an explicit tree structure and recursively trains a sentence embedding model. Our method is fully differentiable and relies on existing and well-known components. We use a standard dependency parsing structure, obtained using a graph-based biaffine dependency parser (Dozat and Manning, 2017). However, our model is not limited to a particular parser architecture as long as it is differentiable.

We organize our paper as follows: we present our model in section 2. In section 3, we evaluate our model on textual entailment and semantic similarity tasks. We then conduct an ablation study and analyze the impact of the parser initialization. We compare the learned structures across initializations and with interpretable annotations (4.1) and we study how latent structures impact performance on downstream tasks (4.2).

2 Model

Our model jointly performs sentence parsing and the prediction of a sentence embedding. The sentence embedding is predicted by a weighted TREE-LSTM whose tree structure is provided by a dependency parser. The TREE-LSTM recursive composition function crucially uses a weighted sum of the child representations whose weights are provided by the parser edges, hence linking the parser outputs to the TREE-LSTM recursion. Figure 1 illustrates the architecture detailed in Eq. 1 to 10.

Parsing model The parser is a standard graph based biaffine dependency parser[1] (Dozat and Manning, 2017). It is formalized in two steps. First, Eq. 1 and 2 compute a weight matrix that is interpreted as a weighted directed graph whose nodes are the sentence tokens:

$$a_k^{(dep)} = \mathrm{MLP}(h_k), \quad a_j^{(head)} = \mathrm{MLP}(h_j) \tag{1}$$

$$s_{kj}^{(arc)} = (a_k^{(dep)} \oplus 1)^\top U^{(b)} a_j^{(head)} + b^{(b)} \tag{2}$$

With $h_k \in \mathbb{R}^d$ the hidden state associated with the word at index k in the input sentence and in Eq. 2, $U^{(b)} \in \mathbb{R}^{(d+1)\times d}$ and $b^{(b)} \in \mathbb{R}$. The symbol $\oplus$ denotes vector concatenation and MLP in Eq. 1 are single-layer perceptron networks.

The second step performs parsing by computing a maximum spanning tree from the graph. As in Dozat and Manning (2017), we use the Max Spanning Tree (MST) algorithm to ensure the well-formedness of the tree (Chu, 1965; Edmonds et al., 1967):

$$\alpha_{kj} = \mathbb{1}_{\mathrm{mst}(s_{kj}^{(arc)})} s_{kj}^{(arc)} \tag{3}$$

Where α_{kj} is the probability of the edge linking node j to node k. For a given node k, there is at most one non-zero edge leading to its governor j.

Compositionally weighted tree LSTM Given a predicted tree structure, we recursively encode the sentence using a variant of the Child Sum Tree model from Tai et al. (2015). The recursion follows the predicted structure: from the leaves to the root. At each step j, the transition function takes as input the word vector representation x_j of the head node j and the previously computed hidden states h_k from all its children.

$$\tilde{h}_j = \sum_{k\in C(j)} \alpha_{kj} h_k, \tag{4}$$

$$i_j = \sigma\left(W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)}\right), \tag{5}$$

$$o_j = \sigma\left(W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)}\right), \tag{6}$$

$$u_j = \tanh\left(W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)}\right), \tag{7}$$

$$f_{jk} = \sigma\left(W^{(f)} x_j + U^{(f)} h_k + b^{(f)}\right), \tag{8}$$

$$c_j = i_j \odot u_j + \sum_{k\in C(j)} f_{jk} \odot c_k, \tag{9}$$

$$h_j = o_j \odot \tanh(c_j), \tag{10}$$

Where x_j and h_j are respectively the word vector representation and hidden state associated with the head node j. In Eq. 4, $C(j)$ denotes the set of children of node j. σ denotes the logistic sigmoid function and $\odot$ denotes elementwise multiplication. Crucially, in our case, Eq. 4 is a weighted sum rather than a standard sum and the weights are those α_{kj} provided by the parser.

