LSTMs Can Learn Syntax-Sensitive Dependencies Well, But Modeling Structure Makes Them Better

Adhiguna Kuncoro[♠♣] **Chris Dyer**[♠] **John Hale**[♠♡]
Dani Yogatama[♠] **Stephen Clark**[♠] **Phil Blunsom**[♠♣]
[♠]DeepMind, London, UK
[♣]Department of Computer Science, University of Oxford, UK
[♡]Department of Linguistics, Cornell University, NY, USA
{akuncoro,cdyer,jthale,dyogatama,clarkstephen,pblunsom}@google.com

Abstract

Language exhibits hierarchical structure, but recent work using a subject-verb agreement diagnostic argued that state-of-the-art language models, LSTMs, fail to learn long-range syntax-sensitive dependencies. Using the same diagnostic, we show that, in fact, LSTMs do succeed in learning such dependencies—provided they have enough capacity. We then explore whether models that have access to explicit syntactic information learn agreement more effectively, and how the way in which this structural information is incorporated into the model impacts performance. We find that the mere presence of syntactic information does not improve accuracy, but when model architecture is determined by syntax, number agreement is improved. Further, we find that the choice of *how* syntactic structure is built affects how well number agreement is learned: top-down construction outperforms left-corner and bottom-up variants in capturing long-distance structural dependencies.

1 Introduction

Recurrent neural networks (RNNs) are remarkably effective models of sequential data. Recent years have witnessed the widespread adoption of recurrent architectures such as LSTMs (Hochreiter and Schmidhuber, 1997) in various NLP tasks, with state of the art results in language modeling (Melis et al., 2018) and conditional generation tasks like machine translation (Bahdanau et al., 2015) and text summarization (See et al., 2017).

Here we revisit the question asked by Linzen et al. (2016): as RNNs model word sequences without explicit notions of hierarchical structure, to what extent are these models able to learn non-local syntactic dependencies in natural language? Identifying number agreement between subjects and verbs—especially in the presence of *attractors*—can be understood as a cognitively-motivated probe that seeks to distinguish hierarchical theories from sequential ones, as models that rely on sequential cues like the most recent noun would favor the incorrect verb form. We provide an example of this task in Fig. 1, where the plural form of the verb ***have*** agrees with the distant subject ***parts***, rather than the adjacent attractors (underlined) of the singular form.

Figure 1: An example of the number agreement task with two attractors and a subject-verb distance of five.

Contrary to the findings of Linzen et al. (2016), our experiments suggest that sequential LSTMs are able to capture structural dependencies to a large extent, even for cases with multiple attractors (§2). Our finding suggests that network capacity plays a crucial role in capturing structural dependencies with multiple attractors. Nevertheless, we find that a strong character LSTM language model—which lacks explicit word representation and has to capture much longer sequential dependencies in order to learn non-local structural dependencies effectively—performs much worse in the number agreement task.

Given the strong performance of word-based LSTM language models, are there are any substantial benefits, in terms of number agreement accuracy, to explicitly modeling hierarchical structures as an inductive bias? We discover that a

Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers), pages 1426–1436
Melbourne, Australia, July 15 - 20, 2018.

certain class of LSTM language models that explicitly models syntactic structures, the recurrent neural network grammars (Dyer et al., 2016, RNNGs), considerably outperforms sequential LSTM language models for cases with multiple attractors (§3). We present experiments affirming that this gain is due to an explicit *composition* operator rather than the presence of predicted syntactic annotations. Rather surprisingly, syntactic LSTM language models without explicit composition have no advantage over sequential LSTMs that operate on word sequences, although these models can nevertheless be excellent predictors of phrase structures (Choe and Charniak, 2016).

Having established the importance of modeling structures, we explore the hypothesis that *how* we build the structure affects the model's ability to identify structural dependencies in English. As RNNGs build phrase-structure trees through top-down operations, we propose extensions to the structure-building sequences and model architecture that enable left-corner (Henderson, 2003, 2004) and bottom-up (Chelba and Jelinek, 2000; Emami and Jelinek, 2005) generation orders (§4).

Extensive prior work has characterized top-down, left-corner, and bottom-up *parsing* strategies in terms of cognitive plausibility (Pulman, 1986; Abney and Johnson, 1991; Resnik, 1992) and neurophysiological evidence in human sentence processing (Nelson et al., 2017). Here we move away from the realm of parsing and evaluate the three strategies as models of *generation* instead, and address the following empirical question: which generation order is most appropriately biased to model structural dependencies in English, as indicated by number agreement accuracy? Our key finding is that the top-down generation outperforms left-corner and bottom-up variants for difficult cases with multiple attractors.

In theory, the three traversal strategies approximate the same chain rule that decompose the joint probability of words and phrase-structure trees, denoted as $p(\boldsymbol{x}, \boldsymbol{y})$, differently and as such will impose different biases on the learner. In §4.3, we show that the three variants achieve similar perplexities on a held-out validation set. As we observe different patterns in number agreement, this demonstrates that while perplexity can be a useful diagnostic tool, it may not be sensitive enough for comparing models in terms of how well they capture grammatical intuitions.

