Squib

On Learning Interpreted Languages with Recurrent Models

Denis Paperno
Utrecht University
Department of Languages,
Literature and Communication
d.paperno@uu.nl

Can recurrent neural nets, inspired by human sequential data processing, learn to understand language? We construct simplified data sets reflecting core properties of natural language as modeled in formal syntax and semantics: recursive syntactic structure and compositionality. We find LSTM and GRU networks to generalize to compositional interpretation well, but only in the most favorable learning settings, with a well-paced curriculum, extensive training data, and left-to-right (but not right-to-left) composition.

1. Introduction

Common concerns in NLP are that current models tend to be very data hungry and rely on shallow cues and biases in the data (e.g., Geva, Goldberg, and Berant 2019), failing at deeper, human-like generalization. In this article, we present a new task of acquiring compositional generalization from a small number of linguistic examples. Our setup is inspired by properties of language understanding in humans.

First, natural language exploits recursion: Natural language syntax consists of constructions, represented in formal grammars as rewrite rules, which can recursively embed other constructions of the same kind. For example, noun phrases can consist of a single proper noun (e.g., *Ann*) but can also, among other possibilities, be built from other noun phrases recursively via the possessive construction, as in *Ann's child*, *Ann's child's friend*, *Ann's child's friend's parent*, and so forth.

Second, recursive syntactic structure drives semantic interpretation. The meaning of the noun phrase *Ann's child's friend* is not the sum of the meanings of the individual words (otherwise it would have been equivalent to *Ann's friend's child*). To interpret a complex expression, one has to follow its syntactic structure, first identifying the meaning of the subconstituent (*Ann's friend*), and then computing the meaning of the whole. Semantic composition, argued to be a core component of language processing in the human brain (Mollica et al. 2020), can be formalized as function application, where one constituent in a complex structure corresponds to an argument of a function that another constituent encodes. For instance, in *Ann's child*, we can think of *Ann* as

Submission received: 29 February 2020; revised version received: 6 July 2021; accepted for publication: 19 August 2021.

https://doi.org/10.1162/COLI_a_00431

denoting an individual and *child* as denoting a function from individuals to individuals. In formal semantics, function argument application is the basic compositionality mechanism, extending to a wide range of constructions. The focus on recursion in semantic composition differentiates our approach from other attempts at examining compositional properties of neural systems (Ettinger et al. 2018; Soulos et al. 2019; Andreas 2019; Mickus, Bernard, and Paperno 2020). Kim and Linzen (2020) include depth of recursion as one of the many aspects of systematic semantic generalization. The work on compositionality learning most closely related to ours relies on artificial languages of arithmetic expressions and sequence operations (Hupkes, Veldhoen, and Zuidema 2018; Nangia and Bowman 2018; Hupkes et al. 2020).

Third, natural language interpretation, while being sensitive to syntactic structure, is robust to syntactic variation. For example, humans are equally capable of learning to interpret and using left-branching structures such as NP $\longrightarrow$ NP*'s* N (*Ann's child*) and right-branching structures such as NP $\longrightarrow$ *the N of NP* (*the child of Ann*). Finally, language processing in humans is sequential; people process and interpret linguistic input on the fly, without much lookahead or waiting for the linguistic structure to be completed. This property has diverse consequences for language and cognition (Christiansen and Chater 2016), and gives a certain degree of cognitive plausibility to *unidirectional recurrent models* compared to other neural architectures.

To summarize, in order to mimic human language capacities, an artificial system must be able to 1) learn languages with compositionally interpreted recursive structures, 2) adapt to surface variation in the syntactic patterns, and 3) process its inputs sequentially. Recurrent neural networks are promising with respect to these desiderata; in a syntactic task (Lakretz et al. 2021), they demonstrated similar error patterns to humans in processing recursive structures. We proceed now to define a semantic task that allows us to directly test models on these capacities.