We use the embedding computed by the weighted TREE-LSTM at the root of the tree as

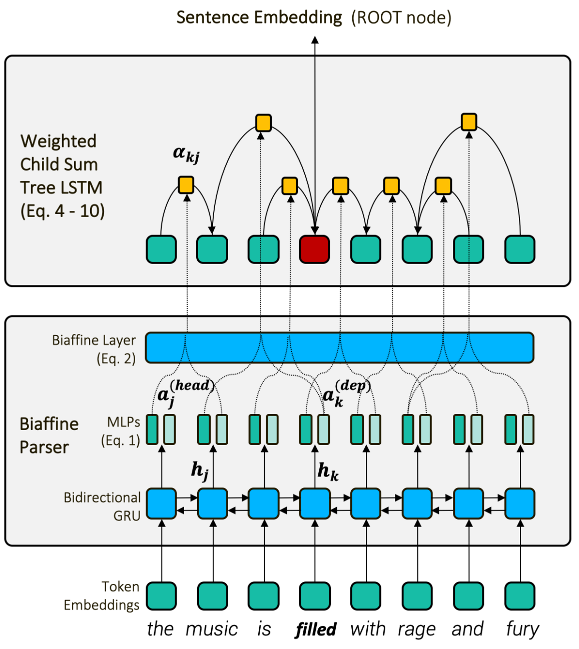


Figure 1: We illustrate the architecture detailed in Eq. 1 to 10. The Biaffine parser provides the sentence structure from which the TREE-LSTM computes sentence embeddings. The full pipeline is differentiable as the TREE-LSTM weights are given by the parser.

[1]We give hyper-parameter details for the biaffine parser in Appendix A.3.

the sentence embedding. The tree shape and the edge weights are given by the best prediction of a graph parser. The parsing model is linked to the TREE-LSTM by the weights α_{kj}. This architecture allows us to jointly update the parser and the TREE-LSTM weights using only the downstream task loss. The supervision comes only from the objective of the downstream task, and no intermediate structure target is required. Our model is fully differentiable and preserves the discreteness of the tree composition process. It relies on a dependency parsing formalism and could accommodate any differentiable parser.

3 Evaluation

Our architecture primarily aims to produce relevant embeddings for downstream tasks. To this end, we compare our setup with other models from the literature on various tasks. For this comparison, we first pre-train the parsing submodel on human-annotated sentences from the Penn Tree Bank (PTB) (Marcus et al., 1993) converted to Stanford dependencies. We then fine-tune the parser's parameters on the task while training the full model[2].

3.1 Semantic textual similarity (STS)

We first evaluate our model on the SICK-R downstream task (Marelli et al., 2014), which is dedicated to assessing models' compositional properties. The dataset comprises 9,927 sentence pairs, distributed in a 4,500/500/4,927 train/dev/test split, annotated for semantic similarity on a 1 to 5 real range. It includes specific examples of variations on passive and active forms, quantifier and modifier switches, or negations[3].

We use a similar training procedure as in Tai et al. (2015). We transform the target y from the SICK-R task into the distribution p defined by:

$$p_i = \begin{cases} y - \lfloor y \rfloor, & i = \lfloor y \rfloor + 1 \\ \lfloor y \rfloor - y + 1, & i = \lfloor y \rfloor \\ 0 & \text{otherwise} \end{cases}$$

We use a dedicated architecture to predict the similarity distribution from a pair of sentences. The similarity module takes as input a pair of sentence vectors h_L and h_R and computes their component-wise product $h_L \odot h_R$ and their absolute difference $|h_L - h_R|$. Given these features, we compute the probability distribution $\hat{p}_\theta$ using a two-layer perceptron network (MLP):

$$\begin{aligned} h_\times &= h_L \odot h_R, \quad h_+ = |h_L - h_R|, \\ h_s &= \sigma(W^{(\times)} h_\times + W^{(+)} h_+ + b^{(h)}), \\ \hat{p}_\theta &= \text{softmax}(W^{(p)} h_s + b^{(p)}), \end{aligned} \tag{11}$$

We use the KL-divergence between the prediction $\hat{p}_\theta$ and the ground truth p as training objective:

$$J(\theta) = \frac{1}{N} \sum_{k=1}^{N} \text{KL}(p^{(k)} || \hat{p}_\theta^{(k)}) + \lambda ||\theta||_2^2 \tag{12}$$

Finally during inference, the similarity score $\hat{y}$ is computed as $\hat{y} = r^\top \hat{p}_\theta$ with $r^\top = [1, \ldots, 5]$.

