Supplementary Materials

Details of the Grammar

S $\rightarrow$ NP VP .
NP $\rightarrow$ Det N
NP $\rightarrow$ Det N PP
NP $\rightarrow$ Det N RC
VP $\rightarrow$ Aux V_intrans
VP $\rightarrow$ Aux V_trans NP
PP $\rightarrow$ P Det N
RC $\rightarrow$ Rel Aux V_intrans
RC $\rightarrow$ Rel Det N Aux V_intrans
RC $\rightarrow$ Rel Aux V_trans Det N
Det $\rightarrow$ {the | some | my | your | our | her}
N $\rightarrow$ {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 $\rightarrow$ {giggle | smile | sleep | swim | wait | move | change | read | eat }
V_trans $\rightarrow$ {entertain | amuse | high_five | applaud | confuse | admire | accept | remember | comfort }
Aux $\rightarrow$ {can | will | could | would }
P $\rightarrow$ {around | near | with | upon | by | behind | above | below }
Rel $\rightarrow$ {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 (i.e. an embedding) 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 length 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,

	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.

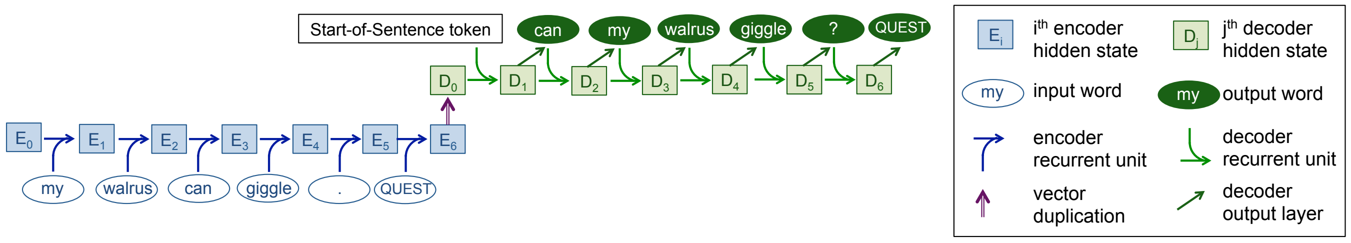

Figure 7: Basic sequence-to-sequence neural network without attention.

this just-outputted word is converted to a distributed representation 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]. For all three of these types of recurrent units, we use the default PyTorch implementations, which are described in the next few paragraphs.

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

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

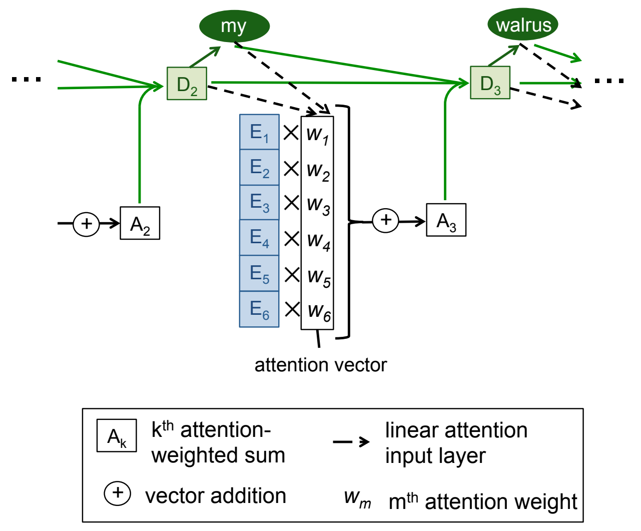

Figure 8: The attention mechanism.

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:

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

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:

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

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

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

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

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

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

For all architectures, the network was trained using stochastic gradient descent for 30,000 batches, each with a batch size of 5.

Test and Generalization Accuracies

Table 4 gives the accuracies of various architectures on the test set and generalization set.

	Test set		Generalization set	
	Word match	POS match	Word match	POS match
SRN	0.001	0.811	0.000	0.000
SRN + attn	0.942	0.999	0.010	0.023
GRU	0.983	0.998	0.008	0.022
GRU + attn	0.975	0.993	0.033	0.041
LSTM	0.999	1.000	0.046	0.069
LSTM + attn	0.998	1.000	0.133	0.185

Table 4: Accuracy of each architecture with the agreement language. *Word match* means getting the output exactly correct, while *POS match* allows words to be replaced by other words of the same part of speech (POS). Each number in this table is an average across 100 networks.

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 sentence encodings were split into a training set (75% of the encodings), a development test set (5% of the encodings), and a test set (20% of the encodings), none of which overlapped. The weights of the classifier were trained on the training set using stochastic gradient descent, and training stopped when the cross entropy loss computed over the development test set ceased to improve. Classification accuracy was then determined based on the withheld test set.