2 Number Agreement with LSTM Language Models

We revisit the number agreement task with LSTMs trained on language modeling objectives, as proposed by Linzen et al. (2016).

Experimental Settings. We use the same parsed Wikipedia corpus, verb inflectors, preprocessing steps, and dataset split as Linzen et al. (2016).[1] Word types beyond the most frequent 10,000 are converted to their respective POS tags. We summarize the corpus statistics of the dataset, along with the test set distribution of the number of attractors, in Table 1. Similar to Linzen et al. (2016), we only include test cases where *all* intervening nouns are of the opposite number forms than the subject noun. All models are implemented using the DyNet library (Neubig et al., 2017).

	Train	Test
Sentences	141,948	1,211,080
Types	10,025	10,025
Tokens	3,159,622	26,512,851

# Attractors	# Instances	% Instances
$n = 0$	1,146,330	94.7%
$n = 1$	52,599	4.3%
$n = 2$	9,380	0.77%
$n = 3$	2,051	0.17%
$n = 4$	561	0.05%
$n = 5$	159	0.01%

Table 1: Corpus statistics of the Linzen et al. (2016) number agreement dataset.

Training was done using a language modeling objective that predicts the next word given the prefix; at test time we compute agreement error rates by comparing the probability of the correct verb form with the incorrect one. We report performance of a few different LSTM hidden layer configurations, while other hyper-parameters are selected based on a grid search.[2] Following Linzen

[1] The dataset and scripts are obtained from https://github.com/TalLinzen/rnn_agreement.

[2] Based on the grid search results, we used the following hyper-parameters that work well across different hidden layer sizes: 1-layer LSTM, SGD optimizers with an initial learning rate of 0.2, a learning rate decay of 0.10 after 10 epochs, LSTM dropout rates of 0.2, an input embedding dimension of 50, and a batch size of 10 sentences. Our use of single-layer LSTMs and 50-dimensional word embedding (learned from scratch) as one of the baselines is consistent with the experimental settings of Linzen et al. (2016).

	n=0	n=1	n=2	n=3	n=4
Random	50.0	50.0	50.0	50.0	50.0
Majority	32.0	32.0	32.0	32.0	32.0
LSTM, H=50†	6.8	32.6	≈50	≈65	≈70
Our LSTM, H=50	2.4	8.0	15.7	26.1	34.65
Our LSTM, H=150	1.5	4.5	9.0	14.3	17.6
Our LSTM, H=250	1.4	3.3	5.9	**9.7**	13.9
Our LSTM, H=350	**1.3**	**3.0**	**5.7**	**9.7**	**13.8**
1B Word LSTM (repl)	2.8	8.0	14.0	21.8	20.0
Char LSTM	**1.2**	5.5	11.8	20.4	27.8

Table 2: Number agreement error rates for various LSTM language models, broken down by the number of attractors. The top two rows represent the random and majority class baselines, while the next row (†) is the reported result from Linzen et al. (2016) for an LSTM language model with 50 hidden units (some entries, denoted by ≈, are approximately derived from a chart, since Linzen et al. (2016) did not provide a full table of results). We report results of our LSTM implementations of various hidden layer sizes, along with our re-run of the Jozefowicz et al. (2016) language model, in the next five rows. We lastly report the performance of a state of the art character LSTM baseline with a large model capacity (Melis et al., 2018).

et al. (2016), we include the results of our replication[3] of the large-scale language model of Jozefowicz et al. (2016) that was trained on the One Billion Word Benchmark.[4] Hyper-parameter tuning is based on validation set perplexity.

Discussion. Table 2 indicates that, given enough capacity, LSTM language models without explicit syntactic supervision are able to perform well in number agreement. For cases with multiple attractors, we observe that the LSTM language model with 50 hidden units trails behind its larger counterparts by a substantial margin despite comparable performance for zero attractor cases, suggesting that network capacity plays an especially important role in propagating relevant structural information across a large number of steps.[5] Our experiment independently derives the same finding as the recent work of Gulordava et al. (2018), who also find that LSTMs trained with language modeling objectives are able to learn number agreement well; here we additionally identify model capacity as one of the reasons for the discrepancy with the Linzen et al. (2016) results.

While the pretrained large-scale language model of Jozefowicz et al. (2016) has certain advantages in terms of model capacity, more training data, and richer vocabulary, we suspect that the poorer performance is due to differences between their training domain and the number agreement testing domain, although the model still performs reasonably well in the number agreement test set.

Prior work has confirmed the notion that, in many cases, statistical models are able to achieve good performance under some aggregate metric by overfitting to patterns that are predictive in *most* cases, often at the expense of more difficult, infrequent instances that require deeper language understanding abilities (Rimell et al., 2009; Jia and Liang, 2017). In the vast majority of cases, structural dependencies between subjects and verbs highly overlap with sequential dependencies (Table 1). Nevertheless, the fact that number agreement accuracy gets worse as the number of attractors increases is consistent with a *sequential recency* bias in LSTMs: under this conjecture, identifying the correct structural dependency becomes harder when there are more adjacent nouns of different number forms than the true subject.