Encoder	r
BOW†	78.2 (1,1)
LSTM†	84.6 (0.4)
Bidirectional LSTM†	85.1 (0.4)
N-ary TREE-LSTM† (Tai et al., 2015)	85.3 (0.7)
Childsum TREE-LSTM† (Tai et al., 2015)	86.5 (0.4)
BERT-base (Devlin et al., 2019)	87.3 (0.9)
Unified TREE-LSTM† (Our model)	87.0 (0.3)

Table 1: Evaluation on the SICK-R task: we pre-train our parsing module on the PTB and continue to update the full model on the SICK-R task. We compare with BERT and models relying on sequential and tree structures. We report Pearson correlation on the test set, by convention as $r \times 100$ (avg. and std. from 5 runs). † indicates models that we trained. All models are trained following the same procedure detailed in Appendix A.1.

Table 1 reports the results from the test set. As expected, structured models perform better than models using weaker underlying structures. We also observe that our model is competitive with a BERT-base upper-line. It is essential to note that BERT models are heavily pre-trained on vast corpora, whereas our structured models are trained only on the SICK-R and PTB data.

3.2 Textual entailment

We also test our model on the Stanford Natural Language Inference (SNLI) task (Bowman et al., 2015), which includes 570k pairs of sentences with

[2]In this configuration, we observe pre-training the parser may cause weights α to become too large in Eq. 3. This leads to poor downstream performance. We correct this with a multiplicative parameter τ whose value is estimated during training. It means we replace Eq. 3 with: $\alpha_{kj} = \tau \cdot \mathbb{1}_{mst(s_{kj}^{(arc)})} s_{kj}^{(arc)}$ for tree weights computation.

[3]Appendix A.1 details the hyper-parameters and training infrastructure.

the labels entailment, contradiction, and neutral, distributed in a 550k/10k/10k train/dev/test split[4].

We use a similar training procedure as in Choi et al. (2018). A dedicated architecture is used to predict the similarity distribution from a pair of sentences. The similarity module takes as input a pair of sentence vectors h_L and h_R and computes their componentwise product $h_L \odot h_R$ and their absolute difference $|h_L - h_R|$. Given these features, we compute the probability distribution $\hat{p}_\theta$ using a three-layer perceptron network (MLP):

$$\begin{aligned} h_\times &= h_L \odot h_R, \quad h_+ = |h_L - h_R|, \\ h_s &= h_\times \oplus h_+ \oplus h_L \oplus h_R \\ h_s &= \mathrm{ReLU}(W^{(1)} h_s + b^{(1)}), \\ h_s &= \mathrm{ReLU}(W^{(2)} h_s + b^{(2)}), \\ \hat{p}_\theta &= \mathrm{softmax}(W^{(p)} h_s + b^{(p)}), \end{aligned} \tag{13}$$

We use the cross entropy loss between the prediction $\hat{p}_\theta$ and the ground truth p as training objective:

$$J(\theta) = -\frac{1}{N} \sum_{k=1}^{N} p^{(k)} \log \hat{p}_\theta^{(k)} + \lambda ||\theta||_2^2 \tag{14}$$

Encoder	Test Acc.
SPINN \w Reinforce (Yogatama et al., 2017)	80.5
CYK and TREE-LSTM (Maillard et al., 2019)	81.6
SPINN (Bowman et al., 2016)	83.2
ST-Gumbel (Choi et al., 2018)	86.0
Structured Alignment (Liu et al., 2018)	86.3
BERT-base (Zhang et al., 2020)	90.7
Unified TREE-LSTM (Our model)	85.0 (0.2)

Table 2: Evaluation on the SNLI-R task: We pre-train our parsing module on the PTB and continue to update the full model on the SNLI task. We compare with BERT and latent tree learning models. We report the accuracy on the test set (avg. and std. from 2 runs).

We report the results in Table 2. Our results are close to Choi et al. (2018), which also compute semantic representations along to discrete tree structures but relies on a distinct syntactic formalism. In models from Liu et al. (2018) and Zhang et al. (2020) sentences are encoded with direct interaction using an attention mechanism. These architectures relying on cross sentence attention outperform those without. We hypothesize that, on this textual entailment task, the final prediction cannot be directly deduced from both sentence embeddings. In this case, BERT and the structured alignment model have a clear advantage since they encode interactions between both sentences.

