Representations of Syntax [MASK] Useful: Effects of Constituency and Dependency Structure in Recursive LSTMs

Michael A. Lepori[1] **Tal Linzen**[1,2] **R. Thomas McCoy**[2]
[1]Department of Computer Science [2]Department of Cognitive Science
Johns Hopkins University
{mlepori1, tal.linzen, tom.mccoy}@jhu.edu

Abstract

Sequence-based neural networks show significant sensitivity to syntactic structure, but they still perform less well on syntactic tasks than tree-based networks. Such tree-based networks can be provided with a constituency parse, a dependency parse, or both. We evaluate which of these two representational schemes more effectively introduces biases for syntactic structure that increase performance on the subject-verb agreement prediction task. We find that a constituency-based network generalizes more robustly than a dependency-based one, and that combining the two types of structure does not yield further improvement. Finally, we show that the syntactic robustness of sequential models can be substantially improved by fine-tuning on a small amount of constructed data, suggesting that data augmentation is a viable alternative to explicit constituency structure for imparting the syntactic biases that sequential models are lacking.

1 Introduction

Natural language syntax is structured hierarchically, rather than sequentially (Chomsky, 1957; Everaert et al., 2015). One phenomenon that illustrates this fact is **English subject-verb agreement**, the requirement that verbs and their subjects must match in number. The hierarchical structure of a sentence determines which noun phrase each verb must agree with; sequential heuristics such as agreeing with the most recent noun may succeed on simple sentences such as (1a) but fail in more complex cases such as (1b):

(1) a. The **boys kick** the ball.
b. The **boys** by the red truck **kick** the ball.

We investigate whether a neural network must process input according to the structure of a syntactic parse in order for it to learn the appropriate rules governing these dependencies, or whether there is sufficient signal in natural language corpora for low-bias networks (such as sequential LSTMs) to learn these structures. We compare sequential LSTMs, which process sentences from left to right, with tree-based LSTMs that process sentences in accordance with an externally-provided, ground-truth syntactic structure.

	No Constituency	Constituency
No Heads	BiLSTM	Constituency LSTM
Heads	Dependency LSTM	Head-Lexicalized LSTM

Table 1: Linguistic properties of our four models.

We consider two types of syntactic structure: **constituency structure** (Chomsky, 1993; Pollard and Sag, 1994) and **dependency structure** (Tesniere, 1959; Hudson, 1984). We investigate models provided with either structure, both structures, or neither structure (see Table 1), and assess how robustly these models learn subject-verb agreement when trained on natural language.[1]

Even with the syntactic biases present in tree-based LSTMs, it is possible that natural language might not impart a strong enough signal to teach a network how to robustly track subject-verb dependencies. How might the performance of these tree-based LSTMs change if they were fine-tuned on a small dataset designed to impart a stronger syntactic signal? Furthermore, would we still need these tree structures, or could a sequential LSTM now learn to track syntactic dependencies?

We find that building in either type of syntactic structure improves performance over the BiLSTM

[1]Code, data, and models are at https://github.com/mlepori1/Representations_Of_Syntax

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 3306–3316

baseline, thus showing that these structures are learned imperfectly (at best) by low-bias models from natural language data. Of the two types of structure, constituency structure turns out to be more useful. The dependency-only model performs well on natural language test sets, but fails to generalize to an artificially-constructed challenge set. After fine-tuning on a small dataset that is designed to impart a strong syntactic signal, the BiLSTM generalizes more robustly, but still falls short of the tree-based LSTMs.

We conclude that for a network to robustly show sensitivity to syntactic structure, stronger biases for syntactic structure need to be introduced than are present in a low-bias learner such as a BiLSTM, and that, at least for the subject-verb agreement task, constituency structure is more important than dependency structure. Both tree-based model structure and data augmentation appear to be viable approaches for imparting these biases.

2 Related Work

Prior work has shown that neural networks without explicit mechanisms for representing syntactic structure can show considerable sensitivity to syntactic dependencies (Goldberg, 2019; Gulordava et al., 2018; Linzen et al., 2016), and that certain aspects of the structure of the sentence can be reconstructed from their internal representations (Lin et al., 2019; Giulianelli et al., 2018; Hewitt and Manning, 2019). Marvin and Linzen (2018) showed that sequential models still have substantial room for improvement in capturing syntax, and other work has shown that models with a greater degree of syntactic structure outperform sequential models on syntax-sensitive tasks (Yogatama et al., 2018; Kuncoro et al., 2018, 2017), including some of the tree-based models used here (Bowman et al., 2015; Li et al., 2015). One contribution of the present work is to tease apart the two major types of syntactic structure to see which one imparts more effective syntactic biases.

