On the Branching Bias of Syntax Extracted from Pre-trained Language Models

Huayang Li Lemao Liu Guoping Huang Shuming Shi
Tencent AI Lab
{alanili,redmondliu,donkeyhuang,shumingshi}@tencent.com

Abstract

Many efforts have been devoted to extracting constituency trees from pre-trained language models, often proceeding in two stages: feature definition and parsing. However, this kind of methods may suffer from the branching bias issue, which will inflate the performances on languages with the same branch it biases to. In this work, we propose quantitatively measuring the branching bias by comparing the performance gap on a language and its reversed language, which is agnostic to both language models and extracting methods. Furthermore, we analyze the impacts of three factors on the branching bias, namely parsing algorithms, feature definitions, and language models. Experiments show that several existing works exhibit branching biases, and some implementations of these three factors can introduce the branching bias.

1 Introduction

Neural language models such as LSTM (Merity et al., 2018; Peters et al., 2018), GPT2 (Radford et al., 2019), and BERT (Devlin et al., 2019; Liu et al., 2019) have achieved state-of-the-art performance in various downstream NLP tasks. Many recent works try to interpret their success by revealing the linguistic properties captured by these language models (Hewitt and Manning, 2019; Clark et al., 2019; Jawahar et al., 2019; Tenney et al., 2019). One interesting line of these works tries to extract discrete constituency trees from pre-trained language models (Mareček and Rosa, 2018, 2019; Kim et al., 2020; Wu et al., 2020). The core of these works is to extract syntax in two stages. Firstly, it defines the feature scores based on a language model, namely, the *feature definition stage*. Secondly, it leverages the feature scores to build a constituency tree, namely, the *parsing stage*.

However, the degree to which the extracted constituency trees match gold constituency annotations may imprecisely reflect the model's competence of capturing syntax, since their final performance may benefit from the branching bias. For example, as pointed out by Dyer et al. (2019), the syntax extracted from the ordered neuron based language model (Shen et al., 2019) is biased to right-branching languages [1] (e.g., English). Nevertheless, the approach to measuring the bias in Dyer et al. (2019) is highly dependent on the architecture of ordered neuron and its parsing algorithm. Therefore, it is far from trivial to be applied to general pre-trained language models.

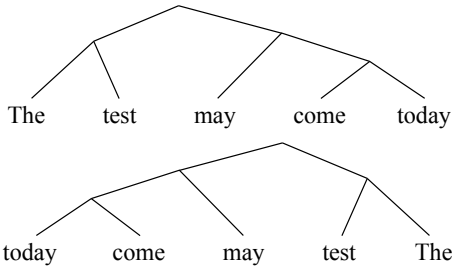

Figure 1: Constituency trees of a right-branching language and its reversed (left-branching) language. The tree at the bottom is obtained by reversing the tree at the top.

This paper proposes a new approach to reveal the branching bias of syntax from pre-trained language models, which is agnostic to model architectures and parsing algorithms. The key idea of our approach is based on the following observation: We can construct a left-branching language by reversing a right-branching language and vice versa. An illustration is given in Figure 1. If a syntax extracting method has no branching bias, the parsing performances on the original language and the re-

[1] Right-branching language is considered to be head-initial, which means the head (e.g., the verb is the head in a verb phrase) always precedes its complements. In contrast, left-branching language is head-final, where the head follows its complements (Kiparsky, 1996; Kroch, 2001).

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 4473–4478

versed language should have little or no difference. Therefore, the performance gap can be used as an indicator of branching bias. Using our approach, we find that some recent works on pre-trained language models suffer from the branching bias (Kim et al., 2020; Wu et al., 2020; Mareček and Rosa, 2018). We further investigate on an in-depth question: Does the bias come from language models? Or the extraction methods (feature definition and parsing algorithm)? We propose a simple approach to quantitatively analyze the bias in them, which tries to control the impacts of other factors while studying a specific part in the pipeline.

