The Challenge of Ending Sequences when Extrapolating

Benjamin Newman **Sang Michael Xie** **Percy Liang**

1 Introduction

Recently, improvements in NLP models' abilities to perform machine translation, answer questions, and conduct conversations have been fueled by the adaptation of deep learning techniques. Despite successes, these models still struggle to generalize in the same ways humans do when we use language. For example, if a person understands what it means to "skip and jump", as well as what it means to perform an action twice, it is likely that they would know how to "skip twice and jump twice" even if they have never explicitly been instructed how to do so (Chomsky, 1957; Frege, 1953; Montague, 1970). NLP sequence models still find this kind of extrapolation challenging—they have trouble generating sequences longer than the ones they were trained on, even if they have been trained with all of the individual components required to generate the correct sequences (Lake and Baroni, 2018). This is the setting we are interested in studying in this work: improving the ability of sequence models to extrapolate to longer sequences.

Some recent work shows evidence that the common practice of including a special end of sequence (EOS) token in the model vocabulary prevents recurrent sequence models (RNNs and Transformers) from extrapolating (Newman et al., 2020). We can see the effect of this practice by breaking down extrapolation into two components. In order to extrapolate a model needs to have:

1. knowledge of how what content to generate for longer sequences (i.e. what tokens to predict).

2. knowledge of when to end sequences.

Including an EOS token at the end of training sequences address both of these components by recasting (2) as (1). This methodology has been used since the 1990's and has provided a useful framework for addressing ending sequences in settings that do not require extrapolation (Collins, 1999). Unfortunately, previous work shows that using EOS tokens leads to poor performance in the extrapolation setting. Dubois et al. (2020) observe that models tend to predict EOS too early in an extrapolation setting—failing at (2). Newman et al. (2020) further this observation, noting that using EOS tokens also hurts models ability to perform (1). They train models without EOS tokens and notice that these are able to produce the correct content (able to perform (1)) whereas models trained with EOS tokens, but prevented from producing them at evaluation time, are unable to perform (1). This is likely because it is easier for RNNs and Transformers to generate recurring patterns rather than predict where sequences end. However, Newman et al. (2020) do not provide an alternative method for ending sequences, leaving models still unable to perform (2). This is a problem because in many generative NLP tasks, such as machine translation, abstractive summarization, or even language modeling, models need to know when to stop generation. In particular, for language models the probability of ending sequences is necessary to define a distribution over strings (as opposed to prefixes of strings).

Our work seeks to fill this gap—we take on the question of predicting the length of sequences when models are extrapolating. Our work focuses on the SCAN dataset: a synthetic sequence-to-sequence dataset meant to test the ability of models to perform the type of extrapolation we are interested in. The dataset consists of input-output pairs where inputs are instructions (e.g. "run and jump") and outputs are sequences of actions corresponding to those instructions (e.g. "RUN JUMP").

Building directly off the work of Newman et al. (2020), we assume that we have models that can generate the correct content when extrapolating, and then try to build models to predict the length of output sequences from the SCAN inputs and the outputs of these models. We reason that it should be easier to predict the length of sequences rather than produce the sequences themselves. We design two types of models: the first learns to regress from the input to the length of the action sequence and the second couples a sequence-to-sequence model with a binary classifier to predict whether each output token is the final one in the sequence. These approaches are ultimately unsuccessful, so we investigate additional ways we to add structure to our models. We settle on using auxiliary supervision , asking models to "show their work" when predicting the lengths of sequences by producing mathematical expressions that evaluate to the sequence length. This auxiliary supervision adds a useful inductive bias that helps our models achieve perfect extrapolation accuracy. We extend this method to address the whole SCAN extrapolation task by having models generate programs, that when run output the SCAN output sequences, achieving 100% extrapolation accuracy in this setting as well. We conclude our analysis by investigating the minimum number of examples that require this additional annotation expression in order to learn how to predict sequence lengths, determining that we can achieve 55% extrapolation accuracy with only 10% of the training data, with room for better performance.

Our findings suggest that ending sequences without adding some additional inductive bias is difficult, and adding a relatively small amount of additional supervision is likely enough to complete the task.

2 Related Work

Length Extrapolation and Sequence Models. Despite their successes, neural sequences models struggle to generate sequences longer than the ones that they have seen at training time (Dubois et al., 2020; Ruis et al., 2020; Hupkes et al., 2020; Klinger et al., 2020). There are a number of reasons why this might be the case, but a prominent belief is that extrapolation requires understanding the global structure of a task, which standard neural architectures have trouble emulating from a modest number of in-domain samples (Mitchell et al., 2018; Marcus, 2018; Gordon et al., 2019). The response often proposed to this critique is to build models that with different inductive biases: for example, different architectures or losses.