3 Models

3.1 BiLSTM

As our baseline model, we used a simple extension to the LSTM architecture (Hochreiter and Schmidhuber, 1997), the **bidirectional LSTM** (BiLSTM; Schuster and Paliwal, 1997). This model runs one LSTM from left to right over a sequence, and another from right to left, without appealing to tree structure. Bidirectional LSTMs outperform unidirectional LSTMs on a variety of tasks (Huang et al., 2015; Chiu and Nichols, 2016), including syntax-sensitive tasks (Kiperwasser and Goldberg, 2016). Ravfogel et al. (2019) also employs BiLSTMs for a similar agreement task.

3.2 Tree LSTMs

To study the effects of explicitly building tree structure into the model architecture, we used the **Constituency LSTM** and the **Dependency LSTM** (Tai et al., 2015), which are types of recursive neural networks (Goller and Kuchler, 1996). The Constituency LSTM operates in accordance with a binary constituency parse, composing together vectors representing a left child and a right child into a vector representing their parent. Models similar to the Constituency LSTM have been proposed by Le and Zuidema (2015) and Zhu et al. (2015).

In a Dependency LSTM, the representations of a head's children are summed, and then composed with the representation of the head itself to yield a representation of the phrase that has that head. See Appendix A for more details on both models.

3.3 Head-Lexicalized Tree LSTMs

To create a model where composition is simultaneously guided by both a dependency parse and a constituency parse, we modified the constituency model described in Section 3.2, turning it into a **head-lexicalized tree LSTM**. In a standard Constituency LSTM, the input for all non-leaf nodes is a vector of all 0's. To add head lexicalization, we instead feed in the word embedding of the correct headword of that constituent as the input, where the choice of headword is determined using the Stanford Dependency Parser (Manning et al., 2014). See Appendix B for more details, as well as an example of a head-lexicalized constituency tree. This model is similar to the head-lexicalized tree LSTM of Teng and Zhang (2017). However, their model learns how to select the heads of constituents in an unsupervised manner; these heads may not correspond to the syntactic notion of heads. Because we seek to understand the effect of using the heads derived from the dependency parse, we provide our models with explicit head information.

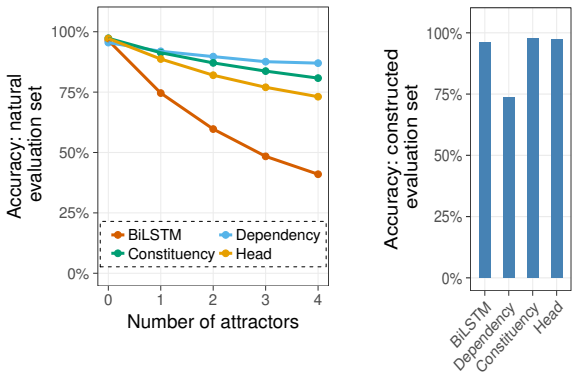
4 Task

We adapted a syntax-sensitive task that previous work has used to assess the syntactic capabilities

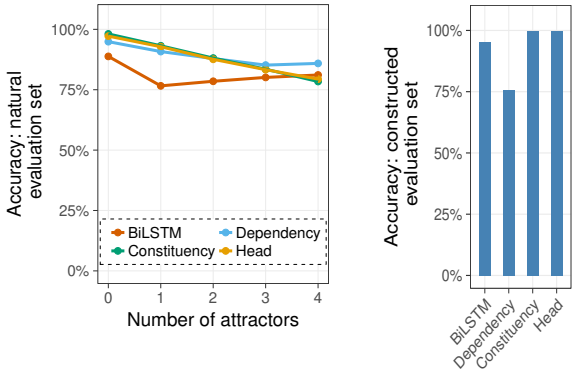
of LSTMs—the number prediction task (Linzen et al., 2016). The most standard version of this task is based on a left-to-right language modeling objective; however, tree-based models are not compatible with left-to-right language modeling. Therefore, we made two modifications to this objective, both of which have precedents in the literature: First, we gave the model an entire present-tense sentence with main verb masked out, following Goldberg (2019). Second, the model's target output was the number of the masked verb: SINGULAR or PLURAL; we follow Linzen et al. (2016) and Ravfogel et al. (2019) in framing number prediction as a classification task. To solve the task, the model must identify the subject whose head is the main verb (in the dependency formalism), and use that information to determine the syntactic number of the verb; e.g., for (2), the answer is SINGULAR.

