Revisiting the poverty of the stimulus: hierarchical generalization without a hierarchical bias in recurrent neural networks

R. Thomas McCoy (tom.mccoy@jhu.edu)
Department of Cognitive Science, Johns Hopkins University

Robert Frank (bob.frank@yale.edu)
Department of Linguistics, Yale University

Tal Linzen (tal.linzen@jhu.edu)
Department of Cognitive Science, Johns Hopkins University

Abstract

Syntactic rules in human language usually refer to the hierarchical structure of sentences. However, the input during language acquisition can often be explained equally well with rules based on linear order. The fact that children consistently ignore these linear explanations to instead settle on hierarchical explanations has been used to argue for an innate hierarchical bias in humans. We revisit this argument by using recurrent neural networks (RNNs), which have no hierarchical bias, to simulate the acquisition of question formation, a hierarchical transformation, in an artificial language modeled after English. Even though this transformation could be explained with a linear rule, we find that some RNN architectures consistently learn the correct hierarchical rule instead. This finding suggests that hierarchical cues within the language are sufficient to induce a preference for hierarchical generalization. This conclusion is strengthened by the finding that adding an additional hierarchical cue, namely syntactic agreement, further improves performance.

Keywords: learning bias; poverty of the stimulus; recurrent neural networks

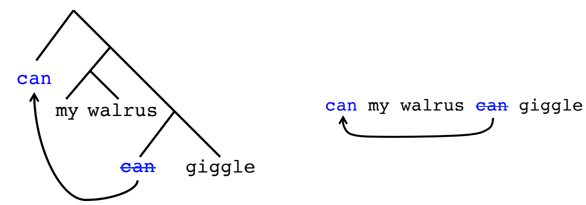
Introduction

Part of learning a language is learning to generalize from a finite number of examples in a way that is consistent with the rules of the language. Although there are many possible ways to generalize from a set of inputs, language learners consistently choose certain types of generalizations over others. This fact is particularly prominent in syntax, in which learners typically learn generalizations appealing to hierarchical structures rather than linear structures. One influential explanation for this consistency across speakers is that humans are born with a learning bias that favors hierarchy.

We use artificial neural networks—learning agents without an explicit bias toward hierarchy—to test whether a hierarchical bias is necessary to account for the ways that human language learners generalize. Specifically, we use such networks to simulate the acquisition of **subject-auxiliary inversion** in English, the syntactic transformation used to turn a declarative statement such as (1a) into a question such as (1b):

(1) a. My walrus **can** giggle.
b. **Can** my walrus giggle?

This transformation is relevant to questions of hierarchical bias because there are two (apparently equally plausible) rules that could generate (1b) from (1a)—a rule that appeals to hierarchical sentence structure and a rule that appeals to linear order.[1] These two rules are illustrated in Figure 6.



(a) **Hierarchical rule:** Move the main verb's auxiliary to the front of the sentence.

(b) **Linear rule:** Move the linearly first auxiliary to the front of the sentence.

Figure 1: Two possible question formation processes.

Though both of these rules correctly account for (1), they make different predictions about (2). The hierarchical rule predicts the correct question in (3a), while the linear rule predicts the incorrect question in (3b):

(2) My walrus that **will** eat **can** giggle.

(3) a. **Can** my walrus that **will** eat ___ giggle?
b. ***Will** my walrus that ___ eat **can** giggle?

Chomsky (1971) argues that, although examples such as this one would disambiguate the two hypotheses, many children never encounter any such examples during language acquisition and that, in the absence of these examples, they must instead be drawing on an innate preference for hierarchical structure to choose the hierarchical rule in Figure 1a over the linear rule in Figure 1b. This influential yet controversial argument is known as the **argument from the poverty of the stimulus** (Chomsky, 1980). A long debate has raged over this argument. Some have questioned the assumption that language learners do not encounter critical cases like the one in (3a) (Pullum & Scholz, 2002), but we focus on a different component of the argument: the assumption that an explicit hierarchical bias is necessary to reliably learn a hierarchical generalization instead of a linear one. Several authors have contested this point by using computational models to argue

[1] Other rules are also possible, such as moving the last auxiliary or moving the alphabetically first auxiliary, but here we only focus on these two.

☐ Training set, test set ☐ Generalization set

	IDENT	QUEST
No RC	*Input:* the newt can confuse my yak by the zebra . *Output:* the newt can confuse my yak by the zebra .	*Input:* the newt can confuse my yak by the zebra . *Output:* can the newt confuse my yak by the zebra ?
RC on object	*Input:* the newt can confuse my yak who will sleep . *Output:* the newt can confuse my yak who will sleep .	*Input:* the newt can confuse my yak who will sleep . *Output:* can the newt confuse my yak who will sleep ?
RC on subject	*Input:* the newt who will sleep can confuse my yak . *Output:* the newt who will sleep can confuse my yak .	*Input:* the newt who will sleep can confuse my yak . *Output:* can the newt who will sleep confuse my yak ?