If the sequential recency conjecture is correct, then LSTMs would perform worse when the structural dependency is more distant in the sequences, compared to cases where the structural dependency is more adjacent. We empirically test this conjecture by running a strong character-based LSTM language model of Melis et al. (2018) that achieved state of the art results on EnWiki8 from the Hutter Prize dataset (Hutter, 2012), with 1,800 hidden units and 10 million parameters. The character LSTM is trained, validated, and tested[6] on the same split of the Linzen et al. (2016) number agreement dataset.

A priori, we expect that number agreement is harder for character LSTMs for two reasons. First, character LSTMs lack explicit word representa-

[3]When evaluating the large-scale language model, the primary difference is that we do not map infrequent word types to their POS tags and that we subsample to obtain 500 test instances of each number of attractor due to computation cost; both preprocessing were also done by Linzen et al. (2016).

[4]The pretrained large-scale language model is obtained from https://github.com/tensorflow/models/tree/master/research/lm_1b.

[5]This trend is also observed by comparing results with H=150 and H=250. While both models achieve near-identical performance for zero attractor, the model with H=250 performs much better for cases with multiple attractors.

[6]For testing, we similarly evaluate number agreement accuracy by comparing the probability of the correct and incorrect verb form given the prefix, as represented by the respective character sequences.

tions, thus succeeding in this task requires identifying structural dependencies between two *sequences* of character tokens, while word-based LSTMs only need to resolve dependencies between *word tokens*. Second, by nature of modeling characters, non-local structural dependencies are sequentially further apart than in the word-based language model. On the other hand, character LSTMs have the ability to exploit and share informative morphological cues, such as the fact that plural nouns in English tend to end with '*s*'.

As demonstrated on the last row of Table 2, we find that the character LSTM language model performs much worse at number agreement with multiple attractors compared to its word-based counterparts. This finding is consistent with that of Sennrich (2017), who find that character-level decoders in neural machine translation perform worse than subword models in capturing morphosyntactic agreement. To some extent, our finding demonstrates the limitations that character LSTMs face in learning structure from language modeling objectives, despite earlier evidence that character LSTM language models are able to implicitly acquire a lexicon (Le Godais et al., 2017).

3 Number Agreement with RNNGs

Given the strong performance of sequential LSTMs in number agreement, is there any further benefit to explicitly modeling hierarchical structures? We focus on recurrent neural network grammars (Dyer et al., 2016, RNNGs), which jointly model the probability of phrase-structure trees and strings, $p(\boldsymbol{x}, \boldsymbol{y})$, through structure-building actions and explicit compositions for representing completed constituents.

Our choice of RNNGs is motivated by the findings of Kuncoro et al. (2017), who find evidence for syntactic headedness in RNNG phrasal representations. Intuitively, the ability to learn heads is beneficial for this task, as the representation for the noun phrase "*The* ***flowers*** *in the vase*" would be similar to the syntactic head ***flowers*** rather than *vase*. In some sense, the composition operator can be understood as injecting a *structural recency* bias into the model design, as subjects and verbs that are sequentially apart are encouraged to be close together in the RNNGs' representation.

3.1 Recurrent Neural Network Grammars

RNNGs (Dyer et al., 2016) are language models that estimate the *joint* probability of string terminals and phrase-structure tree nonterminals. Here we use stack-only RNNGs that achieve better perplexity and parsing performance (Kuncoro et al., 2017). Given the current stack configuration, the objective function of RNNGs is to predict the correct structure-building operation according to a top-down, left-to-right traversal of the phrase-structure tree; a partial traversal for the input sentence "*The flowers in the vase are blooming*" is illustrated in Fig. 3(a).[7]

The structural inductive bias of RNNGs derives from an explicit *composition* operator that represents completed constituents; for instance, the constituent (NP *The flowers*) is represented by a single composite element on the stack, rather than as four separate symbols. During each REDUCE action, the topmost stack elements that belong to the new constituent are popped from the stack and then composed by the composition function; the composed symbol is then pushed back into the stack. The model is trained in an end-to-end manner by minimizing the cross-entropy loss relative to a sample of gold trees.

3.2 Experiments

Here we summarize the experimental settings of running RNNGs on the number agreement dataset and discuss the empirical findings.

Experimental settings. We obtain phrase-structure trees for the Linzen et al. (2016) dataset using a publicly available discriminative model[8] trained on the Penn Treebank (Marcus et al., 1993). At training time, we use these predicted trees to derive action sequences on the training set, and train the RNNG model on these sequences.[9] At test time, we compare the probabilities of the correct and incorrect verb forms given the prefix, which now includes both nonterminal and terminal symbols. An example of the stack contents (i.e. the prefix) when predicting the verb is provided in Fig. 3(a). We similarly run a grid search over the same hyper-parameter range as the sequential

[7] For a complete example of action sequences, we refer the reader to the example provided by Dyer et al. (2016).

[8] https://github.com/clab/rnng

[9] Earlier work on RNNGs (Dyer et al., 2016; Kuncoro et al., 2017) train the model on gold phrase-structure trees on the Penn Treebank, while here we train the RNNG on the number agreement dataset based on predicted trees from another parser.