(2) The girl ***MASK*** the ball.

Linzen et al. (2016) pointed out that there are several incorrect heuristics which models might adopt for this task because these heuristics still produce decent classification accuracy. One salient example is picking the syntactic number of the most recent noun to the left of the verb. We hypothesize that tree-based models will be less susceptible to these non-robust heuristics than sequential models.



(a) Results for models trained on natural language.



(b) Results for models trained on natural language and then exposed to a 500-sentence augmentation set.

Figure 1: Results on binary classification of masked verbs as SINGULAR or PLURAL. All results are averages across 3 runs. Chance performance is 50%.

5 Experiment 1: Natural Language

Data: We train our models on a subset of the dataset from Linzen et al. (2016) that is chosen to have a uniform label distribution (50% SINGULAR and 50% PLURAL). We made this choice because our task format differs from that used in some past work (see Section 4), so performance on the task as we have framed it cannot be directly compared to prior work. In the absence of baselines from the literature, we use chance performance of 50% as a baseline; to ensure that this baseline is reasonable, we balance the label distribution during training to discourage models from becoming biased toward one label.

We use two types of test sets: those that contain adversarial attractors, and those that do not. An **adversarial attractor** is a noun that is between the subject and the main verb of a sentence and that has the opposite syntactic number from the subject noun. Adversarial attractors have been found to produce agreement errors in humans (Bock and Miller, 1991) and neural models (Goldberg, 2019; Gulordava et al., 2018; Linzen et al., 2016). We use code from Goldberg (2019)[2] to extract adversarial datasets containing varying numbers of attractors, from 0 to 4 attractors. Sentence (3) provides an example of a sentence with 4 attractors.

(3) Algorithmic **problems** such as *[type] [checking]* and *[type] [inference]* **are** more difficult for equirecursive types as well.

See Appendix D for details on our corpus and on preprocessing, and Appendix C.1 for training.

Natural language evaluation: All of the tree-based models outperformed the BiLSTM in the presence of attractors (Figure 1a). Compared to prior work with the number prediction task, our BiLSTM performed very poorly on the 4 Attractors dataset. However, our results cannot be directly compared to previous work because of the modifications we have made to the task, data, and training procedure in order to accommodate tree-

[2] https://github.com/yoavg/bert-syntax

based models. In light of these modifications, there are several reasons why the BiLSTM's low accuracy is unsurprising. First, we used a balanced label distribution during training. In the standard dataset from Linzen et al. (2016), the class labels are not balanced, so models evaluated on that dataset might outperform our BiLSTM by exploiting the biased label distribution—a heuristic that our balanced training set discourages. Another potential cause for the BiLSTM's poor performance is that, in order to balance the label frequencies, we used a smaller training set than was used in past work (81,000 sentences instead of 121,000 sentences). Finally, it is possible that allowing models to see the entire sentence may allow them to acquire non-robust heuristics related to the words following the main verb. For example, a model might learn spurious correlation between the syntactic number of subjects and their direct objects. See Appendix E, Table 2 for results on all test sets.

Constructed sentence evaluation: With naturally occurring sentences, it is possible that models perform well not because they have mastered syntax, but rather because of statistical regularities in the data. For example, given *The players *MASK* the ball*, the model may be able to exploit the fact that animate nouns tend to be subjects while inanimate nouns do not. As pointed out by Gulordava et al. (2018), this would allow the model to correctly predict syntactic number, but for the wrong reasons. To test whether our models were leveraging this statistical heuristic, we constructed a 400-sentence test set where this heuristic cannot succeed. We did so using a probabilistic context-free grammar (PCFG) under which all words of a given part of speech are equally likely in all positions; each sentence from this grammar is of the form Subject-Verb-Object, and all noun phrases can optionally be modified by adjectives and/or prepositional phrases (see Appendix F), as in (4):

(4) The **fern** near the sad teachers **hates** the singer.