Table 1: Examples for each combination of a sentence type and a task. RC stands for "relative clause."

for other possible explanations for the preference for hierarchy such as semantic considerations (Fitz & Chang, 2017), a Bayesian prior for shorter grammars (Perfors, Tenenbaum, & Regier, 2011), or the statistical distribution of structures in the observed data (Lewis & Elman, 2001).

Though Lewis and Elman (2001) studied subject-auxiliary inversion by modeling the grammaticality of questions, Frank and Mathis (2007) pointed out that it is better thought of as a task of transforming a declarative sentence into a question. Accordingly, Frank and Mathis (2007) used a type of network that they called a transformational network, based on the work of Botvinick and Plaut (2006) and similar to the sequence-to-sequence models that are popular today, to see if RNNs could learn to transform a declarative sentence into a question. The setup used by Frank and Mathis (2007) was promising, but their results were mixed and difficult to interpret. We therefore use their work as a jumping-off point in the hope that more powerful modern sequence-to-sequence networks might achieve clearer results.

Like Frank and Mathis (2007), we use recurrent neural networks to model the acquisition of subject-auxiliary inversion because they do not contain an explicit bias toward hierarchical structures. In fact, if anything, RNNs appear to have a bias toward linear structures over hierarchical ones (Christiansen & Chater, 1999).

Summary of results

We explore six RNN architectures and find that one of the architectures consistently learns a hierarchical generalization for question formation. Because this architecture does not have a an explicit hierarchical bias, this tendency to make hierarchical generalizations must have been induced by cues toward hierarchical structure in the network's input coupled with biases implicit in the network's computational architecture. We therefore take this finding as evidence that a learner's preference for hierarchy may arise from hierarchical properties of the input without the need for a specific hierarchical bias in the learner. Further evidence for this conclusion comes from an additional finding that adding the additional hierarchical cue of agreement increases the probability that a network will make hierarchical generalizations.

Though the existence of a single non-hierarchically-biased architecture that generalizes in a hierarchical way is sufficient to make these conclusions, the fact that the other five network architectures do not behave in this way is also interesting. Specifically, the qualitatively different behaviors of these architectures, some of which are often assumed to be interchangeable, offers intriguing possibilities for further work studying the learning capabilities of different architectures.

Experimental setup

The Languages

We train our networks on an artificial language consisting of English declarative sentences made with a 66-word vocabulary.[2] Each noun phrase argument of a transitive or intransitive verb can have at most one modifier (either a relative clause or a prepositional phrase). The language does not allow a relative clause to be embedded inside another relative clause. (4) gives some sentences from the language:

(4) a. the walrus can giggle .
b. the yak could amuse your quails by my raven .
c. the walruses that the newt will confuse can high_five your peacocks .

In this language, every verb is associate with one of the auxiliary verbs *can*, *could*, *will*, and *would*. Since such modals do not show agreement, any noun (whether singular or plural) can appear with any auxiliary. We also conduct experiments with a second language identical to the first except with the auxiliaries *do*, *don't*, *does*, and *doesn't*. In this language, subjects now must agree with the auxiliaries of their verbs: Singular subjects require *does* or *doesn't*, while plural subjects require *do* or *don't*. We call these languages the **no agreement language** and the **agreement language**. (5) gives some examples from the agreement language:

(5) a. the walrus does giggle .
b. the yak doesn't amuse your quails by my raven .
c. the walruses that the newt does confuse do high_five your peacocks .

Agreement depends on hierarchical structure; for example, in (5c), *do* agrees with its hierarchically-determined plural

[2]The full context-free grammar used to generate these sentences, along with information about the distributions of constituent and sentence types, can be found at `git.io/subj_aux_supplement`.

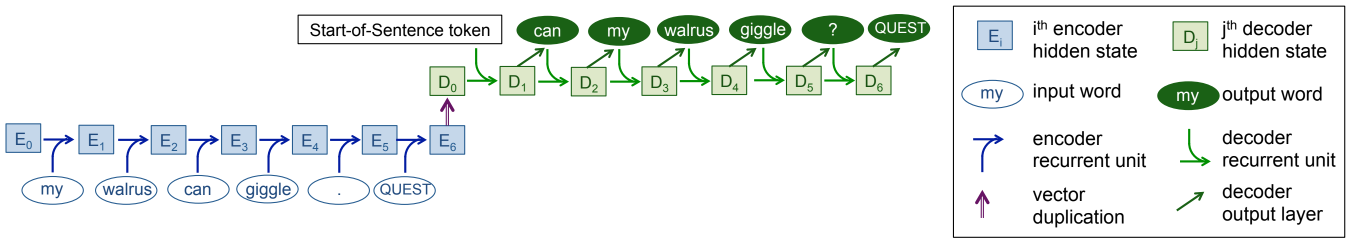


Figure 2: Basic sequence-to-sequence neural network without attention.

subject of *walruses* even though there is a singular noun (*yak*) linearly closer to it. We therefore use these two languages to test whether the presence of one hierarchically-based pattern (agreement) can influence how a learner handles a different hierarchically-based pattern (subject-auxiliary inversion). Note, though, that both languages, even the no-agreement one, contain other cues to hierarchy because they reuse structural units (e.g., by allowing prepositional phrases to modify subject or object nouns): shared structure is more efficiently represented in a hierarchical grammar than a linear one.