At the level of training procedures, previous work has noticed that the EOS tokens appended to the end of training samples negatively impact the ability of models to extrapolate. Dubois et al. (2020) observe that EOS tokens lead to models predicting that sequences should end too early in extrapolation settings. Additionally, Newman et al. (2020) find that including EOS tokens in the training data prevent models from even generating the correct content tokens. Removing them leads to better extrapolative performance at the cost of not being able to end sequences.

The difficult nature of extrapolation is evident in natural language tasks like machine translation as well as synthetic tasks like SCAN, the focus of our work (Murray and Chiang, 2018; Lake and Baroni, 2018).

Length Extrapolation and SCAN. Much of the previous work on extrapolation in sequence models comes from the SCAN task: a task that requires mapping synthetic instruction sequences to sequences of actions (we elaborate more on the composition of the dataset used for this task in Section 3.1). The reason for this focus is because there is a split of the dataset meant to test the ability of models to length extrapolate. Most prior work does not perform well on this split, with accuracies capped at around 20% (Gordon et al., 2019; Lake, 2019; Lake and Baroni, 2018). However, there is a line of recent work that has achieved close to perfect performance. Nye et al. (2020) propose an effective neural program synthesis technique. They train a sequence-to-sequence model to output programs that generate SCAN samples when run, and pretrain this model on samples from grammars that are similar to the one that generates SCAN examples. They use this model in a meta-learning framework to solve the length extrapolation task from a few (short) examples of the SCAN dataset.

Another approach that has recently garnered some success involves using architectures with both symbolic and neural components. Liu et al. (2020) propose a memory-augmented network trained with hierarchical reinforcement learning algorithm that is able to extrapolate. Chen et al. (2020) propose a neuro-symbolic architecture, with the neural model trained generate programs, which they refer to as "execution traces", which are interpreted by a symbolic component. This is similar to (Nye et al., 2020), but does not feature the pretraining or few-shot adaptation to the SCAN task. In our work, we focus on adding additional inductive bias through additional supervision rather than architecture changes or pretraining.

While these approaches have had success, it is not clear if gains on SCAN extrapolation necessarily transfer to natural language tasks (Shaw et al., 2020). That said, we focus on the SCAN task here, as it is still a useful test-bed for prototyping extrapolation methods.

3 Background

3.1 Dataset

In this work, we focus on the SCAN task (Lake and Baroni, 2018). SCAN is a synthetic sequence-to-sequence task meant to simulate data that would be used to train an instruction-following robot. Inputs are commands and outputs are sequences of actions.

Input commands consist of combinations of three types of tokens. Primitive actions (walk, run, look, jump, turn) correspond to a single action a robot would take. Modifiers (left, right, around, opposite, twice, thrice) change the primitive actions in their scope, by adding a direction or repeating them a certain number of times. Commands can have between zero and three modifiers. Conjunctions (and, after) combine two commands.

For example, the command "jump around left twice and run" corresponds to the sequence of actions: "JUMP LEFT JUMP LEFT JUMP LEFT JUMP LEFT JUMP LEFT JUMP LEFT JUMP LEFT JUMP LEFT RUN". ("JUMP LEFT" 8 times, followed by "RUN").

Because we are interested in the length-extrapolation setting, we use a predefined train-test split of the SCAN dataset meant to evaluate the ability of models to length-extrapolate. The training data in this split consists of outputs with 1–22 actions, and the testing data consists of outputs with 24–48 actions. The dataset is designed to test *compositional generalization*: all of the primitive actions, modifiers, and conjunctions as well as their role in creating the final output action sequences are seen during training, so it is possible for a model to learn the correct set of rules, and extrapolate perfectly.

3.2 Task

In this work, our goal is predicting the length of a SCAN action sequence. We are building off the investigations of Newman et al. (2020), so in addition to the SCAN dataset, we also have a sequence-to-sequence model (trained without EOS tokens), that is able to generate sequences whose prefixes are the correct action sequence with high accuracy (60%).