The Dependency LSTM is especially likely to fall prey to word cooccurrence heuristics, as it lacks the ability for a parent to account for the sequential position of its children. This can be an issue when determining whether a verb is supposed to be singular or plural, because the model has no robust way to distinguish a verb's subject from its direct object. The dependency model did indeed perform at chance (See the bar graph in Figure 1a).[3] This suggests that the dependency model's high accuracy is partially due to lexical heuristics rather than syntactic processing. In contrast, the other models performed well, suggesting that they are less susceptible to relying on word cooccurrence.

6 Experiment 2: Fine-tuning

In Experiment 1, tree-based models dramatically outperformed the BiLSTM in the presence of attractors. This difference may have arisen because most natural language sentences are simple, and thus they do not generate enough signal to illustrate the importance of tree structure to a low-bias learner, such as a BiLSTM. Recent work has shown the effectiveness of syntactically-motivated fine-tuning at increasing the robustness of neural models (Min et al., 2020). Would our models generalize more robustly if we added a few training examples that do not lend themselves to non-syntactic heuristics?

To provide the model with a stronger signal about the importance of syntactic structure, we fine-tuned our models on a dataset designed to impart this signal. We used a variant of the PCFG (see Appendix F) from Section 5 to generate a 500-sentence **augmentation set**. This augmentation set cannot be solved using word cooccurrence statistics, and contains some sentences with attractors. The models were then fine-tuned on the augmentation set for just *one epoch* over the 500 examples. See Appendix C.2 for training details.

Results: The head-lexicalized model and the BiLSTM benefited most from fine-tuning, with the head-lexicalized model now matching the performance of the Constituency LSTM, and the BiLSTM showing dramatic improvement on sentences with multiple attractors (Figure 1b; see Appendix E, Table 3 for detailed results). While the BiLSTM's accuracy increased on sentences with attractors, it decreased on the No Attractors test set. We suspect that this is because augmentation discouraged the model from using heuristics: while this makes performance more robust overall, it may hurt accuracy on simple examples where the heuristics give the correct answer (Min et al., 2020). As expected from its architectural limitations, the Dependency LSTM did not noticeably benefit from fine-tuning

[3] Most sentences in the test set have only two nouns. 50% of the time, they will agree in number, and the syntactic number is unambiguous. Random guessing on the other 50% of cases would yield about 75% accuracy.

because it cannot extract the relevant information from the augmentation set. There was no clear effect of augmentation on the Constituency LSTM.[4]

7 Discussion

Overall, we found that neural models trained on natural language achieve much more robust performance on syntactic tasks when syntax is explicitly built into the model. This suggests that the information we provided to our tree-based models is unlikely to be learned from natural language by models with only general inductive biases.

In Experiment 1, the network provided with a dependency parse did the best on most of the natural language test sets. This is unsurprising, as the task is largely about a particular dependency (i.e., the dependency between a verb and its subject). At the same time, as demonstrated by the constructed sentence test, the syntactic capabilities of the Dependency LSTM are inherently limited. Thus, it must default to non-robust heuristics in cases where the unlabeled dependency information is ambiguous. In future work, these syntactic limitations may be overcome by giving the model typed dependencies (which would distinguish between a subject-verb dependency and a verb-object dependency).

One might expect the head-lexicalized model to perform the best, since it can leverage both syntactic formalisms. However, it performs no better than the constituency model when trained on natural language, suggesting that there is little benefit to incorporating dependency structure into a Constituency LSTM. In some cases, the head-lexicalized model without fine-tuning even performs worse than the Constituency LSTM. When fine-tuned on more challenging constructed examples, the head-lexicalized model performed similarly to the Constituency LSTM, suggesting that there is not enough signal in the natural language training set to teach this model what to do with the heads it has been given.

Our results point to two possible approaches for improving how models handle syntax. The first approach is to use models that have explicit mechanisms for representing syntactic structure. In particular, our results suggest that the most important aspect of syntactic structure to include is constituency structure, as constituency models appear to implicitly learn dependency structure as well. Though the models we used require parse trees to be provided, it is possible that models can learn to induce tree structure in an unsupervised or weakly-supervised manner (Bowman et al., 2016; Choi et al., 2018; Shen et al., 2019). Another effective approach for improving the syntactic robustness of neural models is data augmentation, as demonstrated in Experiment 2. With this approach, it is possible to bring the syntactic performance of less-structured models closer to that of models with explicit tree structure, even with an augmentation set generated simply and easily using a PCFG.