2. The Task

Our approach builds closely on natural language and systematically explores the factor of syntactic structure (right vs. left branching). We define toy interpreted languages based on a fragment of English. Data was generated with the following grammars:

Left-branching language: NP $\longrightarrow$ NAME, NP $\longrightarrow$ NP 's RN

Right-branching language: NP $\longrightarrow$ NAME, NP $\longrightarrow$ the RN of NP

Both: NAME: Ann, Bill, Garry, Donna; RN: child, parent, friend, enemy

Interpretation is defined model-theoretically. We randomly generate a model where each proper name corresponds to a distinct individual and each function denoted by a common noun is total—for example, every individual has exactly one enemy, exactly one friend, and so on. This total function interpretation simplifies natural language patterns but is an appropriate idealization for our purposes. An example with all relations is given in Figure 1.

In such a model, each well-formed expression of the language is an individual identifier. The denotation of any expression regardless of its complexity (= number of content words) can be calculated by recursive application of functions to arguments, guided by the syntactic structure.

The task given to the models is to identify the individual that corresponds to each expression; for example *Ann's child's enemy* (complexity 3) is the same person as *Bill* (complexity 1). Because there is just a finite number of individuals in any given model,

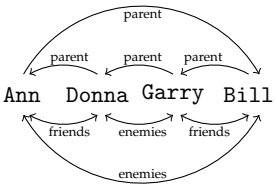

Figure 1
An example of a toy universe with relations indicated by arrows. The *child* relation is the inverse of *parent*.

the task formally boils down to string classification, assigning each expression to one of the set of individuals in the model. For examples of strings and individuals they refer to, see Table 1.

By its nature, solving the task presumes (implicitly) mastering parsing and semantic composition at the same time. Parsing is deciding what pieces of information from different points in the input string to combine in what order, for example, in *the father of the friend of Garry* one starts by combining *friend* and *Garry*, then the result with *father*. Semantic composition is performing the actual combination, in this case by recursively applying functions to arguments: `father(friend(garry))`. Because of how the relations in the universe are randomly generated every time, there is no consistent bias for the model to exploit. So without a compositional strategy, the value of a held-out complex example cannot be reliably determined. Therefore, if a system maps unseen complex expressions (*the father of the friend of Garry*) to their referents with high accuracy, it must have approximated a compositional solution of the task.

3. Data Sets

We use languages with four individuals and four relation nouns. The interpretations of the relation nouns were generated randomly while making sure that "friend" and "enemy" are symmetric and "parent" and "child" are antisymmetric and inverse of each other. The interpretation of the four function elements was randomly assigned

Table 1
Some examples from a data set with top complexity 3 illustrating the targeted semantic generalization.

Example	Referent	Complexity	Partition
Garry	Garry	1	train
Bill's enemy	Ann	2	train
Ann's friend	Donna	2	train
Donna's parent	Garry	2	train
Ann's child	Donna	2	train
Garry's enemy's parent	Ann	3	train
Ann's friend's parent	Garry	3	validation
Bill's enemy's child	Donna	3	test

Table 2
Data set sizes for each maximal complexity value.

Test ex. complexity:	3	4	5	6	7
Train	71	288	1,159	4,640	18,567
Dev	7	28	112	451	1,802
Test	6	24	93	369	1,475
Total	84	340	1,364	5,460	21,844

for model initialization. We used all expressions of the language up to complexity n as data; validation and testing data was randomly selected among examples of the maximal complexity n. Expressions of complexity 1 and 2 were always included in the training partition because they are necessary to learn the interpretation of lexical items. For example, the simplest set of training data (up to complexity 3) contained all names (examples of complexity 1), required to learn the individuals; all expressions with 2 content words like *Ann's child*, necessary for learning the meanings of functional words like *child*; and a random subset of three content word expressions like *Ann's child's friend*, which might guide the systems to learning recursion. To further simplify the task, each data set contained either only left branching or only right branching examples in all three partitions (training/validation/testing). Data set sizes are given in Table 2, which reports the default setup with 80% examples of maximum complexity included in training data and the rest distributed between validation and test data. In this setup, the proportions of data partitions are approximately 85% training, 8% validation, and 7% testing.