The Tasks

Given an input sentence, the networks must perform one of two tasks: identity (i.e., returning the exact sentence that was inputted) or question formation. The task to be performed is indicated by an additional token at the end of the sentence—either *IDENT* for identity or *QUEST* for question formation.

We can divide the inputs into three types: sentences without relative clauses, sentences with a relative clause on the object, and sentences with a relative clause on the subject. Table 1 gives examples for the six possible combinations of a task and a sentence type. During training, one of these six combinations—namely, the question formation task for a sentence with a relative clause on its subject (the shaded cell in Table 1)—is withheld. This is relevant because this is the only one of the six cases that would disambiguate the two hypotheses in Figure 6. Therefore, during training, the networks have no direct evidence for choosing between these hypotheses.

Evaluation

We use two sets of sentences for evaluation, a test set and a generalization set. The test set consists of novel sentences from the five non-withheld cases in Table 1. It is therefore used to assess how well a network has learned the patterns in its training set. The generalization set consists of sentences from the withheld case of the question formation task with sentences with relative clauses on their subjects. This set is therefore used to assess how the network's behavior generalizes to sentence types it has not seen before. The test and generalization set both contained 10,000 sentences, while the training set contained 120,000 sentences (all sentences were unique, both within and across sets).

The Architectures

For all experiments we used the sequence-to-sequence model (Botvinick & Plaut, 2006; Sutskever, Vinyals, & Le, 2014) illustrated in Figure 2.[3] This network has two subcomponents called the **encoder** and the **decoder**. First, the sequence of words from the input sentence is fed to the encoder one word at a time. The encoder maintains a vector called its hidden state, which is intended to represent the input sequence it has seen so far. As each word of the input sentence is received, the encoder updates the hidden state based on two inputs: (i) the previous hidden state and (ii) a distributed representation of the word that was just inputted. The matrix of weights that performs this update is called the encoder's **recurrent unit**.

Once the the entire sentence has been encoded, the decoding process begins. Like the encoder, the decoder maintains a hidden state. This hidden state is initialized with the final hidden state of the encoder, which acts as a vector representation of the input sentence. At each time step, the decoder's hidden state outputs one word. The decoder then uses its own recurrent unit to update its hidden state based on (i) the previous values in the hidden state and (ii) a distributed representation of the word just outputted. The sequence of words generated by the decoder concluding with a special end-of-sentence symbol is taken as the network's output.

We test three types of recurrent units for the encoder and decoder: a simple recurrent network (SRN) (Elman, 1990), a gated recurrent unit (GRU) (Cho et al., 2014), and long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997). The SRN uses a linear combination of its inputs to generate the next hidden state. For SRNs, the signal from a given input tends to rapidly diminish over time, making it difficult for the network to remember information. The design of GRUs and LSTMs addresses this problem by incorporating internal neural layers called gates as explicit mechanisms for remembering or forgetting information from previous states. The key difference between LSTMs and GRUs is in their gates: GRUs have fewer gates than LSTMs, and some single gates in GRUs combine the functions of two different LSTM gates.

For each type of recurrent unit, we experiment with adding an **attention mechanism** (Bahdanau, Cho, & Bengio, 2015), which is a modification to the decoder's recurrent unit. Atten-

[3] Here we mainly aim to give an intuitive description of how our networks operate. For a more precise description of the architecture, see the supplementary materials at `git.io/subj_aux_supplement`.

tion adds a third input, namely a weighted sum of the hidden states from each time step of encoding, to the decoder's recurrent unit. For each step of decoding, the weights used to take this weighted sum are derived from the previous hidden state and the previously outputted word.

For each pair of an architecture and a language, we trained 100 networks with different random initializations for a total of 1200 trained networks. This setup allowed the evaluation of each architecture in general, rather than evaluating just one network whose behavior may be sensitive to its initialization. Each network began with random initializations for the weights for its encoder and decoder recurrent units and the linear layers used to determine attention and the vector representations of words. For each network, these weights were trained using stochastic gradient descent for 30,000 batches with a batch size of 5, a dropout rate of 0.1, and a learning rate of 0.01 (for the GRUs and LSTMs) or 0.001 (for the SRNs). All networks used 256-dimensional hidden states and trained 256-dimensional vector representations of words. These parameter values were chosen through tuning aimed at achieving high accuracy on the test set.

Results

Test set

For the test set, all networks except the vanilla SRN (i.e., the SRN without attention) get over 94% of the output sentences exactly correct, where the accuracy is measured as an average across all 100 trained networks for each architecture. The highest accuracy is 99.9% for the LSTM without attention. Though the vanilla SRN does poorly at getting the output exactly correct, it gets 81% of sentences correct if it is not penalized for replacing a word with another word of the same part of speech, suggesting that its main source of error is a tendency to replace words with other words of the same lexical category. This tendency is a known deficiency of RNNs (Frank & Mathis, 2007) and does not bear on our main concern of the networks' syntactic representations. Therefore, setting aside these lexical concerns, we conclude that all architectures perform well at learning the training language.