Future work should further explore both of these approaches. Our conclusions about the importance of explicit mechanisms for representing syntactic structure can be strengthened by developing different formulations of the tree LSTMs. It seems particularly promising to explore alternative formulations of the Dependency LSTM (as mentioned above) and the effect of learning embeddings of non-terminal symbols for the Constituency LSTM. Finally, future work should investigate whether data augmentation can fully bridge the gap between low-bias learners and structured tree LSTMs, and whether our conclusions apply to other syntactic phenomena besides agreement.

[4]Note that the constructed test set used here is controlled to have no overlap with the augmentation set. Thus, it is not exactly the same as the set used in Section 5, but both corpora are generated from the same CFG.

Acknowledgments

This research was supported by a Google Faculty Award to Tal Linzen, NSF Graduate Research Fellowship No. 1746891, and NSF Grant No. BCS-1920924. Our experiments were conducted using the Maryland Advanced Research Computing Center (MARCC).

References

Kathryn Bock and Carol Ann Miller. 1991. Broken agreement. *Cognitive Psychology*, 23:45–93.

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 (Volume 1: Long Papers)*, pages 1466–1477, Berlin, Germany. Association for Computational Linguistics.

Samuel R. Bowman, Christopher D. Manning, and Christopher Potts. 2015. Tree-structured composition in neural networks without tree-structured ar-

chitectures. In *Proceedings of the 2015 International Conference on Cognitive Computation: Integrating Neural and Symbolic Approaches*, volume 1583 of *COCO'15*, pages 37–42, Aachen, Germany, Germany.

Jason P. C. Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics*, 4:357–370.

Jihun Choi, Kang Min Yoo, and Sang-goo Lee. 2018. Learning to compose task-specific tree structures. In *32nd AAAI Conference on Artificial Intelligence*.

Noam Chomsky. 1957. *Syntactic structures*. Mouton and Co., The Hague.

Noam Chomsky. 1993. A minimalist program for linguistic theory. *The View from Building 20: Essays in Linguistics in Honor of Sylvain Bromberger*, pages 1–53.

Martin B. H. Everaert, Marinus A. C. Huybregts, Noam Chomsky, Robert C. Berwick, and Johan J. Bolhuis. 2015. Structures, not strings: Linguistics as part of the cognitive sciences. *Trends in Cognitive Sciences*, 19(12):729–743.

Mario Giulianelli, Jack Harding, Florian Mohnert, Dieuwke Hupkes, and Willem Zuidema. 2018. Under the hood: Using diagnostic classifiers to investigate and improve how language models track agreement information. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 240–248, Brussels, Belgium. Association for Computational Linguistics.

Yoav Goldberg. 2019. Assessing BERT's syntactic abilities. *arXiv preprint arXiv:1901.05287*.

Cristoph Goller and Andreas Kuchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *1996 IEEE International Conference on Neural Networks*, volume 1, pages 347–352 vol.1. IEEE.

Kristina Gulordava, Piotr Bojanowski, Edouard Grave, Tal Linzen, and Marco Baroni. 2018. Colorless green recurrent networks dream hierarchically. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1195–1205, New Orleans, Louisiana. Association for Computational Linguistics.

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, Minneapolis, Minnesota. Association for Computational Linguistics.

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

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arxiv:1508.01991*.

Richard A. Hudson. 1984. *Word grammar*. Blackwell Oxford.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference for Learning Representations*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. 2017. What do recurrent neural network grammars learn about syntax? In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1249–1258, Valencia, Spain. Association for Computational Linguistics.

Adhiguna Kuncoro, Chris Dyer, John Hale, Dani Yogatama, Stephen Clark, and Phil Blunsom. 2018. LSTMs can learn syntax-sensitive dependencies well, but modeling structure makes them better. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1426–1436, Melbourne, Australia. Association for Computational Linguistics.

Phong Le and Willem Zuidema. 2015. Compositional distributional semantics with long short term memory. In *Proceedings of the 4th Joint Conference on Lexical and Computational Semantics*, pages 10–19, Denver, Colorado. Association for Computational Linguistics.

Jiwei Li, Thang Luong, Dan Jurafsky, and Eduard Hovy. 2015. When are tree structures necessary for deep learning of representations? In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2304–2314, Lisbon, Portugal. Association for Computational Linguistics.