4. Models and Training

We tested the learning capacities of standard recurrent neural models on our task: a vanilla recurrent neural network (SRN; Elman 1991), a long short-term memory network (LSTM; Hochreiter and Schmidhuber 1997), and a gated recurrent unit (GRU; Cho et al. 2014).[1] The systems were implemented in PyTorch and used a single hidden layer of 256 units. Using more layers or fewer units did not substantially improve, and sometimes dramatically deteriorated, the performance of the models. Models were trained from a randomly initialized state on tokenized input (e.g., `["Ann","s","child"]`). The training used the Adam optimizer and negative log likelihood loss for sequence classification. Each model was run repeatedly with different random initializations. Data sets were randomly generated for each initialization of each model.

We also set a curriculum whereby the system is given training examples of minimal complexity first, with more complex examples added gradually in the process of training. The best curriculum settings varied by model and test example complexity, but on average, the most robust results were obtained by adding examples of the next complexity level after every ten epochs (*gentle curriculum*). A curriculum very consistently led to an improvement over the alternative training whereby training examples

1 Code used in the article is available at `https://github.com/dpaperno/LSTM_composition`.

Table 3
Accuracy of gated architectures as a function of the language and input data complexity. Results averaged over 10 random initializations of an LSTM and 10 random initializations of a GRU model. Training data in each run includes examples of complexity up to and including n, with testing data (disjoint from training) of complexity exactly n. Random baseline is 0.25. For more details, see Appendix, Section 1.

Test ex. complexity:	**3**	**4**	**5**	**6**	**7**
Right branching	0.09	0.35	0.16	0.3	0.17
Left branching	**0.94**	**0.92**	**0.88**	**0.92**	**0.92**
Left, no curriculum	0.34	0.25	0.29	0.3	0.28

of all complexity levels were available to the models at all epochs. For more details on alternative curricula, see the Appendix, Section 1.

5. Results

To treat complex expressions, the successful model needs to generalize by learning to compose the representations of simple expressions recursively. But representations of simple expressions (complexity 1 and 2) have to be memorized in one form or another because their interpretation is arbitrary; without such memorization, generalization to complex inputs is impossible.

As in related studies (e.g., Liska, Kruszewski, and Baroni 2018), SRN was behind the gated architectures in accuracy, with a dramatic gap for longer examples. Accuracies of gated models are summarized in Table 3; for more details, see Appendix, Section 1. We find that gated architectures learn compositional interpretation in our task only in the best conditions. A curriculum proved essential for recursive generalization. Informally, the system has to learn to interpret words first, and recursive semantic composition has to be learned later.

All recurrent models generalized only to left-branching structures; the accuracy in the right branching case is much lower. This means that the systems only learn to apply composition following the linear sequence of the input and fail when the order of composition as determined by the syntactic structure runs opposite to the linear order. In other words, the system manages to learn composition when the parse trivially corresponds to the default processing sequence, but fails when nontrivial parsing is involved. Therefore, semantic composition is mastered, but not simultaneously with the independent task of parsing (cf. discussion in Section 2).

6. Zero-shot Compositionality Testing

How much recursive input does a successful system need to master recursive interpretation? Ideally, learners with a strong bias toward languages with recursive syntactic structure (not an implausible assumption when it comes to human language learners) could acquire them in a zero-shot fashion, without any training examples that feature recursive structures. For example, assume that such a learner knows that the name *Donna* and the phrase *Ann's child* refer to the same individual, and that *Donna's enemy* refers to Bill. Without any exposure to examples longer than *Donna's enemy*, the learner could still infer that *Ann's child's enemy* of unseen length 3 is also Bill. In a recurrent neural network, the expectation can be interpreted as follows: Both *Donna* and the

Table 4
LSTM's data hungriness in learning recursion. Percentage of complexity 3 data included in training vs. test accuracy. We report average accuracy over 10 runs (each with a random initialization of the model and data generation) as well as the share of runs with perfect accuracy. Random baseline for accuracy is 0.25.