LSTM and compare the results with the strongest sequential LSTM baseline from §2.

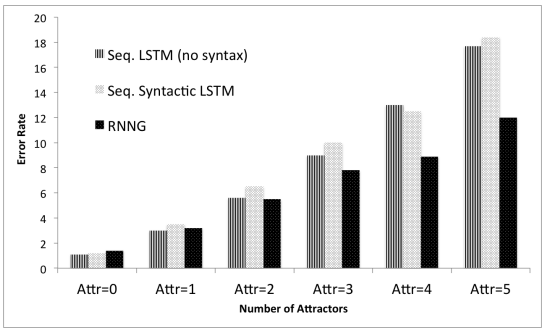

Figure 2: Number agreement error rates for sequential LSTM language models (left), sequential syntactic LSTM language models (Choe and Charniak, 2016, center), and RNNGs (right).

Discussion. Fig. 2 shows that RNNGs (rightmost) achieve much better number agreement accuracy compared to LSTM language models (leftmost) for difficult cases with four and five attractors, with around 30% error rate reductions, along with a 13% error rate reduction (from 9% to 7.8%) for three attractors. We attribute the slightly worse performance of RNNGs on cases with zero and one attractor to the presence of intervening structure-building actions that separate the subject and the verb, such as a REDUCE (step 6 in Fig. 3(a)) action to complete the noun phrase and at least one action to predict a verb phrase (step 15 in Fig. 3(a)) before the verb itself is introduced, while LSTM language models benefit from shorter dependencies for zero and one attractor cases.

The performance gain of RNNGs might arise from two potential causes. First, RNNGs have access to predicted syntactic annotations, while LSTM language models operate solely on word sequences. Second, RNNGs incorporate explicit compositions, which encourage hierarhical representations and potentially the discovery of syntactic (rather than sequential) dependencies.

Would LSTMs that have access to syntactic annotations, but without the explicit composition function, benefit from the same performance gain as RNNGs? To answer this question, we run sequential LSTMs over the same phrase-structure trees (Choe and Charniak, 2016), similarly estimating the joint probability of phrase-structure nonterminals and string terminals but without an explicit composition operator. Taking the example in Fig. 3(a), the sequential syntactic LSTM would have fifteen[10] symbols on the LSTM when predicting the verb, as opposed to three symbols in the case of RNNGs' stack LSTM. In theory, the sequential LSTM over the phrase-structure trees (Choe and Charniak, 2016) may be able to incorporate a similar, albeit implicit, composition process as RNNGs and consequently derive similarly syntactic heads, although there is no inductive bias that explicitly encourages such process.

Fig. 2 suggests that the sequential syntactic LSTMs (center) perform comparably with sequential LSTMs without syntax for multiple attractor cases, and worse than RNNGs for nearly all attractors; the gap is highest for multiple attractors. This result showcases the importance of an explicit composition operator and hierarchical representations in identifying structural dependencies, as indicated by number agreement accuracy. Our finding is consistent with the recent work of Yogatama et al. (2018), who find that introducing elements of hierarchical modeling through a stack-structured memory is beneficial for number agreement, outperforming LSTM language models and attention-augmented variants by increasing margins as the number of attractor grows.

3.3 Further Analysis

In order to better interpret the results, we conduct further analysis into the perplexities of each model, followed by a discussion on the effect of incrementality constraints on the RNNG when predicting number agreement.

Perplexity. To what extent does the success of RNNGs in the number agreement task with multiple attractors correlate with better performance under the perplexity metric? We answer this question by using an importance sampling marginalization procedure (Dyer et al., 2016) to obtain an estimate of $p(\boldsymbol{x})$ under both RNNGs and the sequential syntactic LSTM model. Following Dyer et al. (2016), for each sentence on the validation set we sample 100 candidate trees from a discriminative model[11] as our proposal distribution. As demonstrated in Table 3, the LSTM language model has the lowest validation set perplexity despite substantially worse performance than RNNGs in number agreement with multiple attractors, suggesting that lower perplexity is not neces-

[10] In the model of Choe and Charniak (2016), each nonterminal, terminal, and closed parenthesis symbol is represented as an element on the LSTM sequence.

[11] https://github.com/clab/rnng

sarily correlated with number agreement success.

	Validation ppl.
LSTM LM	**72.6**
Seq. Syntactic LSTM	79.2
RNNGs	77.9

Table 3: Validation set perplexity of LSTM language model, sequential syntactic LSTM, and RNNGs.

Incrementality constraints. As the syntactic prefix was derived from a discriminative model that has access to unprocessed words, one potential concern is that this prefix might violate the incrementality constraints and benefit the RNNG over the LSTM language model. To address this concern, we remark that the empirical evidence from Fig. 2 and Table 3 indicates that the LSTM language model *without* syntactic annotation outperforms the sequential LSTM *with* syntactic annotation in terms of both perplexity and number agreement throughout nearly all attractor settings, suggesting that the predicted syntactic prefix does not give any unfair advantage to the syntactic models.