We can formalize our task as follows: Given a SCAN input command, action sequence pair (x, y) where the length of y is ℓ, and a model f trained to map x to y, we want to learn a model g such that $g(x, f) = \ell$. The way we see such a model being used is as follows: First, we compute $f(x)$ to generate a potentially unboundedly long sequence (in practice, we can choose a maximum length greater than all possible sequence lengths). Then we calculate our estimate of ℓ, $\ell' = g(x, f)$. Finally we return $f(x)_{1:\ell'}$ as our output to compare to y, where $f(x)_{1:\ell'}$ represents the first ℓ' tokens of $f(x)$. Our task is then to learn an accurate $g(x, f)$. Because the role of g is to determine where to end sequences, we refer to g as a *truncation model*.

Our oracle in this setting is a model g that knows ℓ, so it can calculate $f(x)$ and end the sequence at the correct position. As noted earlier, this method obtains an accuracy of 60%.

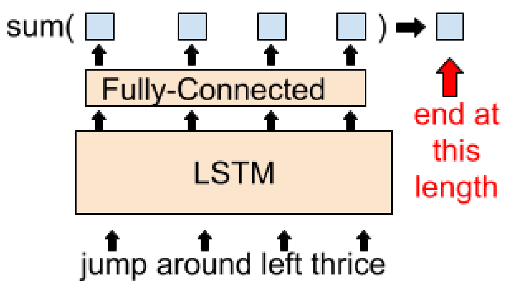
4 Experiment 1: No additional inductive bias

We first consider learning a truncation model from only the SCAN $(x, f(x))$ pairs. Our approaches fall into two categories: cut models and fertility models.

4.1 Methods

Cut models Cut models work by making a prediction at each position in $f(x)$ as to whether that sequence can end at that position. We implement these as seq2seq models, where the input sequence is the input command, and the decoder is fed $f(x)$ at evaluation time, outputting a sequence of binary classifications (Figure 1). The first position where the truncation model predicts the sequence can end is taken as the ending position. To implement this, we used an LSTM seq2seq model (2 layer encoder, 1 layer decoder with 200-dimensional hidden states).

Fertility models Fertility models, on the other hand, aim to learn how each input token contributes to the overall length of the output sequence. In this way, their role is similar to that of fertility models from statistical machine translation (Della Pietra et al., 1997). Because these models only depend on



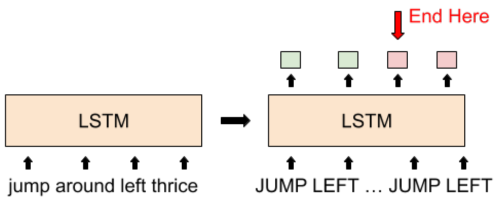


Figure 1: Architectures for the fertility (above) and cut (below) models.

the input sequences, they ignore the model f. This method should be effective if it is easier to learn how to predict the length of the output sequences when output tokens do not have to be generated. These models are implemented with recurrent models that predict a real value for each input token. The sum of these predictions is the predicted sequence length (Figure 1). We run a two-layer bidirectional LSTM encoder over input and regress each hidden state to an one-dimensional tensor and sum the outputs together. The model is only supervised with mean-squared error between the true and predicted lengths, there is no supervision on the individual tokens.

4.2 Results

We find that neither of these methods works particularly well. Both revert back to accuracies of less than 20% when evaluated on sequences of out-of-domain lengths (again out-of-domain lengths are 24–48 tokens rather than 1–22 tokens). This was approximately the same as a model trained with EOS tokens, with the fertility model performing a bit better. Investigating the mistakes that these models made reveals that in all case, the models predicted that sequences were shorter than they should be.

In an attempt to improve the performance of the fertility model, we ensemble it with a cut model. We do this because one of the potential detriments of the fertility model is that it chooses to end the sequence at an invalid location (e.g. between a

Model Type	ID	OOD
Cut	1.0	0.11
Fertility	0.97	0.17
Ensemble	0.62	0.19

Table 1: Accuracies without using additional supervision. ID is in-domain, OOD is out-of-domain, extrapolation lengths.

'JUMP' and a 'LEFT' in the earlier example). We assume that a cut model would not have this problem because the cut model would take into account the tokens in the output sequence when making its length prediction. To create this ensemble, we train a cut model that is not conditioned on the input sequence, and we have it and the fertility model predict where the sequence can end. The first position that the cut model predicts the sequence can end *after* the fertility model's predicted length is returned as the predicted length. This also does not work—dropping in-domain accuracy while keeping extrapolation accuracy low (Table 1). The in-domain accuracy drops because cut model's predictions for some examples is greater than the true length—the opposite of the earlier problem.