Yongjie Lin, Yi Chern Tan, and Robert Frank. 2019. Open sesame: Getting inside BERT's linguistic knowledge. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 241–253, Florence, Italy. Association for Computational Linguistics.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

Rebecca Marvin and Tal Linzen. 2018. Targeted syntactic evaluation of language models. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1192–1202.

Junghyun Min, R. Thomas McCoy, Dipanjan Das, Emily Pitler, and Tal Linzen. 2020. Syntactic data augmentation increases robustness to inference heuristics. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Seattle, Washington. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Carl Pollard and Ivan A. Sag. 1994. *Head-driven phrase structure grammar*. University of Chicago Press.

Shauli Ravfogel, Yoav Goldberg, and Tal Linzen. 2019. Studying the inductive biases of RNNs with synthetic variations of natural languages. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 3532–3542.

Mike Schuster and Kuldip K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45:2673–2681.

Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron C. Courville. 2019. Ordered neurons: Integrating tree structures into recurrent neural networks. In *International Conference on Learning Representations*.

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 (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China. Association for Computational Linguistics.

Zhiyang Teng and Yue Zhang. 2017. Head-lexicalized bidirectional tree LSTMs. *Transactions of the Association for Computational Linguistics*, 5:163–177.

Lucien Tesniere. 1959. *Eléments de syntaxe structurale, Klincksieck*. Pans.

Dani Yogatama, Yishu Miao, Gabor Melis, Wang Ling, Adhiguna Kuncoro, Chris Dyer, and Phil Blunsom. 2018. Memory architectures in recurrent neural network language models. *International Conference on Learning Representations*.

Xiaodan Zhu, Parinaz Sobhani, and Hongyu Guo. 2015. Long short-term memory over recursive structures. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *ICML'15*, pages 1604–1612.

A Appendix: Tree LSTM Details

The constituency-based model that we use is the N-ary Tree-LSTM from Tai et al. (2015), with N fixed at 2 such that the tree is strictly binary; the equations for this model are shown below. Each W is an input weight matrix, each U is a hidden state update weight matrix, each b is a bias term, each x is an input word embedding, and each h is a hidden state. These equations are adaptations of the typical LSTM equations that allow the LSTM to be structured according to a constituency parse. The x_j is the input embedding for a particular node in the constituency tree. In a Constituency LSTM, all leaf nodes receive the embedding for the word at that leaf, while all other nodes receive a vector of 0's. Every non-leaf node is thus a composition of the hidden states of its two children. In these equations, k = 1 or 2, which allows "the left hidden state in a binary tree to have either an excitatory or inhibitory effect on the forget gate of the right child" (Tai et al., 2015). Importantly, this model distinguishes between a node's left and right children.

$$i_j = \sigma(W^{(i)}x_j + \sum_{l=1}^{2} U_l^{(i)} h_{jl} + b^{(i)}) \quad (1)$$

$$f_{jk} = \sigma(W^{(f)}x_j + \sum_{l=1}^{2} U_{kl}^{(f)} h_{jl} + b^{(f)}) \quad (2)$$

$$o_j = \sigma(W^{(o)}x_j + \sum_{l=1}^{2} U_l^{(o)} h_{jl} + b^{(o)}) \quad (3)$$

$$u_j = tanh(W^{(u)}x_j + \sum_{l=1}^{2} U_l^{(u)} h_{jl} + b^{(u)}) \quad (4)$$

$$c_j = i_j \odot u_j + \sum_{l=1}^{2} f_{jl} \odot c_{jl} \quad (5)$$

$$h_j = o_j \odot tanh(c_j) \quad (6)$$

The following equations, also from Tai et al. (2015), define a child-sum Tree LSTM, which we structure according to a dependency parse. Here, the input x_j is the embedding of the headword of that node in the DAG that defines a dependency parse. Note that in this model, the hidden representations of the children of a node are summed. Thus, this model cannot distinguish the linear order of its children.