Furthermore, we run an experiment where the syntactic prefix is instead derived from an incremental beam search procedure of Fried et al. (2017).[12] To this end, we take the highest scoring beam entry at the time that the verb is generated to be the syntactic prefix; this procedure is applied to both the correct and incorrect verb forms.[13] We then similarly compare the probabilities of the correct and incorrect verb form given each respective syntactic prefix to obtain number agreement accuracy. Our finding suggests that using the fully incremental tree prefix leads to even *better* RNNG number agreement performance for four and five attractors, achieving 7.1% and 8.2% error rates, respectively, compared to 9.4% and 12% for the RNNG error rates in Fig. 2.

4 Top-Down, Left-Corner, and Bottom-Up Traversals

In this section, we propose two new variants of RNNGs that construct trees using a different construction order than the top-down, left-to-right order used above. These are a bottom-up construction order (§4.1) and a left-corner construction order (§4.2), analogous to the well-known parsing strategies (e.g. Hale, 2014, chapter 3). They differ from these classic strategies insofar as they do not announce the phrase-structural content of an entire branch at the same time, adopting instead a node-by-node enumeration reminescent of Markov Grammars (Charniak, 1997). This step-by-step arrangement extends to the derived string as well; since all variants generate words from left to right, the models can be compared using number agreement as a diagnostic.[14]

Here we state our hypothesis on why the build order matters. The three generation strategies represent different chain rule decompositions of the joint probability of strings and phrase-structure trees, thereby imposing different biases on the learner. Earlier work in *parsing* has characterized the plausibility of top-down, left-corner, and bottom-up strategies as viable candidates of human sentence processing, especially in terms of memory constraints and human difficulties with center embedding constructions (Johnson-Laird, 1983; Pulman, 1986; Abney and Johnson, 1991; Resnik, 1992, *inter alia*), along with neurophysiological evidence in human sentence processing (Nelson et al., 2017). Here we cast the three strategies as models of language generation (Manning and Carpenter, 1997), and focus on the empirical question: which generation order has the most appropriate bias in modeling non-local structural dependencies in English?

These alternative orders organize the learning problem so as to yield intermediate states in generation that condition on different aspects of the grammatical structure. In number agreement, this amounts to making an agreement controller, such as the word *flowers* in Fig. 3, more or less salient. If it is more salient, the model should be better-able to inflect the main verb in agreement with this controller, without getting distracted by the attractors. The three proposed build orders are compared in Fig. 3, showing the respective configurations (i.e. the prefix) when generating the main verb in a sentence with a single attractor.[15] In ad-

[12] As the beam search procedure is time-consuming, we randomly sample 500 cases for each attractor and compute the number agreement accuracy on these samples.

[13] Consequently, the correct and incorrect forms of the sentence might have different partial trees, as the highest scoring beam entries may be different for each alternative.

[14] Only the order in which these models build the nonterminal symbols is different, while the terminal symbols are still generated in a left-to-right manner in all variants.

[15] Although the stack configuration at the time of verb generation varies only slightly, the configurations encountered

dition, we show concrete action sequences for a simpler sentence in each section.

4.1 Bottom-Up Traversal

In bottom-up traversals, phrases are recursively constructed and labeled with the nonterminal type once all their daughters have been built, as illustrated in Fig. 4. Bottom-up traversals benefit from shorter stack depths compared to top-down due to the lack of incomplete nonterminals. As the commitment to label the nonterminal type of a phrase is delayed until its constituents are complete, this means that the generation of a child node cannot condition on the label of its parent node.

In n-ary branching trees, bottom-up completion of constituents requires a procedure for determining how many of the most recent elements on the stack should be daughters of the node that is being constructed.[16] Conceptually, rather than having a single REDUCE operation as we have before, we have a complex REDUCE(X, n) operation that must determine the type of the constituent (i.e., X) as well as the number of daughters (i.e., n).

In step 5 of Fig. 4, the newly formed NP constituent only covers the terminal *worms*, and neither the unattached terminal *eats* nor the constituent (NP *The fox*) is part of the new noun phrase. We implement this extent decision using a stick-breaking construction—using the stack LSTM encoding, a single-layer feedforward network, and a logistic output layer—which decides whether the top element on the stack should be the leftmost child of the new constituent (i.e. whether or not the new constituent is complete after popping the current topmost stack element), as illustrated in Fig. 5. If not, the process is then repeated after the topmost stack element is popped. Once the extent of the new nonterminal has been decided, we parameterize the decision of the nonterminal label type; in Fig. 5 this is an NP. A second difference to top-down generation is that when a single constituent remains on the stack, the sentence is not necessarily complete (see step 3 of Fig. 4 for examples where this happens). We thus introduce an explicit STOP action (step 8, Fig. 4), indicating the tree is complete, which is only assigned non-zero probability when the stack has a single complete constituent.

during the history of the full generation process vary considerably in the invariances and the kinds of actions they predict.

[16]This mechanism is not necessary with strictly binary branching trees, since each new nonterminal always consists of the two children at the top of the stack.

	Avg. stack depth	Ppl.
TD	12.29	94.90
LC	11.45	95.86
BU	7.41	96.53

Table 4: Average stack depth and validation set perplexity for top-down (TD), left-corner (LC), and bottom-up (BU) RNNGs.