Share of recursive examples included in training	0.0	0.2	0.4	0.6	0.8
Training/testing examples	20/29	32/24	45/18	58/12	71/6
Share of training examples that are recursive	0.0	0.375	0.556	0.655	0.718
Average accuracy	0	0.61	0.76	0.89	**0.98**
Perfect accuracy	0	0	0	0.4	**0.9**

phrase *Ann's child* are expected to be mapped to similar hidden states, and because one of those hidden states allow the system to identify Bill after processing *'s enemy* in *Donna's enemy*, the same can be expected for the phrase *Ann's child's enemy*. To test whether such zero-shot (or few-shot) recursion capacity actually arises, we train the model while varying the amount of recursion examples available as training data from 0% to 80% of all examples of complexity 3. As shown in Table 4, the LSTM needs to be trained on a vast majority of recursive examples to be able to generalize. Similar results for GRU and SRN are reported in the Appendix, Section 2.

Recursive compositionality does not come for free and has to be learned from extensive data. This observation goes in line with negative findings in literature on compositionality learning (Liska, Kruszewski, and Baroni 2018; Hupkes et al. 2018; Lake and Baroni 2018; Ruis et al. 2020). Contrary to some of those findings, we do observe generalization to bigger structures *after* substantial evidence for a compositional solution is available to the system. GRU and LSTM models trained on examples of complexities 1–3 perform well on examples of unseen greater lengths (96% and 99% accuracy on complexity 4, respectively; 95% and 98% on complexity 5 etc.). Only on complexity 7 did the trained LSTM show somewhat degraded performance (77% accuracy), with GRU maintaining robust generalization (96% accuracy). For detailed results, see Appendix, Section 3. Interestingly, models trained on complexities 1–3 and generalizing recursively to unseen lengths sometimes performed better than the same models exposed to more complex inputs during training (Table 3). This further strengthens our conclusions above about the importance of curriculum: Complex training examples might be detrimental for recursive generalization at late as well as early learning stages.

What drives generalization success in these settings? We hypothesize that the RNNs are good at approximating a finite state solution to the interpretation problem, which could run as follows: When processing the token *Ann*, an automaton would transition to the state 'Ann'; after processing *'s child*, they transition from 'Ann' to the state 'Donna'; and so forth. The state of the automaton after processing the whole sequence corresponds to the referent of the phrase. This particular finite state solution relies on left-to-right processing and only applies to left-branching structures (*Ann's child's friend*) but not to right-branching ones (*the friend of the child of Ann*).

7. Discussion: The Left-Branching Asymmetry of Recurrent Models

In our experiments, recurrent models generalize reliably only to left-branching structures. This suggests that RNNs, despite greater theoretical capacities, remain well-suited

to sequential, left-to-right processing and do not learn a fully recursive solution to the interpretation problem. As literature suggests, recurrent models can generalize to non-left-branching data patterns if the language allows for a simpler interpretation strategy that minimizes computation over recursive structures, such as the "cumulative" strategy of Hupkes, Veldhoen, and Zuidema (2018). However, a cumulative interpretation strategy might not be an easy solution to learn for natural language or artificial languages. Even in Hupkes et al., where a GRU model showed reasonable average performance for mixed-directionality structures, a much lower error level is reported for left branching examples and a much higher error level for right branching examples.

The left-to-right branching asymmetry of recurrent models is doubtlessly related to their natural representational structure. When processing a left-branching structure left to right, at any point the system needs to "remember" only one element, the intermediate result of computation (an individual in our toy language, or a number in the arithmetic language). In the case of our language, the incremental interpretation of the left-branching model keeps track of the current individual (e.g., "Ann") and switches to the next individual according to the next relation term encountered (e.g., "friend"). For a universe with k individuals, a sequential model must distinguish only k states. If, however, one interprets right-branching examples (*the father of the friend of Ann*), one has to keep a representation of a function to be applied to the individual whose name follows (example of such a function: *the father of the friend*). There are k^k $U \rightarrow U$ functions over the domain of individuals U, so the intermediate representation space must be significantly richer than in the left-branching case where only k states suffice. The number of states is a problem not only from the viewpoint of model expressiveness, but also from the learning point of view: It is not realistic to observe all k^k functions at training time.