[4]Appendix A.2 details the hyper-parameters and training infrastructure.

4 Impact of the parser initialization

Our framework primarily aims to be a structured sentence encoder. Accordingly, we have demonstrated in the previous section that our architecture is competitive with comparable approaches and might even be competitive with BERT-based models. We are also interested in interpreting the structures the model actually learns and how such structures impact downstream performance.

In the previous section, we pre-trained the parser on human annotated data. However, the optimal structure might differ from the task. Moreover, for computational reasons, it might even differ significantly from linguistic insights. In this section we perform an ablation study to better understand how the initialization of the parser impacts the resulting structures (4.1) and the final downstream performance (4.2). We define two initialization scenarios below. In both, we either continue to update the parser when fine-tuning the model on downstream tasks or freeze the parser and only train the TREE-LSTM. These two configurations are indicated with respectively ✓ and × symbols.

Linguistic annotations Tree-structured models traditionally rely on linguistic structures obtained by parsers (Tai et al., 2015). For languages such as English, linguistic resources are available; it is technically possible to pre-train the parser. However, resources such as the PTB are not available in all languages. To better quantify the benefits of using linguistic annotations, we propose the following configurations, using various proportions of the PTB to initialize the parser:

- In the **PTB-All** configuration, the parser is previously pre-trained on the PTB. This configuration is the same as in section 3.
- In the **PTB-$\emptyset$** configuration, the parser parameters are randomly initialized
- We also consider an initialization with only a small proportion of the **PTB** and train a parser by only using 100 randomly selected samples. This configuration is referred as **PTB-100**.

Unsupervised structures Many lines of work investigate if attention matrices from large pre-trained models reflect syntactic structures (Jawahar et al., 2019; Clark et al., 2019; Ravishankar et al., 2021) or if tree structures can be integrated into transformers (Wang et al., 2019; Bai et al., 2021).

Since our model is not specific to any parser architecture. It is possible to use the internal representations from BERT to infer sentence structure.

BERT relies upon the self-attention mechanism. Inside each layer, tokens are computed as a weighted combination from each other. For each token x, a query and key vector are computed using a linear transformation detailed in Eq 15. Given these vector tuples, the attention weights s are computed following Eq 16 in which N refers to the dimension of the query and key vectors.

$$q_j, k_j = W^{(q,k)} x_j + b^{(q,k)} \tag{15}$$

$$s_{kj} = \text{softmax}\left(\frac{k_k \cdot q_j}{\sqrt{N}}\right) \tag{16}$$

We induce a tree structure following a procedure close from Ravishankar et al. (2021). We interpret the combination weights s as a weighted graph whose nodes are tokens. We then apply Eq 2 to induce a maximum spanning tree from the attention matrix as detailed in section 2. We make use of the last layer and induce a tree for each attention head taken separately[5]. Given the tree structure induced from BERT, we apply our TREE-LSTM model detailed in Eq. 4 to 10. In this configuration, we only use BERT as an unsupervised parser to infer a sentence structure. The semantic composition along with the structure to produce a sentence embedding is solely computed by the weighted TREE-LSTM.

4.1 Impact on parses

This section analyzes to which extent the structures generated by our model are comparable with meaningful linguistic annotations. We compare the parses generated by two distinct models differing by their initialization on the development set of both tasks. Our reference is the silver parses from the PTB-All configuration, where the parser is previously pre-trained on the full PTB and not updated during training.

Table 3 measures the Unlabeled Attachment Score (UAS) between the two parsers, that is, the ratio from the number of common arcs between two parses by the total number of arcs[6].