2 Methodology

2.1 Measuring branching bias

Intuitively, branching bias means that the induced syntax tends a specific branching structure, such as the right-branching in Shen et al. (2019), as pointed out by Dyer et al. (2019). For example, a right-branching bias will exaggerate the method's performance on a right-branching language and undermine its performance on a left-branching language. Therefore, a natural way to quantify the branching bias in syntax is to compare the performance gap between two natural languages with different branches (e.g., English and Japanese).

Unfortunately, due to the intrinsic differences between the two natural languages, it may be unfair to compare their performances directly. Therefore, for a language L, we build a synthetic language L' by reversing the word order in the way of right-to-left, rather than the left-to-right order in language L. If language L is right-branching, then language L' will be left-branching, as shown in Figure 1. Based on this observation, we use a natural language L and a synthetic language L' to measure the performance gap.

More concretely, the performance gap between language L and L', namely the *branching gap*, is defined as follows:

$$B = m(\boldsymbol{t}, \boldsymbol{g}) - m(\boldsymbol{t}', \boldsymbol{g}'), \quad (1)$$

where m is a metric function to measure the quality of the parsing tree (e.g., f1-score); $\boldsymbol{t}$ is a tree extracted by a syntax extracting method on language L, and $\boldsymbol{g}$ is its golden truth; $\boldsymbol{t}'$ and $\boldsymbol{g}'$ are defined similarly but on the reversed language L'. To make the comparison in Eq.(1) fairer, we guarantee that training and testing datasets for both languages are the same except for the word order.

If a syntax extracting method is unbiased, the branching gap would be nearly 0.[2] The sign of indicates the direction of the branching bias. It is worth noting that the proposed approach to measure branching bias is independent of the model architecture and the syntax extracting method, unlike the approach used in Dyer et al. (2019). Therefore, our approach can be naturally applied to any pre-trained language models and syntax extracting methods. Besides, Dyer et al. (2019) mainly focus on the branching bias in a specific parsing algorithm (Shen et al., 2019). In our work, we further analyze the branching bias in feature definitions and language models, besides parsing algorithms.

2.2 Factors affecting branching bias

Since constituency trees are extracted from a pre-trained language model using a syntax extracting method, the branching bias may owe to both the syntax extracting method and the language model. More precisely, the branching bias may be affected by parsing algorithms, definitions of feature scores, and language models. In the rest of this section, we will investigate the branching bias in each of the three factors one-by-one.

Bias in parsing algorithm Since the parsing algorithm is on the top of the language model and feature definition, To analyze the bias in a parsing algorithm alone, we need to exclude the influences of these two factors. To this end, we propose to assign a sequence of random scores as the feature scores and then run the parsing algorithm using these random scores to obtain the constituency tree. The random feature scores are generated according to a uniform distribution[3]. Since the feature scores are independent of both the language model and the feature definition, the branching bias can be introduced solely by the parsing algorithm if a noticeable branching gap is observed.

Bias in feature definition Feature definition is the type of information (e.g., hidden vectors or attention matrix) from a language model, converted into feature scores, and then fed into a parsing algorithm. Some feature definitions may also intrinsically contain branching bias. To reveal the bias solely dependent on a specific feature definition, instead of using the original weights (e.g., hidden

[2] Though we defined the branching gap on the sentence level, it can be easily applied to the corpus level.

[3] For feature scores that need to be normalized, and we will assign the random value before the normalizing operation.

#	Syntax Extracting Method	Model	Parsing Alg.	L	L'	Gap
1	(Mareček and Rosa, 2018)	BERT	ATTNSPAN	27.81	29.60	−1.79
2		GPT2	ATTNSPAN	27.90	23.49	+4.41
3	(Kim et al., 2020)	BERT	DIST	24.93	25.23	−0.30
4		GPT2	DIST	31.09	36.29	−5.20
5	(Wu et al., 2020)	BERT	MART	35.82	19.52	+16.30
6		GPT2	MART	32.39	21.99	+10.40

Table 1: The branching gaps of some syntax extracting methods. The results are corpus-level F1 scores on English. L is the original language and L' is its reversed version.

representations and attention weights) outputted by a pre-trained language model, we propose randomly initializing them and using them to compute feature scores. Then we run an unbiased parsing algorithm on the feature scores generated in this way. Therefore, if there is a noticeable branching gap, the branching bias will be attributed to the feature definition. The pipeline to extract syntax is independent of the language model, and the fixed parsing algorithm is unbiased.