The brute force approach to left-to-right parsing or interpretation consists in keeping a stack of representations to be integrated at a later step; but in practice learned recurrent neural models are known to be too poor representationally to emulate stacks even if they are more powerful in practice than finite state machines (Weiss, Goldberg, and Yahav 2018; Bernardy 2018). As Hupkes et al.'s analysis suggests, GRUs can emulate stack behavior to a restricted degree. The cumulative strategy that they argue the GRU learns makes use of stacks of binary values, but generalizing stack behavior to an unseen situation that required novel stack values apparently fails, as evidenced by the dramatically low performance on right branching examples of unseen complexity. Hupkes et al.'s models thus probably learn to operate only over small stacks by memorizing specific stack states and transitions between them (for their task, the relevant stack states are few in number). Their recurrent models can therefore be interpreted as finding "approximate solutions by using the stack as unstructured memory" (Hao et al. 2018).

What solutions do recurrent models learn when they fail to generalize compositionally in the right branching case? To find out, we analyzed the predictions of SRN, GRU, and LSTM models for right branching languages, and found a consistent pattern. They do learn names correctly (e.g., *Ann*), but struggle when it comes to anything more complex. In a right branching input language, all models tend to "forget" long distance information and ignore anything but the suffix, assigning the same referent to *the enemy of Ann*, *the friend of the enemy of Ann*, *the child of the enemy of Ann*, and so forth. Often, but not always, models also overfit, predicting for complex examples the referent which occurred more often than others in the training data, even by a small margin. None of these observations pertain to learning left-branching patterns, which are essentially a

mirror image of right branching ones, although for them an RNN could just as easily fall back on suffix matching or the majority class. Instead, a converged model for left-branching data proceeds incrementally as per the expectations outlined above, correctly classifying *Donna* as *Donna*, *Donna's parent* as *Garry*, *Donna's parent's friend* as *Bill*, and so on for longer inputs. Crucial earlier parts of the string affect the output, and the majority class of the training data does not tend to come up as a prediction.

8. Conclusion

The results reported in this squib both are encouraging and point to limitations of recurrent models. RNNs do learn compositional interpretation from data of small absolute size and generalize to more complex examples; certain favorable conditions are required, including a curriculum and a left branching language. Our findings are cautiously optimistic compared to claims in the literature about neural nets' qualitatively limited generalization capacity (Baroni 2019). In the future, we plan to further explore the capacities of diverse models on our task. Indeed, other architectures might contain reasonable biases toward processing recursive languages not limited to the left branching case. Would it be beneficial, for instance, to augment the recurrent architecture with stack memory (Joulin and Mikolov 2015; Yogatama et al. 2018), add a chart parsing component (Le and Zuidema 2015; Maillard, Clark, and Yogatama 2017), or switch from sequential to attention-based processing altogether (Vaswani et al. 2017)? Or would this be possible only at the cost of the remarkable data efficiency we observed in recurrent models?

One idea seems especially appealing: If unidirectional RNN models can handle left recursive branching while right recursion is its mirror image, could bidirectional RNNs offer a general solution? There are, however, reasons for skepticism. Bidirectional models showed mixed results in our preliminary experiments, although further hyperparameter tuning and stricter regularization might lead to improvement. Further, even if biRNNs master both branching directions separately, they may still struggle with mixed structures which are common in natural language.

Lastly, whether human language learning has similar properties to those of RNNs presents an interesting area for exploration. Do humans, like our RNNs, require substantial data to learn recursion? And do humans, like our RNNs, have a bias toward left branching structures, which are easy for recursive sequential processing? (e.g., *Garry's father*, rather than *the father of Garry*). Suggestive evidence for the latter: Left-branching possessive constructions emerge in infant speech even in languages that don't have them, for example, *Yael sefer* 'Yael's book' (Hebrew, Armon-Lotem 1998) or *zia trattore* 'aunt's tractor' (Italian, Torregrossa and Melloni 2014).