$$\tilde{h}_j = \sum_{k \in C(j)} h_k \quad (7)$$

$$i_j = \sigma(W^{(i)}x_j + U^{(i)}\tilde{h}_j + b^{(i)}) \quad (8)$$

$$f_{jk} = \sigma(W^{(f)}x_j + U^{(f)}h_k + b^{(f)}) \quad (9)$$

$$o_j = \sigma(W^{(o)}x_j + U^{(o)}\tilde{h}_j + b^{(o)}) \quad (10)$$

$$u_j = tanh(W^{(u)}x_j + U^{(u)}\tilde{h}_j + b^{(u)}) \quad (11)$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k \quad (12)$$

$$h_j = o_j \odot tanh(c_j) \quad (13)$$

B Appendix: Details of the Head-Lexicalized Tree LSTM Variant

Our head-lexicalized tree LSTM architecture is structured exactly the same as the Constituency LSTM. Thus, Equations 1 through 6 characterize the parameters and operations performed by the head-lexicalized tree LSTM. The difference between the two architectures lies in the input, x_j. In the Constituency LSTM, a node j was provided an input vector x_j only if j was a leaf node. In the head-lexicalized tree LSTM model, we use a dependency parse to generate a tag for each node in the constituency tree, which identifies which word in the corresponding constituent is the most dominant word in the dependency tree. The word embedding corresponding to the most dominant word in constituent j is then provided as input x_j. Thus, every node in the tree receives an input vector, and the root node is guaranteed to have the headword of the whole sentence provided as input.

More formally, a dependency parse forms a tree, T_D. For each word, w, in a given sentence, denote its score, $s(w)$, as the depth of w in T_D. A constituency parse forms a tree T_C. For every node j in T_C, let l_j denote the set of words corresponding to children of j that are leaves of T_C. The input vector x_j is then just the embedding of $w = \arg\min_{w \in l_j} s(w)$. Ties should not exist within a constituent, but if they do (due to parsing errors), then they are broken arbitrarily.

See Figure 2 for an example of a head-lexicalized constituency tree.

C Appendix: Training Details

C.1 Experiment 1

We use an embedding size and hidden cell size of 100 for every model. Our word embeddings

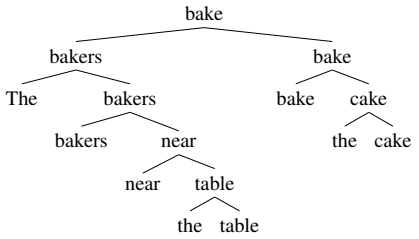


Figure 2: Head-lexicalized constituency tree for the sentence *The bakers near the table bake the cake*.

are 100-dimensional pretrained GloVe embeddings from the Wikipedia 2014 + Gigaword 5 distribution (glove.6b.zip) (Pennington et al., 2014), and we do not tune them during training. We also employ the Adam optimizer (Kingma and Ba, 2015) with the PyTorch default learning rate of 0.001. Because this is a binary classification problem, we use binary cross entropy as our loss function. These hyperparameter choices are based on Linzen et al. (2016), but we increase the hidden size from 50 to 100, in order to create slightly more capacity. Though this may seem small, the models achieved high overall accuracy, suggesting that model size was not a bottleneck.

We cap training at 50 epochs, but also employ early stopping. The early stopping procedure is as follows: Train for 10,000 sentences, then evaluate on the validation data. Stop when the average decrease in validation loss over the previous five evaluations is less than 0.0005. For all models, this occurs after about 1 or 1.5 epochs. During training, the parameters that resulted in the best validation loss are saved, and these weights are used during testing. We repeat this procedure for three random initializations of each model. The reported results are averages over these three models.

In order to turn a tree LSTM into a binary classifier, we feed the hidden state of the root into a linear layer that condenses the output into a single value, and squash the result to the range [0, 1] using a sigmoid activation function. If the result of that process is greater than 0.5, then we predict label 1, else we predict label 0. For the bidirectional LSTM, we take the representation of the masked verb from both the left to right and right to left passes and feed both of these into a linear classifier. Then we repeat the process described above, using a sigmoid activation function to constrain the prediction to the range [0, 1], and classifying based on this value.

C.2 Experiment 2

We take the same models from Experiment 1 and fine tune them on the augmentation set. We train for one epoch with the same parameters used in Experiment 1, and then use the resulting weights to evaluate the models.