Generalization set

On the generalization set, the average whole sentence accuracies are much lower, being only about 13% in the best-performing architecture (the LSTM with attention). However, getting the output exactly correct is a demanding metric; RNNs make many errors that would ruin these accuracies, such as repeating or omitting words or confusing similar words. Our motivation is not in getting perfect performance, however, but rather in testing whether the networks adopt the hierarchical or linear rule for question formation. Metrics that are sensitive to extraneous errors might mask conclusions about these hypotheses. To abstract away from such extraneous errors, for the generalization set we focus on accuracy at the first word of the output. Because all examples in the generalization set involve question formation, this word is al-

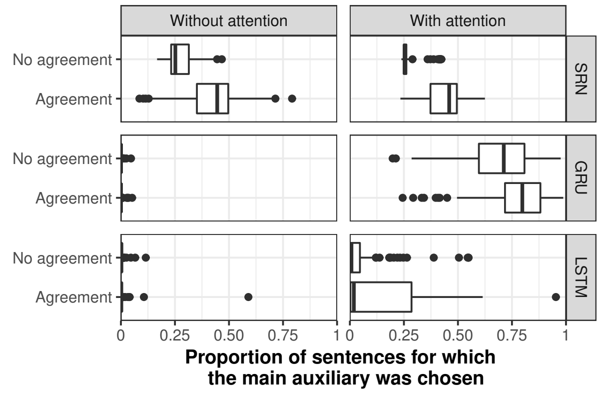


Figure 3: Accuracies at predicting the correct auxiliary to begin questions of the withheld type.

ways the auxiliary that is moved to form the question, and the identity of this auxiliary is enough to differentiate the hypotheses. For example, if the input is *my yak who the seal can amuse will giggle . QUEST*, a hierarchically-generalizing network would choose *will* as the first word of the output, while a linearly-generalizing network would choose *can*. Note that such an analysis only disambiguates the hypotheses if the two possible auxiliaries are different, so we only consider sentences where that is the case. For the agreement language, we make the further stipulation that both auxiliaries must agree with the subject so that the correct auxiliary cannot be determined based on agreement alone. Figure 3 gives the accuracies on this metric across the six architectures for the two different languages (individual points represent different initializations). We draw three conclusions from Figure 3:

Agreement leads to more hierarchical generalization behavior: All six architectures were significantly more likely ($p < 0.01$) to choose the main auxiliary when trained on the agreement language than when trained on the no-agreement language. This suggests that cues to hierarchical structure, such as agreement, can lead a learner to prefer a hierarchical representation without any changes to the learner.

Initialization matters: For each architecture, accuracy often varied considerably across random initializations. For example, though most LSTMs with attention had low accuracies, there was one with near-perfect accuracy. This observation highlights the importance of testing many initializations.

Different architectures perform qualitatively differently: Of the six architectures, only the GRU with attention showed a strong preference for choosing the main auxiliary instead of the linearly first auxiliary. By contrast, the vanilla GRU chose the linearly first auxiliary nearly 100% of the time. Thus, for the GRU, attention makes a qualitative difference for generalization: The GRU with attention tends to generalize hierarchically, while the vanilla GRU tends to generalize linearly.

By contrast, both LSTM architectures chose the linearly first auxiliary nearly 100% of the time. Both SRN architectures showed little preference between choosing the main auxiliary or the linearly first auxiliary; in fact the SRNs often chose an auxiliary that was not even in the input sentence, whereas the GRUs and LSTMs almost always chose one of the two auxiliaries in the input. In the next section, we take some preliminary steps toward exploring why the architectures behave in qualitatively different ways.

Analyses of Networks

One plausible explanation for the differences in generalization across architectures is that the representations learned by different architectures may focus on different aspects of the sentence. Specifically, linearly-generalizing networks might use representations encoding linearly-relevant information, while hierarchically-generalizing networks might use representations encoding hierarchically-relevant information.

To test this hypothesis, we analyzed the networks' encodings of input sentences to see what information they contained. These encodings (which are given by the final hidden state of the encoder, such as E_6 in Figure 2) are the decoder's only source of information about the input sentence (aside from the information gleaned from attention in networks with attention). We analyze the extent to which the encodings represent three properties of the input sentences: the main auxiliary, the fourth word, and the head noun of the subject (which, in the simple language we used, was always the sentence's second word). Examples are shown in Table 2.

Main auxiliary: A sentence's main auxiliary can appear in many different linear positions, but it has a consistent hierarchical position. Since hierarchical information is sufficient to identify the main auxiliary while linear information is not, a network whose encodings encode sentences' main auxiliaries must be encoding some hierarchical structure.

Fourth word: A sentence's fourth word has a well-defined place in the sentence's linear order, but it does not have any meaning within a hierarchical representation because it could have one of several hierarchical roles, including the main verb, the determiner on a prepositional object, or the auxiliary verb inside a subject relative clause. Therefore, a network whose encodings encode each sentence's fourth word must be encoding some information about linear structure.

Subject noun/second word: The head noun of the subject is interesting because, in our artificial language, it is always the sentence's second word. Thus, this word can be reliably identified either by linear information (as the second word) or hierarchical information (as the subject noun).