Bias in language model The pre-trained language model is the input of a syntax extracting method. We further analyze the branching bias in a language model. To analyze the branching bias in it, we firstly choose an unbiased syntax extracting pipeline (i.e., both the feature definition and parsing algorithm are fair) and then calculate the branching gap using the well-trained language models on languages L and L'. Since there is no branching bias within our selected extracting method, the branching bias can be attributed to the input itself, if a branching gap is observed.

3 Experiments

3.1 Settings

Data We choose English as the main language in our experiments. The English data used for training language models is the concatenation of 1M lines of Wikipedia data (Devlin et al., 2019) and the Penn TreeBank (PTB) (Marcus et al., 1993) training data. We use PTB-22 and PTB-23 for validation and test, respectively. Besides, to rule out the impact of other linguistic properties, we also conduct part of our experiments on German and Chinese. We use the German Treebank from the SPMRL (Seddah et al., 2014) and Penn Chinese TreeBank (CTB) (Xue et al., 2005) with their provided test sets to evaluate previous methods on those two languages, respectively.

Language Models In our experiments, we train three different language models (i.e., BERT, GPT2, LSTM) for English and its reversed language [4]. The BERT and GPT2 models are trained using Huggingface's Transformers (Wolf et al., 2019) and we use the default parameters of their base settings (Devlin et al., 2019; Radford et al., 2019; Wolf et al., 2019). The LSTM model is trained using awd-lstm-lm[5], and we use the parameters similar to Merity et al. (2018). Models used for extracting syntax are selected according to the PPL on validation set. The tokenizers for BERT and GPT2 are trained using the toolkit huggingface/tokenizers[6], and their vocabulary sizes are 22000 and 35000 respectively. The tokenizer of GPT2 is shared with LSTM.

Syntax Extracting Methods To evaluate the branching bias, we use the codes[7] of Kim et al. (2020) and Wu et al. (2020), and re-implement the algorithm in Mareček and Rosa (2018). The parsing algorithms proposed by them are referred to as DIST, MART, and ATTNSPAN respectively. Note that Kim et al. (2020) propose a trick to explicitly inject right-branching bias to their method, and we set the weight of this injected external bias to zero in our experiments. For feature definitions, we mainly focus on three types of feature definitions, which are hidden representation (Kim et al., 2020), full attention (Mareček and Rosa, 2018), and prefix attention (Kim et al., 2020; Wu et al., 2020). [8] The

[4] We train a language model on the reversed language by reversing the entire training corpus

[5] https://github.com/salesforce/awd-lstm-lm

[6] https://github.com/huggingface/tokenizers

[7] https://github.com/galsang/trees_from_transformers and https://github.com/LividWo/Perturbed-Masking

[8] Prefix-attention means the attention is performed over the prefix words as in GPT2 whereas full-attention is over all words in a sentence as in BERT.

#	Parsing Alg.	EN			ZH			DE		
		L	L'	Gap	L	L'	Gap	L	L'	Gap
1	ATTNSPAN	21.37	21.45	−0.08	17.15	16.98	+0.17	17.79	17.78	+0.01
2	DIST	18.30	18.27	+0.03	15.28	15.76	−0.48	17.01	16.94	−0.07
3	MART	26.11	15.41	+10.70	19.90	9.51	+10.39	8.95	17.07	−8.12
4	RANDOM	18.31	18.37	−0.06	15.33	15.03	+0.30	16.99	16.98	+0.01
5	RIGHT-B	35.82	10.40	+25.42	19.77	8.11	+11.66	7.99	16.54	−8.55

Table 2: The branching gaps of parsing algorithms with random feature scores. Feature scores are generated according to a uniform distribution $[-1, 1]$. The results are averaged corpus-level F1 scores with 10 random seeds. L is the original language and L' is its reversed version.