4.2 Left-Corner Traversal

Left-corner traversals combine some aspects of top-down and bottom-up processing. As illustrated in Fig. 6, this works by first generating the leftmost terminal of the tree, *The* (step 0), before proceeding bottom-up to predict its parent NP (step 1) and then top-down to predict the rest of its children (step 2). A REDUCE action similarly calls the composition operator once the phrase is complete (e.g. step 3). The complete constituent (NP *The fox*) is the leftmost child of its parent node, thus an NT_SW(S) action is done next (step 4).

The NT_SW(X) action is similar to the NT(X) from the top-down generator, in that it introduces an open nonterminal node and must be matched later by a corresponding REDUCE operation, but, in addition, swaps the two topmost elements at the top of the stack. This is necessary because the parent nonterminal node is not built until after its left-most child has been constructed. In step 1 of Fig. 6, with a single element *The* on the stack, the action NT_SW(NP) adds the open nonterminal symbol NP to become the topmost stack element, but after applying the swap operator the stack now contains (NP | *The* (step 2).

4.3 Experiments

We optimize the hyper-parameters of each RNNG variant using grid searches based on validation set perplexity. Table 4 summarizes average stack depths and perplexities[17] on the Linzen et al. (2016) validation set. We evaluate each of the variants in terms of number agreement accuracy as an evidence of its suitability to model structural dependencies in English, presented in Table 5. To account for randomness in training, we report the error rate summary statistics of ten different runs.

[17]Here we measure perplexity over $p(\boldsymbol{x}, \boldsymbol{y})$, where y is the presumptive gold tree on the Linzen et al. (2016) dataset. Dyer et al. (2016) instead used an importance sampling procedure to marginalize and obtain an estimate of $p(\boldsymbol{x})$.

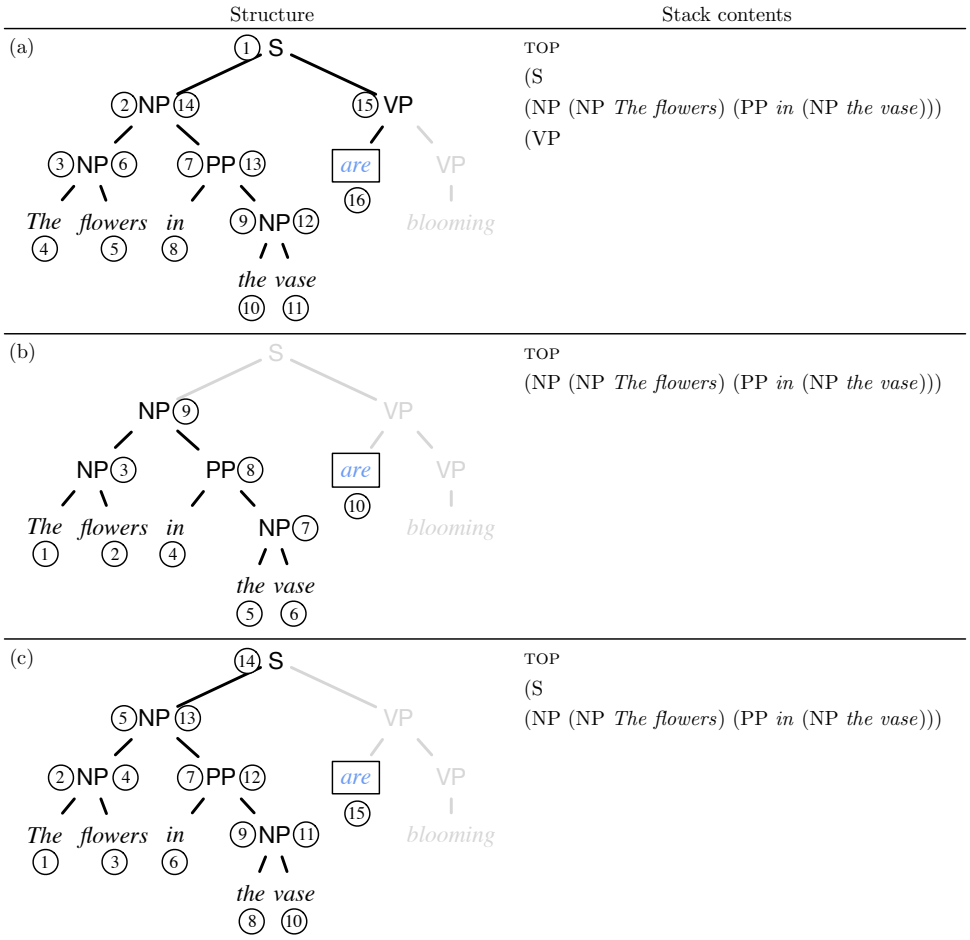

Figure 3: The (a) top-down, (b) bottom-up, and (c) left-corner build order variants showing in black the structure that exists as well as the generator's stack contents when the word *are* is generated during the derivation of the sentence *The flowers in the vase are blooming*. Structure in grey indicates material that will be generated subsequent to this. Circled numbers indicate the time when the corresponding structure/word is constructed. In (a) and (c), nonterminals are generated by a matched pair of NT and REDUCE operations, while in (b) they are introduced by a single complex REDUCE operation.