Analysis: For each trained network, we train three linear classifiers, one for each of these three parameters. Each classifier takes a sentence's 256-dimensional encoding as a set of 256 features for which the classifier learns weights to classify the sentence based on its main auxiliary, subject noun, or fourth word. These weights are trained on a training set and tested on a withheld test set. The test results are in Figure 4.

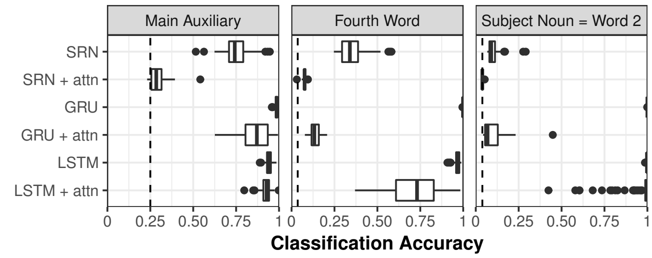


Figure 4: Linear classifier results. Dotted lines indicate chance baselines.

At the main auxiliary classification task, the SRNs with attention barely beat chance; this fact might partly explain why the SRNs with attention generalize poorly. All other architectures do well at this task. Since the identity of the main auxiliary is the only information required for our auxiliary prediction task, these results suggest that the differences in performance stem not from the inability to identify the main auxiliary but rather from a misinterpretation of the task as requiring fronting of the linearly first auxiliary.

Now consider the fourth word and subject noun classifications. Both LSTMs and the GRUs without attention perform well at both tasks, while the GRU with attention does poorly. The GRU with attention does perform well at classifying the main auxiliary, so overall the GRU with attention appears to use its encoding only for information that cannot be straightforwardly obtained from linear order, such as the main auxiliary, rather than information that could be obtained from linear order even if, like the subject head noun, that information is hierarchically relevant. On the other hand, the fact that the GRU without attention and both LSTM architectures perform very well at all three tasks suggests that they use their encodings for both linear and hierarchical information. This difference is interesting because the GRU with attention is the only architecture that shows a strong preference for choosing the main auxiliary. Thus, perhaps the better generalization ability of the GRU with attention arises not from a better ability to encode relevant hierarchical information (because all four architectures have that ability) but rather from an ability to ignore linear information, as Frank and Mathis (2007) argue.

Comparing RNN Mistakes with Human Mistakes

We now once again consider the full outputs of our networks, rather than just the first words, to compare the networks' errors with the types of errors that humans make when acquiring English as identified by Crain and Nakayama (1987). Here we only consider the outputs of the GRU with attention networks, as that was the only architecture that appeared to generally produce the correct auxiliary (see Figure 3).

We decompose subject-auxiliary inversion into two subtasks: placing an auxiliary at the start of the sentence and deleting an auxiliary from within the sentence. 65% of the outputs could have been formed by inserting an auxiliary before the sentence and deleting zero or one of the auxiliaries

Main auxiliary	Fourth word	Subject noun
my unicorns **would** laugh .	my unicorns would **laugh** .	my **unicorns** would laugh .
my quail with her yak **will** read .	my quail with **her** yak will read .	my **quail** with her yak will read .
his newt who can giggle **could** swim .	his newt who **can** giggle could swim .	his **newt** who can giggle could swim .

Table 2: Examples of the entities identified by the linear classifiers.

	Prepose 1st	Prepose 2nd	Prepose other
Delete 1st	0.07	0.24	0.04
Delete 2nd	0.00	0.03	0.00
Delete none	0.04	0.21	0.02

Table 3: Analysis of output question types based on which auxiliary has been deleted (if any) and which auxiliary has been placed at the start of the sentence. Each number is the percent of GRU + attention outputs that fit that category. *1st* and *2nd* refer to the first and second auxiliaries in the input.

in the sentence. Table 3 breaks down those results based on which auxiliary was preposed and which (if any) was deleted.

Two errors are by far the most commmon. One of them (delete none/prepose second – e.g. generating *could his newt who can giggle could swim ?* from *his newt who can giggle could swim .*) is one which Crain and Nakayama (1987) found to be common among English-learning children and which is compatible with hierarchical generalization. The other frequent error (delete first, prepose second – e.g. generating *could his newt who giggle could swim ?* from *his newt who can giggle could swim .*) is never observed by Crain and Nakayama (1987) and is incompatible with a hierarchical generalization. Thus, though these networks do in some ways behave like humans, in other ways they make mistakes that humans never would.

Conclusions and Future Work

We found that GRUs with attention consistently learn a hierarchical generalization about question formation despite having no explicit hierarchical bias. This finding suggests that hierarchical cues arising from distributional properties of the input are sufficient to lead a learner of the appropriate sort to posit hierarchical representations not only for sentences, but also for operations that transform them. This conclusion is supported by the fact that the additional hierarchical cue of agreement increases the likelihood a network will induce hierarchical generalizations.