4.3 Toy Arithmetic Task

We investigate the failure of these models using an arithmetic task that captures the components necessary to predict the length of action sequences in the SCAN task—nested multiplication and addition. Inputs are sequences of digits as well as addition and multiplication signs and the outputs are what these sequences evaluate to. This is a good approximation for the task of predicting the length of SCAN sequences. Tokens like "left" and "walk" correspond to adding ones while tokens like "twice" and "around" correspond to multiplying by two and four respectively. We vary the level of difficulty of this task by training on inputs with various nesting levels of addition and multiplication.

For example, input output pairs would look like:

$$\text{one one one two one one} \rightarrow 7 \qquad (1)$$

$$\text{one twice two twice} \rightarrow 6 \qquad (2)$$

$$\text{one twice-1 two thrice-1 twice-2} \rightarrow 14 \qquad (3)$$

Where (1) has no nesting (i.e. multiplication), (2) has one level of nesting and (3) has two. For example, the output in (3) would be calculated as: $(1 \times 2) + (2 \times 3 \times 2) = 2 + 12 = 14$. When we

ND	MM	architecture	ID	OOD
0	-	fertility	1.0	1.0
0	-	cut	1.0	1.0
1	2	fertility	1.0	1.0
1	2	cut	1.0	0.98
1	3	fertility	1.0	1.0
1	3	cut	1.0	0.88
2	2	fertility	1.0	1.0
2	2	cut	1.0	0.85
2	3	fertility	1.0	0.80
2	3	cut	1.0	0.15

Table 2: Accuracies using the toy dataset. ND is the nesting depth. MM is the maximum amount we multiply by. ID is in-domain accuracy and OOD is out-of-domain, extrapolation, accuracy.

vary the number of nesting levels, the data includes all data with smaller nesting levels as well. We also vary the maximum amount we multiply by (the two options are just "twice" or "twice" and "thrice"). For the samples where there is no nesting, the training sequences have outputs between 1 and 20, for testing the lengths are between 21 and 25. For the samples where there are one or two levels of nesting, the training outputs range from 1 to 15, while the out-of-domain outputs range from 16-22. We use the same training set and test set sizes as SCAN.

We use the same fertility and cut models. The fertility model predicts the sum explicitly, while the cut model operates on a sequence of 'ONE' output tokens whose length is the evaluated input expression. We find that they are both effective at extrapolating when the nesting depth is shallow, but with deeper nesting, they fail (Table 2). The SCAN task is most similar to the case where the nesting depth is 2 and the maximum number we multiply by is 3, which is where we perform worst out-of-domain. These results suggest that the difficulty of calculating the length of the output action sequence in the SCAN task comes from both the nesting depth (e.g. the amount of multiplications that have to occur) and by having a large value (3 in this case, 4 in the SCAN data) as the maximum multiplier. That said, the fertility model is able to do much better than both the cut model in the toy arithmetic task and the fertility model in the SCAN task. This difference is likely due to the fact that the SCAN out of domain samples are longer than the toy arithmetic out of domain samples.

5 Experiment 2: Adding Additional Inductive Bias for Compositionality

5.1 Mathematical Supervision

We were unable to see high extrapolation performance using standard methods and the standard extrapolation dataset split. As such, we investigate alternative ways of including structure in the learning process. A variety of ways to do this have been suggested by prior work. These include specialized architectures, data augmentation, and meta-learning techniques to name a few (Andreas, 2019; Lake, 2019; Nye et al., 2020; Chen et al., 2020).

We choose a different approach, not investigated previously in the context of the SCAN task—adding additional supervision to encourage the models to learn compositionally. Specifically, we supervised models with examples of "showing their work" when calculating the length of SCAN output sequences. For example, for an input sequence like "jump around left twice and run", with an output length of 17, this additional supervision takes the form of a string like "$(((1+1)*4)*2)+1$", which evaluates to 17. Our goal in using supervision of this form is that our models will more easily be able to associate each token with a value and operation.

We generate the data we need for this additional form of supervision automatically by writing a program to map SCAN input sequences to mathematical expressions. This is easy because the SCAN data is synthetically generated.

The models we use are standard sequence-to-sequence models. The encoder consists of a 2-layer 200-dimensional bidirectional LSTM and the decoder is a 1-layer 200-dimensional LSTM. Our target sequences are the mathematical expressions that evaluate to the length of the action sequences. The output vocabulary for our seq2seq models consists of the individual characters (numerical digits, "$+$", "$*$", and parentheses), and to generate expressions at evaluation time we greedily sample from the decoder.