D Appendix: Data

The original dataset contains approximately 1.3 million sentences. We use the Stanford constituency parser and Stanford dependency parser (Manning et al., 2014) to generate the two types of parse trees for each of these sentences, and then convert these objects into suitable representations for our models. In this process, a small percentage of examples were discarded due to the parser failing to parse them. We deviate from past work by ensuring that both classes (SINGULAR and PLURAL) are of equal size. This results in more data from the majority class (singular verb class) being thrown away. After these exclusions, we have approximately 903,000 sentences remaining. We provide our models 9% of this (81,300 sentences) to train on, 0.1% (904 sentences) to validate, and then generate our test sets from the remainder of the data. All sentences were stripped of quotation marks, apostrophes, parentheses and hyphens in order to minimize parsing failures.

The sizes of our test sets are as follows: No Attractors (50,000 sentences), Any Attractors (52,815 sentences), One Attractor (41,902 sentences), Two Attractors (8,473 sentences), Three Attractors (1,884 sentences), and Four Attractors (556 sentences). Note also that the Any Attractors dataset is the union of the One, Two, Three, and Four Attractors datasets.

E Appendix: Full Results

Table 2 contains the full results after training all models on natural language. Table 3 contains the full results after augmentation.

F Appendix: Probabilistic Context Free Grammars

Figure 3 contains the probabilistic context free grammar used to generate the constructed corpora.

$S \rightarrow DetP_s\ \ VP_s \mid DetP_p\ \ VP_p$
$DetP_s \rightarrow Det\ \ NP_s$
$DetP_p \rightarrow Det\ \ NP_p$

$NP_s \rightarrow Adj\ \ NP_s \mid NP_s\ \ PP \mid Noun_s$
$NP_p \rightarrow Adj\ \ NP_p \mid NP_p\ \ PP \mid Noun_p$

$PP \rightarrow Prep\ \ DetP_s \mid Prep\ \ DetP_p$

$VP_s \rightarrow Verb_s\ \ DetP_s \mid Verb_s\ \ DetP_p$
$VP_p \rightarrow Verb_p\ \ DetP_p \mid Verb_p\ \ DetP_p$

$Det \rightarrow$ the

$Noun_s \rightarrow$ plane | plant | bear | bird | car | dancer | singer
| president | squirrel | cloud | actor | doctor | nurse | chair
| student | teacher | fern
$Noun_p \rightarrow$ planes | plants | bears | birds | cars | dancers
| singers | presidents | squirrels | clouds | actors | doctors
| nurses | chairs | students | teachers | ferns

$Verb_s \rightarrow$ eats | pleases | loves | likes | hates | destroys | creates
| fights | bites | shoots | arrests | takes | leaves | buys
| brings | carries | kicks
$Verb_p \rightarrow$ eat | please | love | like | hate | destroy | create
| fight | bite | shoot | arrest | take | leave | buy | bring
| carry | kick

$Adj \rightarrow$ fancy | green | handsome | pretty | large | big | scary
| nice | happy | sad | dangerous | evil | sloppy

$Prep \rightarrow$ on | by | near | around

Figure 3: Probabilistic Context-free grammar used for creating constructed datasets. For the constructed language test set, the probabilities for the three potential expansions of NP_s and NP_p are .1, .1, .8, respectively. For the augmentation set, these probabilities are .69, .04, .27. For all other nonterminals, all possible expansions have uniform probability in both test and augmentation sets. PPs are present in approximately one third of sentences in both the test and augmentation sets.

Attractors	BiLSTM	Dependency	Constituency	Head
No	96.4%	95.5%	**97.3%**	97.2%
Any	70.8%	**91.4%**	90.2%	87.0%
1	74.6%	**91.9%**	91.3%	88.7%
2	59.7%	**89.7%**	87.1%	82.0%
3	48.4%	**87.6%**	83.7%	77.0%
4	41.0%	**87.0%**	80.8%	73.1%
Constructed	96.0%	73.8%	**97.6%**	97.3%

Table 2: Natural language results for all datasets. Best performances are bolded. All numbers are averaged over three models.

Attractors	BiLSTM	Dependency	Constituency	Head
No	88.8%	94.9%	**98.1%**	97.2%
Any	77.1%	90.1%	**91.9%**	91.5%
1	76.6%	90.8%	**93.2%**	92.8%
2	78.5%	87.9%	**88.1%**	87.6%
3	80.1%	**85.2%**	83.5%	83.3%
4	81.1%	**85.9%**	78.4%	79.4%
Constructed	95.3%	75.8%	99.7%	**99.8%**

Table 3: Results for all datasets after augmentation. All numbers are averaged over three models.