#	Feature Def.	EN			ZH			DE		
		L	L'	Gap	L	L'	Gap	L	L'	Gap
1	HIDDEN	18.39	18.29	−0.10	15.32	15.30	+0.02	16.88	17.10	+0.28
2	PREFIX-ATTN	20.44	13.17	+7.27	16.78	12.66	+4.12	14.93	18.83	−3.90
3	FULL-ATTN	18.33	18.38	−0.05	15.12	15.04	+0.08	16.84	16.79	+0.05

Table 3: The branching gaps while applying DIST to different randomized feature definitions. The uniform distribution $[-1, 1]$ is used to randomize the weights of feature definitions. The results are averaged corpus-level F1 scores with 10 random seeds.

hyper-parameters (e.g., choices of attention head and hidden layer) of syntax extracting methods are tuned on the validation set.

3.2 Main Results

As shown in Table 1, the behaviors of different approaches are widely divergent. We find that the branching bias in BERT+ATTNSPAN and BERT+DIST are relatively lower than other approaches. However, the results of GPT2+ATTNSPAN and BERT/GPT2+MART demonstrate significant right-branching biases. GPT2+DIST shows a tendency towards left-branching. Since these approaches are pipelined, which part of their methods has an impact on the branching bias is still unclear.

The results reported in Table 1 is a little worse than those reported in Kim et al. (2020); Wu et al. (2020). One reason is that we evaluate the results on the corpus-level F1 score following the standard, rather than sentence-level (Kim et al., 2020). The other reason is that our training data is small, since it is too expensive to train reversed language models on a huge dataset. However, these results are obtained by running the released codes of Kim et al. (2020); Wu et al. (2020), and thus, we think it will not affect our findings.

3.3 Factors affecting branching bias

Branching Bias in Parsing Algorithm The branching gaps of different parsing algorithms are shown in Table 2. Observing from the experiment results in English, The branching gaps of MART is significantly larger than 0, which means it has a tendency to right-branching. In contrast, the branching gaps of parsing algorithm ATTNSPAN and DIST are nearly 0, which means they do not bias to left-branching or right-branching. Although DIST is inspired by the parsing algorithm in Shen et al. (2019), it is an unbiased, which is consistent with the claim in Kim et al. (2020). We also evaluate the parsing algorithm of Shen et al. (2019), and its branching gap is +3.22 on English, which is consistent with the finding in Dyer et al. (2019).

To examine whether some other language properties might play a role in this process, we also conduct experiments on different languages, which can help rule out the impact of specific language properties. The results in Table 2 show that MART has the same trend as RIGHT-B baseline (row 5) on both Chinese and German datasets, which is consistent with the finding on the English dataset. It is also worth noting that the branching gap for MART is positive on Chinese and English datasets, whereas it is negative on German. The reason is that both Chinese and English are right-branching languages, while German is inclined to be left-branching. However, both head-initial and head-final structures occur in the German language from the viewpoint of linguistics. In addition, one interesting observation is that, the performances of ATTNSPAN are always higher than the RANDOM

#	Language Model	L	L'	Gap
1	BERT	24.93	25.23	-0.30
2	GPT2	23.85	26.09	-2.24
3	LSTM	28.72	26.22	$+2.50$
4	RANDOM	18.31	18.37	-0.06

Table 4: The branching gaps of different language models using an unbiased pipeline HIDDEN + DIST. The results are corpus-level F1 scores on English.

baseline. We hypothesize that ATTNSPAN may have a bias towards the balance tree, due to its way to compute weights of splitting points.

Branching Bias in Feature Definition As shown in Table 3, we choose the unbiased parsing algorithm DIST to further analyze the branching bias in feature definitions. It is worth noting that, after normalization, the attention matrix of PREFIX-ATTN is lower triangular, and that of FULL-ATTN is fully filled. We find that the feature definitions based on HIDDEN and FULL-ATTN are unbiased. However, PREFIX-ATTN tends to generate right-branching trees, where the branching gap is $+7.27$ on English. This finding is consistent with that on Chinese and German. One possible explanation about PREFIX-ATTN is that the attention scores will become distracted with the prefix grows, such that the feature scores in the front of the sequence, which has a larger value, would be picked at first.