In future work we will further probe the architectures we tested to better understand why they behave in such qualitatively different ways. Understanding these differences may lend insight into how the abstract computational properties of a learner relate to apparently language-specific questions of hierarchy vs. linearity. We also anticipate replacing our artificial training language with a corpus of child-directed speech to see if our findings extend to naturalistic linguistic input.

Acknowledgments

Our experiments were conducted using the resources of the Maryland Advanced Research Computing Center (MARCC). We thank Joe Pater, Paul Smolensky, and the JHU Computational Psycholinguistics group for helpful comments.

References

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR.*

Botvinick, M. M., & Plaut, D. C. (2006). Short-term memory for serial order: A recurrent neural network model. *Psychological Review*, *113*(2), 201–233.

Cho, K., Van Merrinboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of EMNLP.*

Chomsky, N. (1971). *Problems of knowledge and freedom.* New York: Pantheon.

Chomsky, N. (1980). Rules and representations. *Behavioral and Brain Sciences*, *3*(1), 1–15.

Christiansen, M. H., & Chater, N. (1999). Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, *23*(2), 157–205.

Crain, S., & Nakayama, M. (1987). Structure dependence in grammar formation. *Language*, 522–543.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, *14*(2), 179–211.

Fitz, H., & Chang, F. (2017). Meaningful questions: The acquisition of auxiliary inversion in a connectionist model of sentence production. *Cognition*, *166*, 225–250.

Frank, R., & Mathis, D. (2007). Transformational networks. In *Proceedings of the Workshop on Psychocomputational Models of Human Language Acquisition.*

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Lewis, J. D., & Elman, J. L. (2001). Learnability and the statistical structure of language: Poverty of stimulus arguments revisited. In *Proceedings of BUCLD.*

Perfors, A., Tenenbaum, J. B., & Regier, T. (2011). The learnability of abstract syntactic principles. *Cognition*, *118*(3), 306–338.

Pullum, G. K., & Scholz, B. C. (2002). Empirical assessment of stimulus poverty arguments. *The Linguistic Review*, *18*(1-2), 9–50.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Proceedings of NIPS.*

Supplementary Materials

Details of the Grammar

S → NP VP .
NP → Det N
NP → Det N PP
NP → Det N RC
VP → Aux V_intrans
VP → Aux V_trans NP
PP → P Det N
RC → Rel Aux V_intrans
RC → Rel Det N Aux V_intrans
RC → Rel Aux V_trans Det N
Det → {the | some | my | your | our | her}
N → {newt | newts | orangutan | orangutans | peacock | peacocks | quail | quails | raven | ravens | salamander | salamanders | tyrannosaurus | tyrannosauruses | unicorn | unicorns | vulture | vultures | walrus | walruses | xylophone | xylophones | yak | yaks | zebra | zebras }
V_intrans → {giggle | smile | sleep | swim | wait | move | change | read | eat }
V_trans → {entertain | amuse | high_five | applaud | confuse | admire | accept | remember | comfort }
Aux → {can | will | could | would }
P → {around | near | with | upon | by | behind | above | below }
Rel → {who | that }

Figure 5: Context-free grammar for the no agreement language. The grammar contains 6 determiners (*Det*), 26 nouns (*N*), 9 intransitive verbs (*V_intrans*), 9 transitive verbs (*V_trans*), 4 auxiliaries (*Aux*), 8 prepositions (*P*), and 2 relativizers (*Rel*).

Figure 5 contains the context-free grammar used to generate the no agreement language. 120,000 sentences were generated from this grammar as the training set, with each example randomly assigned either the identity task or the question formation task. If a sentence was assigned the question formation task and contained a relative clause on the subject, it was not included in the training set.

The agreement language was generated from a similar grammar but with the auxiliaries changed to *do*, *does*, *don't*, and *doesn't*. In addition, to ensure proper agreement, the grammar for the agreement language had separate rules for sentences with singular subjects and sentences with plural subjects, as well as separate rules for relative clauses with singular subjects and relative clauses with plural subjects.

Figure 6a shows how frequent each sentence type was based on the types of modifiers present in the sentence and which noun phrases those modifiers were modifying. Figure 6b shows the same statistics for the agreement language. In general, for a given left hand side in the grammar in Figure 5, all rules with that left hand side were equally probable; so, for example, one third of noun phrases were unmodified, one third were modified by a prepositional phrase, and one third were modified by a relative clause. The one exception to this generalization is that intransitive sentences with unmodified subjects are rare in both languages. This is because we did not allow any repeated items within or across data sets, and since there are relatively few possible intransitive sentences with unmodified subjects, this uniqueness constraint prevented the unmodified intransitive case from being as common as the modified cases. The no agreement language has roughly twice as many intransitive sentences with unmodified subjects as the agreement language does because there are twice as many possible sentences of that type for the no agreement language than the agreement language, but otherwise the two languages are essentially the same in the distributions of their constructions.

Details of the Architecture

Figure 2 (reproduced here as Figure 7) depicts the basic sequence-to-sequence architecture underlying all of our experiments. Here we elaborate on the different components of this architecture.