Parser 1	Parser 2	SICK-R (dev UAS)	SNLI (dev UAS)
Impact of parser fine-tuning			
PTB-100 (✓)	PTB-100 (×)	85.2 (1.5)	5.6 (1.9)
PTB-All (✓)	PTB-All (×)	98.4 (0.1)	11.7 (0.9)
Impact of the PTB sample size			
PTB-100 (✓)	PTB-∅ (✓)	6.3 (0.0)	10.1 (10.7)
PTB-All (✓)	PTB-∅ (✓)	10.1 (0.0)	15.1 (15.4)
PTB-All (✓)	PTB-100 (✓)	76.9 (0.7)	0.3 (0.2)
Unsupervised parser			
BERT (×)	PTB-All (×)	—	13.0 (4.9)
BERT (✓)	PTB-All (×)	—	13.7 (2.7)

Table 3: Impact of the parser initialization on parses: we compare the parses from the SICK-R and SNLI development sets using different parser initializations. We obtained the PTB parses with the graph parser initialized on a given proportion of the PTB (section 2). Regarding BERT , we inferred the structures from the pattern learn by the pre-trained model (section 4). We either continue to update the parser (✓) when fine-tuning the model on downstream tasks or freeze the parser (×) and only train the TREE-LSTM. UAS corresponds to the mean pairwise comparison of two configurations between two runs (std. in parentheses).

We observed distinct behaviors given both tasks. We believe this effect is due to the differences between training configurations. In particular, we use the Adagrad optimizer for the SICK-R task and Adam for the SNLI task.

For the SICK-R task, the UAS between PTB-∅ and PTB-All are very low. This reveals that the parses obtained with only downstream task supervision have few in common with gold linguistic parses. In this regard, we share the observation from Williams et al. (2018) that latent trees obtained from sole downstream supervision are not meaningful in syntax. However, PTB-All and PTB-100 are remarkably close; only a few PTB samples are needed to obtain intelligible linguistic parses with our setup. Regarding the PTB-100 configuration, we note an evolution of the parses when fine-tuning on the downstream task. We hypothesize that the model can adapt to the dataset's specificity.

For the SNLI task, fine-tuning the parser deeply impacts the shape of the parses. Depending from the initialization, parses will converge to distinct structures. Indeed, the UAS between all configura-

[5] We give details about the hyper-parameters in Appendix A.4.

[6] We present some parse tree examples in Appendix A.5.

tions is very low. Moreover, when using a random initialization (PTB-∅), the standard deviation between UAS from various runs is very high: without fixed initialization, parses become unstable.

For the initialization with an unsupervised structure, we only evaluate our setup on the SNLI task, which has more training samples. We compare the structures obtained with BERT with the silver trees from the PTB-All-× configuration. We present the mean UAS over the trees obtained for all attention heads. The standard deviation is relatively high, pointing underlying structures differ given the attention head. Nonetheless, self-supervised structures do not align well with linguistic insights. When updating BERT together with the TREE-LSTM, the UAS increases while the standard deviation decreases. As BERT is fine-tuned, structures tend to become more standard and present slightly more similarities with linguistic patterns.

4.2 Impact on downstream tasks

We observed in previous section 4.1 that the initialization and the training configuration of the parser component deeply impact the resulting parses. We now study the impact of the parser initialization on downstream performance.

PTB sample size	Parser fine-tuning	SICK-R (r)	SNLI (Acc.)
Linguistic annotations			
PTB-∅	✓	85.6 (85.6)	84.6 (85.5)
PTB-100	×	86.4 (86.6)	84.5 (85.5)
PTB-100	✓	86.5 (86.9)	84.9 (85.8)
PTB-All	×	86.8 (87.2)	85.0 (85.8)
PTB-All	✓	87.0 (87.5)	85.0 (85.5)
Unsupervised parser			
BERT	×	—	84.4 (85.3)
BERT	✓	—	84.6 (85.1)

Table 4: Impact of the parser initialization on downstream task performance: We pre-train the parser module with a given sample size from the PTB. We either freeze (×) or update (✓) the parser during the fine-tuning. We report the average test score set from 5 runs for SICK-R and 2 runs for SNLI (the score from the development set are in parentheses). We report Pearson correlation by convention as $r \times 100$.

Table 4 compares the impact of the different initializations for both tasks. We report the Pearson correlation on the test set of the SICK-R task and the accuracy on the test set from the SNLI task.

We either freeze the parser component or continue to update it, given the downstream loss for each initialization. Fine-tuning the parser on the task generally leads to an improvement of the downstream results. In that regard, we share the observation from other latent tree learning methods (Maillard et al., 2019; Choi et al., 2018); models jointly learning the parsing and composition function outperform those with a fixed structure.