Branching Bias in Language Models After the analyses in previous steps, we will use the unbiased parsing algorithm and feature definition, DIST and HIDDEN, to evaluate the branching bias in language models. Note that the results in this section is different from those in Table 1, since other feature definitions are prohibited except for HIDDEN.

Our experiments conducted on language models are shown in Table 4. The performances of BERT on both branching are nearly the same, where the branching gap is just -0.30. In contrast, slight branching gap is observed on both GPT2 and LSTM. The branching gap of GPT2 is -2.24. With the same left-to-right paradigm, LSTM behaviors a positive branching gap $+2.50$. The opposite branching gap may be caused by the difference between model architectures, where GPT2 is based on self-attention (Vaswani et al., 2017) and LSTM is based on gating mechanism (Hochreiter and Schmidhuber, 1997). However, the random noises may also play a role in this observation, since the performance range of GPT2 models trained on the original English dataset with different random seed can also reach around 1.50. We will investigate it in future works.

4 Conclusion

In this paper, we propose an approach to quantitatively analyze the branching bias in extracting syntax from pre-trained language models. Unlike previous work, our approach is more general to be applied to any pre-trained language models and syntax extracting methods. Furthermore, we systematically analyze three factors in depth that may affect the branching bias: the language model, feature definition, and parsing algorithm. Our experiments show that branching biases are in many recent works. In addition, these biases can be brought by each of the three factors. We appeal that researchers should carefully design their syntax extracting method to reveal the real competence of syntax from a pre-trained language model.

References

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*, pages 276–286, Florence, Italy. 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, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Chris Dyer, Gábor Melis, and Phil Blunsom. 2019. A critical analysis of biased parsers in unsupervised parsing. *arXiv preprint arXiv:1909.09428*.

John Hewitt and Christopher D Manning. 2019. A structural probe for finding syntax in word representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4129–4138.

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

Ganesh Jawahar, Benoît Sagot, Djamé Seddah, Samuel Unicomb, Gerardo Iñiguez, Márton Karsai, Yannick

Léo, Márton Karsai, Carlos Sarraute, Éric Fleury, et al. 2019. What does bert learn about the structure of language? In *57th Annual Meeting of the Association for Computational Linguistics (ACL), Florence, Italy*.

Taeuk Kim, Jihun Choi, Daniel Edmiston, and Sanggoo Lee. 2020. Are pre-trained language models aware of phrases? simple but strong baselines for grammar induction. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Paul Kiparsky. 1996. The shift to head-initial vp in germanic. *Studies in comparative Germanic syntax*, 2:140–179.

Anthony S Kroch. 2001. *Syntactic change*. na.

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

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

David Mareček and Rudolf Rosa. 2018. Extracting syntactic trees from transformer encoder self-attentions. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 347–349.

David Mareček and Rudolf Rosa. 2019. From balustrades to pierre vinken: Looking for syntax in transformer self-attentions. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 263–275, Florence, Italy. Association for Computational Linguistics.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. Regularizing and optimizing LSTM language models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of NAACL-HLT*, pages 2227–2237.

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

Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. 2014. Introducing the SPMRL 2014 shared task on parsing morphologically-rich languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 103–109, Dublin, Ireland. Dublin City University.

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.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R. Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R. Bowman, Dipanjan Das, and Ellie Pavlick. 2019. What do you learn from context? probing for sentence structure in contextualized word representations. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

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

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Zhiyong Wu, Yun Chen, Ben Kao, and Qun Liu. 2020. Perturbed masking: Parameter-free probing for analyzing and interpreting BERT. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4166–4176, Online. Association for Computational Linguistics.

Naiwen Xue, Fei Xia, Fu-Dong Chiou, and Marta Palmer. 2005. The penn chinese treebank: Phrase structure annotation of a large corpus. *Natural language engineering*, 11(2):207.