The network consists of two components, the encoder and the decoder. The encoder's hidden state is initialized at E_0 as a 256-dimensional vector of all zeroes. The network is then fed the first word of the input sentence, represented in a distributed manner as a 256-dimensional vector whose elements are learned during training. This distributed representation of the first word, along with the initial hidden state, is fed through the encoder's recurrent unit, and the 256-dimensional output becomes the next hidden state of the encoder, E_1. Each subsequent word of the input sentence is then fed into the network, turned into its distributed representation learned by the network, and passed through the hidden state along with the previous hidden state to generate the next hidden state.

Once all of the input words have been passed through the encoder, the final hidden state of the encoder is used as the initial hidden state of the decoder, D_0. This hidden state and a special start-of-sentence token (represented by a 256-dimensional distributed representation that is learned during training) are passed as inputs to the decoder's recurrent unit, which outputs a new 256-dimensional vector as the next decoder hidden state, D_1. A copy of this new hidden state is also passed through a linear layer whose output is a vector with a magnitude equal to the vocabulary size. The softmax function is then applied to this vector (so that its values sum to 1 and all fall between 0 and 1). Then, the element of this vector with the highest value is taken to correspond to the output word for that timestep; this correspondence is determined by a dictionary relating each index in the vector to a word in the vocabulary. For the next time step of decoding, this just-outputted word is converted to a distributed representa-

	Identity	Question formation
Intransitive:		
No modifiers	0.012	0.012
PP on subject	0.122	0.125
RC on subject	0.121	0.000
Transitive:		
No modifiers	0.040	0.040
PP on subject	0.041	0.041
PP on object	0.041	0.041
RC on subject	0.041	0.000
RC on object	0.041	0.041
PP on subject, PP on object	0.040	0.040
PP on subject, RC on object	0.041	0.041
RC on subject, PP on object	0.041	0.000
RC on subject, RC on object	0.040	0.000

(a) No agreement language

	Identity	Question formation
Intransitive:		
No modifiers	0.005	0.005
PP on subject	0.125	0.123
RC on subject	0.120	0.000
Transitive:		
No modifiers	0.040	0.041
PP on subject	0.042	0.041
PP on object	0.042	0.041
RC on subject	0.042	0.000
RC on object	0.042	0.042
PP on subject, PP on object	0.041	0.042
PP on subject, RC on object	0.042	0.041
RC on subject, PP on object	0.042	0.000
RC on subject, RC on object	0.042	0.000

(b) Agreement language

Figure 6: Frequencies of sentence types in the two training sets. PP stands for *prepositional phrase* and RC stands for *relative clause*. Each line lists all modifiers in the sentences in question, so for example *PP on object* excludes cases where there is also a modifier on the subject.

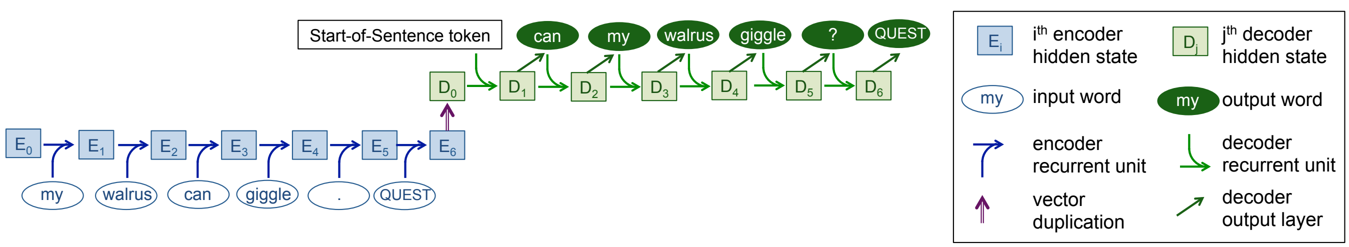


Figure 7: Basic sequence-to-sequence neural network without attention.

tion and is then taken as an input to the decoder's recurrent unit, along with the previous decoder hidden state, to generate the next decoder hidden state and the next output word. Once the outputted word is an end-of-sequence token (either *IDENT* or *QUEST*), decoding stops and the sequence of outputted words is taken as the output sentence. At all steps of this decoding process, whenever a distributed representation is used, dropout with a dropout proportion of 0.1 is applied to the vector, meaning that each of its values will with 10% probability be turned to 0. This practice is meant to combat overfitting of the network's parameters.

There are two main dimensions of variation that we use for this basic architecture. First is the attention mechanism, depicted in Figure 8, which is a modification to the decoder's recurrent unit. The attention mechanism adds a third input (which we refer to as the attention-weighted sum) to the decoder recurrent unit. This attention-weighted sum is determined as follows: First, the distributed representation of the previous output word and the previous hidden state are passed through a linear layer whose output is a vector of length equal to the number of words in the input. This vector is the vector of attention weights. Each of these weights is then multiplied by the hidden state of the encoder at the encoding time step equal to that weight's index. All of these products are then added together to give the attention-weighted sum, which is passed as an input to the decoder recurrent unit along with the previous output word and the previous hidden state.