Models using the full or partial annotated data outperform models relying on the sole downstream supervision (PTB-∅), in particular on the SICK-R task. We previously observed that fine-tuning the parser can lead to tree structure diverging from linguistic patterns. Nonetheless, regarding the downstream performance, human annotation appears as a good initialization for our model.

Models relying on linguistic-driven structures seem to achieve better performance. Nonetheless, the difference is thin, and we present here an average score across trees obtained from all attention heads. Therefore some attention heads might present structures as efficient as linguistic patterns.

5 Conclusion and future work

We investigate the relevance of incorporating tree-like structural bias in the context of sentence semantic inference. To this end, we formulate an original model for learning tree structure with distant downstream supervision. Our model is based on well-known components and could therefore accommodate a variety of parsing architectures such as graph parsers or attention matrices from BERT.

We evaluate our model on textual entailment and semantic similarity tasks and outperform sequential models and tree-structured models relying on external parsers. Moreover, when initialized on human-annotated structures, our model improves inference close to BERT base performance on the semantic similarity task.

We then conduct an ablation study to quantify the impact of the parser initialization on the resulting structures and downstream performance. We corroborate that the sole use of downstream supervision is insufficient to produce parses that are easy to interpret. To encourage convergence towards readable linguistic structures, we examine a number of initialization setups. Our structures often converge toward trivial branching patterns, which have few in common with gold linguistic parses. Yet, regarding downstream performance, linguistic insights appear as a relevant initialization.

References

Jiangang Bai, Yujing Wang, Yiren Chen, Yaming Yang, Jing Bai, Jing Yu, and Yunhai Tong. 2021. Syntax-bert: Improving pre-trained transformers with syntax trees. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*, pages 3011–3020. 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. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 632–642. The Association for Computational Linguistics.

Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. 2016. A fast unified model for parsing and sentence understanding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734.

Jihun Choi, Kang Min Yoo, and Sang-goo Lee. 2018. Learning to compose task-specific tree structures. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 5094–5101. AAAI Press.

Noam Chomsky. 1956. Three models for the description of language. *IRE Trans. Inf. Theory*, 2(3):113–124.

Yoeng-Jin Chu. 1965. On the shortest arborescence of a directed graph. *Scientia Sinica*, 14:1396–1400.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does BERT look at? an analysis of bert's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@ACL 2019, Florence, Italy, August 1, 2019*, pages 276–286. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

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

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 199–209. The Association for Computational Linguistics.

Jack Edmonds et al. 1967. Optimum branchings. *Journal of Research of the national Bureau of Standards B*, 71(4):233–240.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 3651–3657. Association for Computational Linguistics.

Yoon Kim, Alexander M. Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. 2019. 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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 1105–1117. Association for Computational Linguistics.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of lstms to learn syntax-sensitive dependencies. *Trans. Assoc. Comput. Linguistics*, 4:521–535.

Yang Liu, Matt Gardner, and Mirella Lapata. 2018. Structured alignment networks for matching sentences. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1554–1564. Association for Computational Linguistics.

Jean Maillard, Stephen Clark, and Dani Yogatama. 2019. Jointly learning sentence embeddings and syntax with unsupervised tree-lstms. *Nat. Lang. Eng.*, 25(4):433–449.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. 2014. A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC 2014, Reykjavik, Iceland, May 26-31, 2014*, pages 216–223. European Language Resources Association (ELRA).

Vinit Ravishankar, Artur Kulmizev, Mostafa Abdou, Anders Søgaard, and Joakim Nivre. 2021. Attention can reflect syntactic structure (if you let it). In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*, pages 3031–3045. Association for Computational Linguistics.

Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron C. Courville. 2019. Ordered neurons: Integrating tree structures into recurrent neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Haoyue Shi, Hao Zhou, Jiaze Chen, and Lei Li. 2018. On tree-based neural sentence modeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 4631–4641. Association for Computational Linguistics.

Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. 2011. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John McIntyre Conference Centre, Edinburgh, UK, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 151–161. ACL.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1556–1566. The Association for Computer Linguistics.