Using this method, we are able to achieve 100% exact-match accuracy on generating the expressions that evaluate to the lengths of out-of-domain sequences. While this method does not involve predicting the final output length, the expressions that the model produces can be evaluated by a standard calculator program to arrive at the final length prediction.

There are likely two reasons for our success. First is that using mathematical expressions to supervise models likely makes it easier for them to map input tokens to their contribution to the total length. The second reason this method is successful is likely due to the fact that the problem of mapping inputs and mathematical expressions does not actually require length extrapolation. The mathematical expressions and input sequences in the extrapolation case are the same size as the in-domain lengths. The difference between in domain and out-of-domain sequences is in which tokens occur more often—longer action sequences will have more nested multiplication and have more threes and fours. Shorter action sequences have less nested multiplication and more twos. We can summarize this success by noticing that we were able to effectively extrapolate through improved credit assignment and by converting the extrapolation problem into a one that did not require length extrapolation.

5.2 Program Supervision

The mathematical supervision proved effective in part because we were able to convert a task that required length extrapolation into one that did not. We can apply this principal not just to calculating the length of SCAN sequences, but to solving the original SCAN task mapping instructions to sequences of actions. We can do this by training models to produce *programs* that, when run, output the SCAN action sequences rather than predict the SCAN action sequences token by token themselves. Essentially, we are moving the SCAN task into the space of programs, where no length extrapolation is required.

We define this space of programs to include Python programs that generate SCAN action sequences. Similar to the case with mathematical expressions, because the SCAN dataset is procedurally generated, we can automatically generate the programs we need to supervise our model with. The models used are the same sequence-to-sequence models used to train models with mathematical expressions as supervision. The only difference is that the number of hidden dimension of the LSTMs has increased from 200 to 1000 to account for the larger set of tokens in the programs and the longer term dependencies that come from generating programs compared to mathematical expressions.

We find that we are able to achieve about 100% (though not perfect) accuracy on producing programs that evaluate to out-of-domain sequences. There are only three mistakes, and they all involve repeating a set of actions two times (which occurs in in-domain lengths) rather than three times (which only appears when extrapolating).

% Train Samples	ID Accuracy	OOD Accuracy
10%	0.286	0.118
20%	0.995	0.996
30%	0.998	0.900
50%	0.998	0.944
60%	1.0	1.0
70%	1.0	1.0
100%	1.0	0.999

Table 3: Training models for different dataset sizes allows us to investigate how many programs we need to generate in order to take advantage of the supervision they offer. Training set size is a percentage of the total number of training examples. Examples are chosen randomly. ID is in-domain accuracy, OOD is out of domain (extrapolation) accuracy.

5.3 Bootstrapping Additional Programmatic Supervision

A question that arises when training a model using extra programmatic supervision is whether we need to use the entire training dataset in order to benefit from training on programs rather than sequences. For example, in a situation where it is expensive to generate programs, it would be useful to consider the minimum number of programs necessary to achieve high extrapolative performance.

We take random samples of the training set of different sizes to train our sequence-to-sequence models on. We find that 10% of the training dataset is not enough to perform well, but 20% is enough to achieve high accuracy when we train with programs (Table 3).

While this is still a large number of examples, it suggests a self-training algorithm for getting the number of examples required down even farther. The algorithm is as follows:

1. Choose a training set with a small set of (INPUT, PROGRAM) pairs.

2. Train a seq2seq model on this training set

3. Run the trained model on the entire original training set, which consists of (INPUT, PROGRAM) pairs. For all programs:

   (a) Run the model to predict the program that generates the action sequence
   (b) Run the predicted program
   (c) If the program output matches the gold output, then add the input and program to the training set

4. Repeat until the training set stops growing.

The effectiveness of this strategy likely depends on the size and composition of the original training set. Using this algorithm, we are able to begin with a seed set 10% of the size of the entire dataset and end up with a set 83% the size of the entire dataset. Unfortunately, this increase did not translate to ideal performance. The in-domain accuracy was 82% while out-of-domain accuracy was 55%. This is better than the accuracy using only the seed set (see Table 3) but is not perfect. This is probably due to some examples that are not covered by the seed set and therefore never get learned. Future work should involve exploring the effect of this seed set and the dynamics of the self-training procedure. We will likely be able to see improvements if we choose this seed set intelligently—for example by ensuring the seed set includes examples of all individual concepts. Additionally, it could be useful to choose this seed set through some active learning procedure.