Appendices

1. Recurrent Architectures and Alternative Curricula

LSTM, GRU, and Elman's Simple Recurrent Network (SRN) were implemented in Python 3.7.0 using built-in functions of PyTorch 1.5.0. Each model was trained for 100 epochs or until no improvement on the validation set was observed for 22 epochs. A number of curricula was compared. The default "gentle" curriculum consisted in adding examples of the next complexity level after every 10 epochs. No curriculum

Table A.1
Accuracy of LSTM as a function of the language and input data complexity. Training data in each run includes examples of complexity up to and including n; testing data (disjoint from training) contained examples of complexity exactly n. Random baseline is 0.25.

Test ex. complexity:	**3**	**4**	**5**	**6**	**7**
Right branching	0.02	0.29	0.12	0.23	0.1
Left branching	0.9	**0.9**	**0.76**	**0.93**	**0.9**
Left, no curriculum	0.23	0.2	0.24	0.26	0.25
Left, steep curriculum	**0.95**	0.68	0.71	0.33	0.33
Left, slow curriculum	0.42	0.73	0.55	0.69	0.61

Table A.2
Accuracy of GRU as a function of the language and input data complexity. Training data in each run includes examples of complexity up to and including n; testing data (disjoint from training) contained examples of complexity exactly n. Random baseline is 0.25.

Test ex. complexity:	**3**	**4**	**5**	**6**	**7**
Right branching	0.17	0.42	0.21	0.37	0.25
Left branching	0.98	0.95	**1**	0.92	0.94
Left, no curriculum	0.45	0.3	0.34	0.34	0.31
Left, steep curriculum	**1**	0.99	0.81	0.43	0.43
Left, slow curriculum	**1**	**1**	0.997	**1**	**0.997**

Table A.3
Accuracy of SRN as a function of the language and input data complexity. Training data in each run includes examples of complexity up to and including n; testing data (disjoint from training) contained examples of complexity exactly n. Random baseline is 0.25.

Test ex. complexity:	**3**	**4**	**5**	**6**	**7**
Right branching	0.13	0.28	0.09	0.28	0.15
Left branching	0.82	0.4	0.39	0.34	0.33
Left, no curriculum	**0.89**	0.28	0.33	0.23	0.32
Left, steep curriculum	0.8	0.36	0.42	0.36	0.34
Left, slow curriculum	0.77	**0.64**	**0.47**	**0.45**	**0.38**

means that all training examples were used at all epochs. A slow curriculum consisted in adding examples of the next complexity level after every 20 epochs. A steep curriculum consisted in adding training examples of complexity 2 after 10 epochs and all other training examples after another 10 epochs. Results of the three architectures for all curricula are given in Tables A.1, A.2, and A.3.

2. Data Hungriness for Different Architectures

Tables A.4, A.5, and A.6 report the few-shot compositional generalization of different recurrent models. The models were trained on examples of complexity 1 and 2 and some proportion of examples of complexity 3 (i.e., minimal recursive examples). The proportion of recursive examples ranged from 0.0 to 0.8.

Table A.4
Data hungriness for learning recursion by an LSTM at complexity 3; percentage of data complexity 3 included in training data vs. test accuracy. The results are based on 10 runs with different random seeds. We report average accuracy as well as the share of runs with perfect accuracy. Random baseline for accuracy is 0.25.

Rec. in train	0.0	0.2	0.4	0.6	0.8
Average accuracy	0	0.61	0.76	0.89	**0.98**
Perfect accuracy	0	0	0	0.4	**0.9**

Table A.5
Data hungriness for learning recursion by a GRU at complexity 3; percentage of data complexity 3 included in training data vs. test accuracy. The results are based on 10 runs with different random seeds. We report average accuracy as well as the share of runs with perfect accuracy. Random baseline for accuracy is 0.25.