Input: *The fox eats worms*

	Stack	**Action**
0		GEN(*The*)
1	*The*	GEN(*fox*)
2	*The* | *fox*	REDUCE(NP,2)
3	(NP *The fox*)	GEN(*eats*)
4	(NP *The fox*) | *eats*	GEN(*worms*)
5	(NP *The fox*) | *eats* | *worms*	REDUCE(NP,1)
6	(NP *The fox*) | *eats* | (NP *worms*)	REDUCE(VP,2)
7	(NP *The fox*) | (VP *eats* (NP *worms*))	REDUCE(S,2)
8	(S (NP *The fox*) (VP *eats* (NP *worms*))	STOP

Figure 4: Example Derivation for Bottom-Up Traversal. '|' indicates separate elements on the stack. The REDUCE(X, n) action takes the top n elements on the stack and creates a new constituent of type X with the composition function.

	Avg.(±sdev)/min/max		
	n=2	**n=3**	**n=4**
LM	5.8(±0.2)/5.5/6.0	9.6(±0.7)/8.8/10.1	14.1(±1.2)/13.0/15.3
TD	5.5(±0.4)/4.9/5.8	**7.8**(±0.6)/7.4/8.0	**8.9**(±1.1)/7.9/9.8
LC	**5.4**(±0.3)/5.2/5.5	8.2(±0.4)/7.9/8.7	9.9(±1.3)/8.8/11.5
BU	5.7(±0.3) 5.5/5.8	8.5(±0.7)/8.0/9.3	9.7(±1.1)/9.0/11.3

Table 5: Number agreement error rates for top-down (TD), left-corner (LC), and bottom-up (BU) RNNGs, broken down by the number of attractors. LM indicates the best sequential language model baseline (§2). We report the mean, standard deviation, and minimum/maximum of 10 different random seeds of each model.

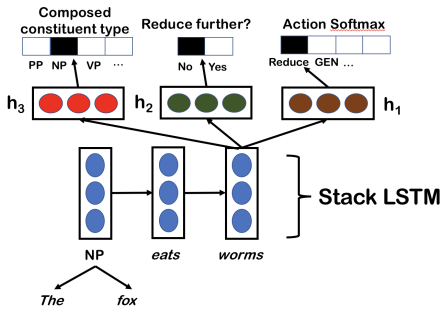

Figure 5: Architecture to determine type and span of new constituents during bottom-up generation.

Input: *The fox eats worms*

	Stack	Action
0		GEN(*The*)
1	*The*	NT_SW(NP)
2	(NP | *The*	GEN(*fox*)
3	(NP | *The* | *fox*	REDUCE
4	(NP *The fox*)	NT_SW(S)
5	(S | (NP *The fox*)	GEN(*eats*)
6	(S | (NP *The fox*) | *eats*	NT_SW(VP)
7	(S | (NP *The fox*) | (VP | *eats*	GEN(*worms*)
8	(S | (NP *The fox*) | (VP | *eats* | *worms*	NT_SW(NP)
9	(S | (NP *The fox*) | (VP | *eats* | (NP | *worms*	REDUCE
10	(S | (NP *The fox*) | (VP | *eats* | (NP *worms*)	REDUCE
11	(S | (NP *The fox*) | (VP *eats* (NP *worms*))	REDUCE
12	(S (NP *The fox*) (VP *eats* (NP *worms*)))	N/A

Figure 6: Example Derivation for left-corner traversal. Each NT_SW(X) action adds the open nonterminal symbol (X to the stack, followed by a deterministic **swap** operator that swaps the top two elements on the stack.

Discussion. In Table 5, we focus on empirical results for cases where the structural dependencies matter the most, corresponding to cases with two, three, and four attractors. All three RNNG variants outperform the sequential LSTM language model baseline for these cases. Nevertheless, the top-down variant outperforms both left-corner and bottom-up strategies for difficult cases with three or more attractors, suggesting that the top-down strategy is most appropriately biased to model difficult number agreement dependencies in English. We run an approximate randomization test by stratifying the output and permuting within each stratum (Yeh, 2000) and find that, for four attractors, the performance difference between the top-down RNNG and the other variants is statistically significant at $p < 0.05$.

The success of the top-down traversal in the domain of number-agreement prediction is consistent with a classical view in parsing that argues top-down parsing is the most human-like parsing strategy since it is the most *anticipatory*. Only anticipatory representations, it is said, could explain the rapid, incremental processing that humans seem to exhibit (Marslen-Wilson, 1973; Tanenhaus et al., 1995); this line of thinking similarly motivates Charniak (2010), among others. While most work in this domain has been concerned with the parsing problem, our findings suggest that anticipatory mechanisms are also beneficial in capturing structural dependencies in language modeling. We note that our results are achieved using models that, in theory, are able to condition on the entire derivation history, while earlier work in sentence processing has focused on cognitive memory considerations, such as the memory-bounded model of Schuler et al. (2010).