Second, we also vary the structure of the recurrent unit used for the encoder and decoder. The three types of recurrent units we experiment with are simple recurrent networks (SRNs) (Elman, 1990), gated recurrent units (GRUs) (Cho et al., 2014), and long short-term memory (LSTM) units (Hochreiter & Schmidhuber, 1997)[4].

The SRN concatenates its inputs, passes the result of the concatenation through a linear layer whose output consists of linear combinations of the elements of the input vector, and finally applies the hyperbolic tangent function to the result to create a vector whose values are mostly either very close to -1 or very close to +1. This hidden state update can be expressed with the following equation:

[4]All of these architectures have multiple variants. We used the default PyTorch implementations of all relevant architectures.

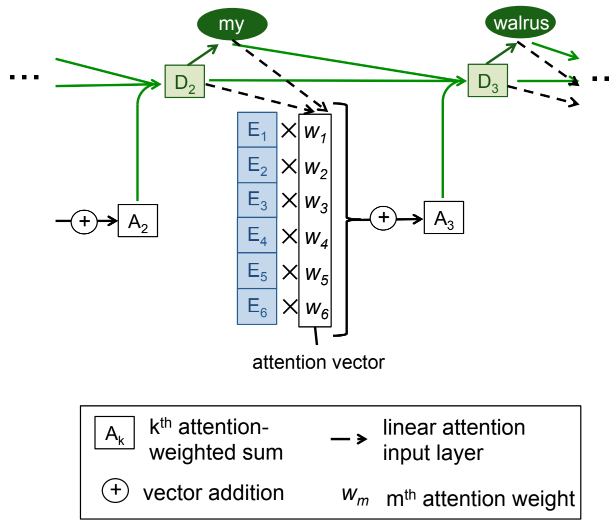


Figure 8: The attention mechanism.

$$D_i = tanh(W * [D_{i-1}, w_{i-1}] + b) \tag{1}$$

where D_i is the ith hidden state of the decoder, w_i indicates the ith output word, W is a matrix of learned weights, b is a learned vector called the bias term, and $[v_1, v_2, ...]$ indicates the concatenation of vectors v_1, v_2, ... If attention is used, this equation then becomes

$$D_i = tanh(W * [D_{i-1}, w_{i-1}, A_i] + b) \tag{2}$$

where A_i is the ith attention-weighted sum.

The GRU adds several internal vectors called gates to the basic SRN structure. Specifically, these gates are called the reset gate r_t, the input gate z_t, and the new gate n_t, each of which has a corresponding matrix of weights (W_r for r_t, W_z for z_t, and two separate matrices W_{nw} and W_{nD} for n_t). The reset and input gates both take the previous hidden state and the previously outputted word (as a distributed representation) as inputs. The new gate also takes these two inputs as well as the reset gate as a third input. The next hidden state is then generated as the product of the input gate and the previous hidden state plus the product of one minus the input gate times the new gate. Intuitively, this can be thought of as the input gate determining which elements of the hidden state to preserve and which to change. The elements to be preserved are preserved through the term that is the product of the input gate times the previous hidden state, while the elements to be changed are determined through the term that is the product of one minus the input gate times the new gate; the new gate here determines what the updated values for these changed terms should be. Overall the GRU update can be expressed with the following equations (σ indicates the sigmoid function):

$$r_t = \sigma(W_r * [D_{t-1}, w_{t-1}] + b_r) \tag{3}$$

$$z_t = \sigma(W_z * [D_{t-1}, w_{t-1}] + b_z) \tag{4}$$

$$n_t = tanh(W_n w * w_{t-1} + b_n w + r_t * W_n D * (D_{t-1} + b_n D)) \tag{5}$$

$$h_t = z_t * D_{t-1} + (1 - z_t) * n_t \tag{6}$$

Like the GRU, the LSTM also uses gates—specifically, the input gate i_t, forget gate f_t, cell gate g_t, and output gate o_t. Furthermore, while the other architectures all just use the hidden states as the memory of the network, the LSTM adds a second vector called the cell state c_t that acts as another persistent state that is passed from time step to time step. These components interact according to the following equations to produce the next hidden state and cell state:

$$i_t = \sigma(W_i * [D_{t-1}, w_{t-1}] + b_i) \tag{7}$$

$$f_t = \sigma(W_f * [D_{t-1}, w_{t-1}] + b_f) \tag{8}$$

$$g_t = tanh(W_g * [D_{t-1}, w_{t-1}] + b_g) \tag{9}$$

$$o_t = \sigma(W_o * [D_{t-1}, w_{t-1}] + b_o) \tag{10}$$

$$c_t = f_t * c_{t-1} + i_t * g_t \tag{11}$$

$$h_t = o_t * tanh(c_t) \tag{12}$$

Details of the Linear Classifiers

Each linear classifier consisted of a single linear layer which took as its input a 256-dimensional vector (specifically, the encoding of a sentence) and outputted a vector of dimension equal to the number of possible values for the feature used as the basis of classification. For example, since there are four auxiliaries, the main auxiliary classifier had an output of dimensionality 4. Each element in this output corresponded to a specific value for the feature being used as the basis for classification, and for a given input the element of the output with the highest value was taken as the classification for that input. The weights of the classifier were trained using stochastic gradient descent.