Yau-Shian 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 2019, Hong Kong, China, November 3-7, 2019*, pages 1061–1070. Association for Computational Linguistics.

Adina Williams, Andrew Drozdov, and Samuel R. Bowman. 2018. Do latent tree learning models identify meaningful structure in sentences? *Trans. Assoc. Comput. Linguistics*, 6:253–267.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. 2017. Learning to compose words into sentences with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Zhuosheng Zhang, Yuwei Wu, Hai Zhao, Zuchao Li, Shuailiang Zhang, Xi Zhou, and Xiang Zhou. 2020. Semantics-aware BERT for language understanding. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 9628–9635. AAAI Press.

A Appendices

A.1 SICK-R training configuration

Hyper-parameters We set the hyperparameters given literature on the domain, in particular regarding choices made in Tai et al. (2015). For all experiments detailed in the current section, the batch size is fixed to 25, weight decay to $1e^{-4}$ and gradient clipping set to 5.0. The learning rate is set to 0.025 for the TREE-LSTM parameters. When using a pre-training procedure for the parser, we set the learning rate to $5e^{-3}$ and use the following warm-up: for the first epoch, the parser is frozen. For the following epochs, all parameters are trained. At each epoch, the parser learning rate is divided by a factor of two. Without pre-training, the learning rate is set to $5e^{-4}$ for the parser. All model weights are initialized with a Xavier distribution. The hidden size of the similarity architecture is set to 50. The TREE-LSTM hidden size is set to 150. We use the Adagrad optimizer. We do not apply any dropout. We perform training for a maximum of 20 epochs and stop when no improvement was observed on the development set for 3 consecutive epochs. Regarding the vocabulary, we limit the size to 20,000 words and initialize the embeddings layer with 300-dimensional GloVe embeddings[7]. The embeddings are not updated during training.

Training infrastructure We trained all models on a single 1080 Ti Nvidia GPU. Training time for each epoch is approximately 1 minute. The model counts 13.7M parameters. Data can be downloaded using the SentEval package[8].

A.2 SNLI training configuration

Hyper-parameters We set the hyper-parameters given literature on the domain, in particular regarding choices made in Choi et al. (2018). For all experiments detailed in section 3.2, the batch size is fixed to 128, weight decay to 0, and gradient clipping set to 5.0. The learning rate is set to $1e^{-3}$ for the TREE-LSTM and the parser. The hidden size of the similarity architecture is set to 1,024. The TREE-LSTM hidden size is set to 600. We use the Adam optimizer. We apply a 0.2 dropout within the similarity architecture. We perform training for a maximum of 20 epochs and stop when no improvement was observed on the development set for 3 consecutive epochs. Regarding the vocabulary, we limit the size to 100,000 words and initialize the embeddings layer with 300-dimensional GloVe embeddings. The embeddings are not updated during training.

Training infrastructure We trained all models on a single 1080 Ti Nvidia GPU. Training time for each epoch is approximately 2h30 hours. The model counts 13.7M parameters. Data can be downloaded using the SentEval package[9].

A.3 Model Architecture

Regarding the biaffine parser, all parameters are chosen given Dozat and Manning (2017) recommendations. We use a hidden size of 150 for the MLPs layers and 100 for the biaffine layer. The dropout rate is fixed to 0.33. We use an open-source implementation of the parser and replace the pos-tags features with character level features. Therefore we don't need pos-tags annotations to parse our corpus[10]. We encode words using 100-dimensional GloVe embedding and a character embedding size of 50. Word vectors are then fed to a bidirectional LSTM with 3 layers of size 400.

A.4 BERT unsupervised parsing

When using BERT to perform unsupervised parsing, we use the implementation of BERT-base model from the Transformers library[11]. When fine-tuning the parser component, we set the learning rate to $2e^{-5}$ When fine-tuning BERT parser, each epoch takes around 5 hours on the SNLI. Without fine-tuning, this time is reduced to 90 minutes.