6 Conclusion

Despite the recent successes of deep learning models on many NLP tasks, human-like extrapolation remains difficult. Previous work notes that removing the EOS token from the end of training examples helps models extrapolate, but prevents them from ending sequences (Newman et al., 2020). In this work, we take on this challenge of determining how to end these sequences. In the end, we are able to do this task. Our method requires reframing the problem to remove the length-extrapolation component by training a sequence-to-sequence model to generate a mathematical expression that evaluates to the correct sequence length. We expand on this method by training models that are able to extrapolate with 100% accuracy on the original SCAN task by predicting programs that generate SCAN action sequences when run, and are able to achieve 55% accuracy on out of domain lengths with additional supervision for 10% of the training dataset.

In our case, and all other successful approaches to the SCAN extrapolation task, additional information is needed. Sometimes this information is included in the form of pretraining (Nye et al., 2020), sometimes in the architecture (Chen et al., 2020), and in our case in the data we use to supervise the model. Even though these methods are effective, the question of how they transfer to tasks with language is still important to consider. Nevertheless, they are all positive steps toward understanding what it takes for neural models to extrapolate.

References

Jacob Andreas. 2019. Good-enough compositional data augmentation. *arXiv preprint arXiv:1904.09545*.

Xinyun Chen, Chen Liang, Adams Wei Yu, Dawn Song, and Denny Zhou. 2020. Compositional generalization via neural-symbolic stack machines. *Advances in Neural Information Processing Systems*, 33.

Noam Chomsky. 1957. *Syntactic Structures*. Mouton, The Hague.

Michael Collins. 1999. Head-driven statistical models for natural language processing. *PhD dissertation, University of Pennsylvania*.

Stephen A Della Pietra, Mark Epstein, Salim Roukos, and Todd Ward. 1997. Fertility models for statistical natural language understanding. In *35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 168–173.

Yann Dubois, Gautier Dagan, Dieuwke Hupkes, and Elia Bruni. 2020. Location Attention for Extrapolation to Longer Sequences. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 403–413, Online. Association for Computational Linguistics.

Gottlob Frege. 1953. The foundations of arithmetic: A logico-mathematical enquiry into the concept of number, trans. jl austin. *Oxford: Basil Blackwell*, 3:3e.

Jonathan Gordon, David Lopez-Paz, Marco Baroni, and Diane Bouchacourt. 2019. Permutation equivariant models for compositional generalization in language. In *International Conference on Learning Representations*.

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

Tim Klinger, Dhaval Adjodah, Vincent Marois, Josh Joseph, Matthew Riemer, Alex 'Sandy' Pentland,

and Murray Campbell. 2020. A study of compositional generalization in neural models. *arXiv preprint arXiv:2006.09437*.

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, pages 2873–2882.

Brenden M Lake. 2019. Compositional generalization through meta sequence-to-sequence learning. In *Advances in Neural Information Processing Systems*, pages 9791–9801.

Qian Liu, Shengnan An, Jian-Guang Lou, Bei Chen, Zeqi Lin, Yan Gao, Bin Zhou, Nanning Zheng, and Dongmei Zhang. 2020. Compositional generalization by learning analytical expressions. *Advances in Neural Information Processing Systems*, 33.

Gary Marcus. 2018. Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631*.

Jeff Mitchell, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Extrapolation in NLP. *arXiv preprint arXiv:1805.06648*.

Richard Montague. 1970. Universal grammar. *Theoria*, 36(3):373–398.

Kenton Murray and David Chiang. 2018. Correcting length bias in neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, Brussels, Belgium. Association for Computational Linguistics.

Benjamin Newman, John Hewitt, Percy Liang, and Christopher D. Manning. 2020. The EOS decision and length extrapolation. In *BlackBoxNLP@EMNLP*.

Maxwell Nye, A. Solar-Lezama, J. Tenenbaum, and B. Lake. 2020. Learning compositional rules via neural program synthesis. *ArXiv*, abs/2003.05562.

Laura Ruis, Jacob Andreas, Marco Baroni, Diane Bouchacourt, and Brenden M Lake. 2020. A benchmark for systematic generalization in grounded language understanding. *arXiv preprint arXiv:2003.05161*.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2020. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? *arXiv preprint arXiv:2010.12725*.