Rec. in train	0.0	0.2	0.4	0.6	0.8
Average accuracy	0.69	0.9	0.83	0.98	**1**
Perfect accuracy	0.3	0	0.2	0.9	**1**

Table A.6
Data hungriness for learning recursion by an SRN at complexity 3; percentage of data complexity 3 included in training data vs. test accuracy. The results are based on 10 runs with different random seeds. We report average accuracy as well as the share of runs with perfect accuracy. Random baseline for accuracy is 0.25.

Rec. in train	0.0	0.2	0.4	0.6	0.8
Average accuracy	0.75	0.66	0.61	**0.88**	0.78
Perfect accuracy	0	0	0	**0.6**	**0.6**

3. Generalization to Unseen Lengths

To illustrate generalization to unseen (high) lengths, we report the performance of different models when trained on data of complexity 1 through 3 and tested on higher complexities (Table A.7) or when trained on data of complexity 1 through n and tested on data of complexity $n + 1$ (Table A.8).

Table A.7
Generalization of recurrent models from training examples of complexity 1–3 to examples of higher complexity. Accuracy averaged over 10 runs. Random baseline is 0.25.

Test ex. complexity:	**4**	**5**	**6**	**7**
LSTM	**0.99**	**0.98**	0.93	0.77
GRU	0.96	0.95	**0.95**	**0.96**
SRN	0.76	0.53	0.67	0.77

Table A.8
Generalization of recurrent models from training examples of previous levels of complexity to the next level of complexity absent from training data. Accuracy averaged over 10 runs. Random baseline is 0.25.

Test ex. complexity:	4	5	6	7
LSTM	**0.99**	0.95	0.8	0.7
GRU	0.96	**0.996**	**0.98**	**0.94**
SRN	0.76	0.4	0.38	0.33

Acknowledgments

The research was supported in part by CNRS PEPS ReSeRVe grant. I thank Germán Kruszewski, Marco Baroni, and anonymous reviewers for useful input.

References

Andreas, Jacob. 2019. Measuring compositionality in representation learning. *arXiv preprint arXiv:1902.07181*.

Armon-Lotem, Sharon. 1998. Mommy sock in a minimalist eye: On the acquisition of DP in Hebrew. *Issues in the Theory of Language Acquisition. Essays in Honor of Jürgen Weissenborn*. Bern: Peter Lang, pages 15–36.

Baroni, Marco. 2019. Linguistic generalization and compositionality in modern artificial neural networks. *Philosophical Transactions of the Royal Society B*. 375(1791):20190307. https://doi.org/10.1098/rstb.2019.0307, PubMed: 31840578

Bernardy, Jean-Philippe. 2018. Can recurrent neural networks learn nested recursion? *LiLT (Linguistic Issues in Language Technology)*, 16(1):1–20. https://doi.org/10.33011/lilt.v16i.1417

Cho, Kyunghyun, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*. https://doi.org/10.3115/v1/W14-4012

Christiansen, Morten H. and Nick Chater. 2016. The now-or-never bottleneck: A fundamental constraint on language. *Behavioral and Brain Sciences*, 39:1–72. https://doi.org/10.1017/S0140525X1500031X, PubMed: 25869618

Elman, Jeffrey L. 1991. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7(2–3):195–225. https://doi.org/10.1007/BF00114844

Ettinger, Allyson, Ahmed Elgohary, Colin Phillips, and Philip Resnik. 2018. Assessing composition in sentence vector representations. *arXiv preprint arXiv.1809.03992*.

Geva, Mor, Yoav Goldberg, and Jonathan Berant. 2019. Are we modeling the task or the annotator? An investigation of annotator bias in natural language understanding datasets. *arXiv preprint arXiv.1908.07898*. https://doi.org/10.18653/v1/D19-1107

Hao, Yiding, William Merrill, Dana Angluin, Robert Frank, Noah Amsel, Andrew Benz, and Simon Mendelsohn. 2018. Context-free transductions with neural stacks. *arXiv preprint arXiv:1809.02836*. https://doi.org/10.18653/v1/W18-5433

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. Long short-term memory, *Neural Computation*, 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735, PubMed: 9377276

Hupkes, Dieuwke, Verna Dankers, Mathijs Mul, and Elia Bruni. 2020. Compositionality decomposed: How do neural networks generalise? *Journal of Artificial Intelligence Research*, 67:757–795. https://doi.org/10.1613/jair.1.11674

Hupkes, Dieuwke, Anand Singh, Kris Korrel, Germán Kruszewski, and Elia Bruni. 2018. Learning compositionally through attentive guidance. *CoRR*, abs/1805.09657.