A.5 Visualization of the parses

We illustrate the effect summarize on Table 3 on some random examples. Figures from the first column (2a, 2c and 2e) show the parses obtained without updating the parser component on the downstream task. Figures from the second column(2b, 2d and 2f) show the evolution of the parses for the same initialization but after fine-tuning the parser on the SNLI task. Figures from the first row (2a and 2b) are initialized using the full PTB, the second row (2c and 2d) is initialized using 100 PTB

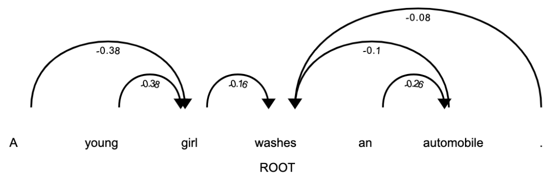
[7]https://nlp.stanford.edu/projects/glove/

[8]https://github.com/facebookresearch/SentEval

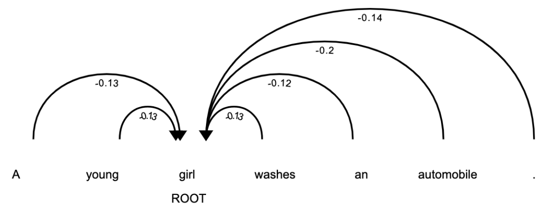
[9]https://github.com/facebookresearch/SentEval

[10]https://github.com/yzhangcs/biaffine-parser

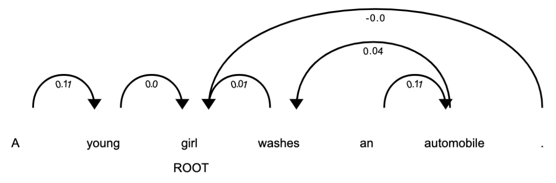
[11]https://huggingface.co/transformers/model_doc/bert.html



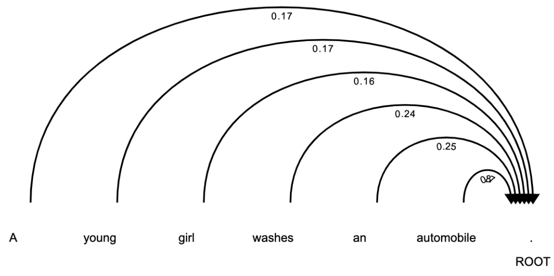
(a) Parse obtained using the the PTB-All (×) configuration.



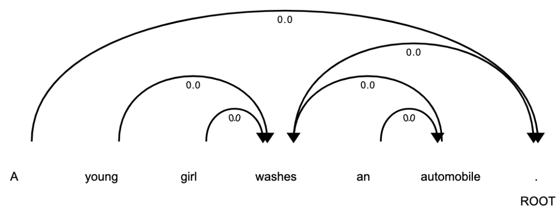
(b) Parse obtained using the the PTB-All (✓) configuration.



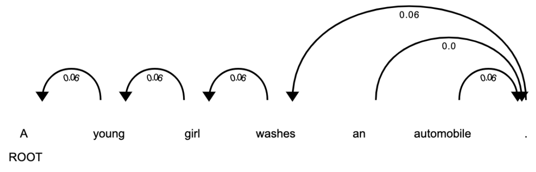
(c) Parse obtained using the the PTB-100 (×) configuration.



(d) Parse obtained using the the PTB-100 (✓) configuration.



(e) Parse obtained using the attention head #1 and without updating BERT.



(f) Parse obtained using the attention head #1 and updating BERT.

Figure 2: Example of parse obtained using various configurations from our model. The parser component is initialized on PTB-All (2a, 2b), PTB-100 (2c, 2d) or BERT (2e, 2f). We either freeze (×) or update (✓) the parser during the fine tuning on the SNLI. We include the weights α produced from the parser. We report the accuracy from a single run on the test set.

samples while the one from the last row (2e and 2f) are initialized using unsupervised patterns.

For the initialization with the PTB, we observe the fine-tuning makes the tree evolve to trivial structures and tend to connect every node to an arbitrary root. We hypothesize, such trivial structures present advantages from a computational point of view. As observed in Shi et al. (2018), trivial trees without syntax properties might lead to surprisingly good results. Shi et al. (2018) hypothesize that trivial trees gain might benefit from shallow and balanced properties.

For BERT parser initialization, we observe the fine-tuning produces rather sequential patterns, with words connected to direct neighbors. Some isolated groups of words also present inner connections.