5 Conclusion

Given enough capacity, LSTMs trained on language modeling objectives are able to learn syntax-sensitive dependencies, as evidenced by accurate number agreement accuracy with multiple attractors. Despite this strong performance, we discover explicit modeling of structure does improve the model's ability to discover non-local structural dependencies when determining the distribution over subsequent word generation. Recurrent neural network grammars (RNNGs), which jointly model phrase-structure trees and strings and employ an explicit composition operator, substantially outperform LSTM language models and syntactic language models without explicit compositions; this highlights the importance of a hierarchical inductive bias in capturing structural dependencies. We explore the possibility that how the structure is built affects number agreement performance. Through novel extensions to RNNGs that enable the use of left-corner and bottom-up generation strategies, we discover that this is indeed the case: the three RNNG variants have different generalization properties for number agreement, with the top-down traversal strategy performing best for cases with multiple attractors.

Acknowledgments

We would like to thank Tal Linzen for his help in data preparation and answering various questions. We also thank Laura Rimell, Nando de Freitas, and the three anonymous reviewers for their helpful comments and suggestions.

References

Steven Abney and Mark Johnson. 1991. Memory requirements and local ambiguities for parsing strategies. *Journal of Psycholinguistic Research* .

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proc. of ICLR*.

Eugene Charniak. 1997. Statistical techniques for natural language parsing. *AI Magazine* .

Eugene Charniak. 2010. Top-down nearly-context-sensitive parsing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Cambridge, MA, pages 674–683. http://www.aclweb.org/anthology/D10-1066.

Ciprian Chelba and Frederick Jelinek. 2000. Structured language modeling. *Computer Speech and Language* 14(4).

Do Kook Choe and Eugene Charniak. 2016. Parsing as language modeling. In *Proc. of EMNLP*.

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

Ahmad Emami and Frederick Jelinek. 2005. A neural syntactic language model. *Machine Learning* 60:195–227.

Daniel Fried, Mitchell Stern, and Dan Klein. 2017. Improving neural parsing by disentangling model combination and reranking effects. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Vancouver, Canada, pages 161–166.

Kristina Gulordava, Piotr Bojanowski, Edouard Grave, Tal Linzen, and Marco Baroni. 2018. Colorless green recurrent networks dream hierarchically. In *Proc. of NAACL*.

John T Hale. 2014. *Automaton theories of human sentence comprehension*. CSLI Publications.

James Henderson. 2003. Inducing history representations for broad coverage statistical parsing. In *Proc. of NAACL*.

James Henderson. 2004. Discriminative training of a neural network statistical parser. In *Proc. of ACL*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* .

Marcus Hutter. 2012. The human knowledge compression contest .

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. In *Proc. of EMNLP*.

Philip N. Johnson-Laird. 1983. *Mental Models*. Harvard University Press.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling.

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 *Proc. of EACL*.

Gaël Le Godais, Tal Linzen, and Emmanuel Dupoux. 2017. Comparing character-level neural language models using a lexical decision task. In *Proc. of EACL*.

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* .

Christopher D. Manning and Bob Carpenter. 1997. Probabilistic parsing using left corner language models. In *Proc. of IWPT*.

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

William Marslen-Wilson. 1973. Linguistic structure and speech shadowing at very short latencies. *Nature* 244:522–523.

Gabor Melis, Chris Dyer, and Phil Blunsom. 2018. On the state of the art of evaluation in neural language models. In *Proc. of ICLR*.

Matthew J. Nelson, Imen El Karoui, Kristof Giber, Xiaofang Yang, Laurent Cohen, Hilda Koopman, Sydney S. Cash, Lionel Naccache, John T. Hale, Christophe Pallier, and Stanislas Dehaene. 2017. Neurophysiological dynamics of phrase-structure building during sentence processing. *Proceedings of the National Academy of Sciences of the United States of America* .

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980* .

Stephen Pulman. 1986. Grammars, parsers, and memory limitations. *Language and Cognitive Processes* .

Philip Resnik. 1992. Left-corner parsing and psychological plausibility. In *Proc. of COLING*.

Laura Rimell, Stephen Clark, and Mark Steedman. 2009. Unbounded dependency recovery for parser evaluation. In *Proc. of EMNLP*.

William Schuler, Samir AbdelRahman, Tim Miller, and Lane Schwartz. 2010. Broad-coverage parsing using human-like memory constraints 36(1):1–30.

Abigail See, Peter Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proc. of ACL*.

Rico Sennrich. 2017. How grammatical is character-level neural machine translation? assessing mt quality with contrastive translation pairs. In *Proc. of EACL*.

Michael Tanenhaus, Michael Spivey-Knowlton, Kathleen Eberhard, and Julie Sedivy. 1995. Integration of visual and linguistic information in spoken language comprehension. *Science* 268:1632–1634.

Alexander Yeh. 2000. More accurate tests for the statistical significance of result differences. In *Proc. of COLING*.

Dani Yogatama, Yishu Miao, Gabor Melis, Wang Ling, Adhiguna Kuncoro, Chris Dyer, and Phil Blunsom. 2018. Memory architectures in recurrent neural network language models. In *Proc. of ICLR*.