Hupkes, Dieuwke, Sara Veldhoen, and Willem Zuidema. 2018. Visualisation and 'diagnostic classifiers' reveal how recurrent and recursive neural networks process hierarchical structure. *Journal of Artificial Intelligence Research*, 61(1):907–926. https://doi.org/10.1613/jair.1.11196

Joulin, Armand, and Tomas Mikolov. 2015. Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in Neural Information Processing Systems*, pages 190–198.

Kim, Najoung and Tal Linzen. 2020. COGS: A compositional generalization challenge based on semantic interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9087–9105. https://doi.org/10.18653/v1/2020.emnlp-main.731

Kirov, Christo and Robert Frank. 2012. Processing of nested and cross-serial dependencies: An automaton perspective on SRN behaviour. *Connection Science*, 24(1):1–24. https://doi.org/10.1080/09540091.2011.641939

Lake, Brenden, and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, pages 2879–2888.

Lakretz, Yair, Dieuwke Hupkes, Alessandra Vergallito, Marco Marelli, Marco Baroni, and Stanislas Dehaene. 2021. Mechanisms for handling nested dependencies in neural-network language models and humans. *Cognition*, 213:1–24. https://doi.org/10.1016/j.cognition.2021.104699, PubMed: 33941375

Le, Phong and Willem Zuidema. 2015. The forest convolutional network: Compositional distributional semantics with a neural chart and without binarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1155–1164. https://doi.org/10.18653/v1/D15-1137

Liska, Adam, Germán Kruszewski, and Marco Baroni. 2018. Memorize or generalize? Searching for a compositional RNN in a haystack. *CoRR*, abs/1802.06467.

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

Mickus, Timothee, Timothée Bernard, and Denis Paperno. 2020. What meaning-form correlation has to compose with. *arXiv preprint arXiv:2012.03833*.

Mollica, Francis, Matthew Siegelman, Evgeniia Diachek, Steven T. Piantadosi, Zachary Mineroff, Richard Futrell, Hope Kean, Peng Qian, and Evelina Fedorenko. 2020. Composition is the core driver of the language-selective network. *Neurobiology of Language*, 1(1):104–134. https://doi.org/10.1162/nol_a_00005

Nangia, Nikita, and Samuel R. Bowman. 2018. ListOps: A diagnostic dataset for latent tree learning. *arXiv preprint arXiv:1804.06028*. https://doi.org/10.18653/v1/N18-4013

Ruis, Laura, Jacob Andreas, Marco Baroni, Diane Bouchacourt, and Brenden M. Lake. 2020. A benchmark for systematic generalization in grounded language understanding. In *Advances in Neural Information Processing Systems*, volume 33, pages 19861–19872, Curran Associates, Inc.

Soulos, Paul, Tom McCoy, Tal Linzen, and Paul Smolensky. 2019. Discovering the compositional structure of vector representations with role learning networks. *arXiv preprint arXiv:1910.09113*. https://doi.org/10.18653/v1/2020.blackboxnlp-1.23

Torregrossa, Jacopo and Chiara Melloni. 2014. English compounds in child Italian. In *New Directions in the Acquisition of Romance Languages, Selected Proceedings of the Romance Turn V*, pages 346–371, Cambridge Scholars Publishing.

Vaswani, Ashish, 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.

Weiss, Gail, Yoav Goldberg, and Eran Yahav. 2018. On the practical computational power of finite precision RNNs for language recognition. *arXiv preprint arXiv:1805.04908*. https://doi.org/10.18653/v1/P18-2117

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