The EOS Decision and Length Extrapolation

Benjamin Newman **John Hewitt** **Percy Liang** **Christopher D. Manning**
Stanford University
{blnewman,johnhew,pliang,manning}@stanford.edu

Abstract

Extrapolation to unseen sequence lengths is a challenge for neural generative models of language. In this work, we characterize the effect on length extrapolation of a modeling decision often overlooked: predicting the end of the generative process through the use of a special end-of-sequence (EOS) vocabulary item. We study an oracle setting—forcing models to generate to the correct sequence length at test time—to compare the length-extrapolative behavior of networks trained to predict EOS (+EOS) with networks not trained to (−EOS). We find that −EOS substantially outperforms +EOS, for example extrapolating well to lengths 10 times longer than those seen at training time in a bracket closing task, as well as achieving a 40% improvement over +EOS in the difficult SCAN dataset length generalization task. By comparing the hidden states and dynamics of −EOS and +EOS models, we observe that +EOS models fail to generalize because they (1) unnecessarily stratify their hidden states by their linear position is a sequence (structures we call *length manifolds*) or (2) get stuck in clusters (which we refer to as *length attractors*) once the EOS token is the highest-probability prediction.

1 Introduction

A core feature of the human language capacity is the ability to comprehend utterances of potentially unbounded length by understanding their constituents and how they combine (Frege, 1953; Chomsky, 1957; Montague, 1970). In NLP, while better modeling techniques have improved models' abilities to perform some kinds of systematic generalization (Gordon et al., 2019; Lake, 2019), these models still struggle to extrapolate; they have trouble producing and processing sequences longer than those seen during training even if they are composed of familiar atomic units.

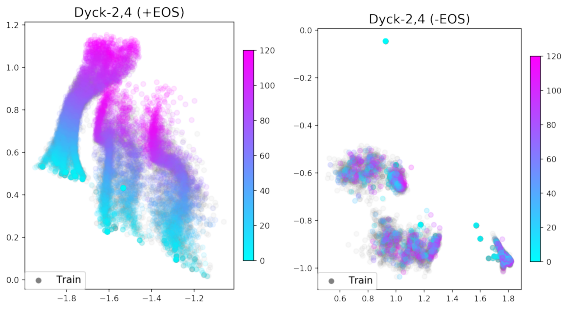


Figure 1: The hidden state dynamics differ between a model that has to predict EOS (**left**) and one that does not (**right**). Color varies with the hidden states' position in the sequence.

In this work, we investigate how an understudied modeling decision affects the ability of neural generative models of language to extrapolate to longer sequences. Models need to place a distribution over all sequence lengths, and to accomplish this when they generate from left to right, they make a decision about whether to generate a content token or a special end-of-sequence (EOS) token at each position in the sequence. In practice, however, the decision to have models predict where sequences end in this way, which we refer to as the *EOS decision*, yields models that predict sequences should end too early in length-extrapolation settings (Hupkes et al., 2020; Dubois et al., 2020).

We conceptually decompose length extrapolation into two components: (i) the ability to produce the right content, and (ii) the ability to know when to end the sequence. Standard evaluations like exact match or BLEU (Papineni et al., 2002) on length-generalization datasets like SCAN and gSCAN (Lake and Baroni, 2018; Ruis et al., 2020) evaluate the ability to perform both components, but models are known to fail because of (ii) alone, so these metrics don't test (i) in isolation. To help evaluate (i) independent of (ii), another evaluation

Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, pages 276–291
Online, November 20, 2020.

tool has been to exclude the effect of the EOS decision at test time by forcing models to generate to the correct output length (Lake and Baroni, 2018) or evaluating whether the strings they generate match prefixes of the correct outputs (Dubois et al., 2020; Hupkes et al., 2020).

In contrast, our work is the first to explore what happens if we don't have to perform (ii) (predicting where to end the sequence) **even at training time**. In this case, can we perform better at (i) (generating the right content)? We endeavor to answer this question by comparing the extrapolative ability of models trained without EOS tokens (−EOS) to models trained with them in their training data (+EOS).

First, we look at a simple formal language called Dyck-(k,m) that contains strings of balanced parentheses of k types with a bounded nesting depth of m (Hewitt et al., 2020); see Table 1. Looking at a simple bracket closing task we find that −EOS models close brackets with high accuracy on sequences 10 times longer than those seen at training time, whereas +EOS models perform substantially worse, suggesting fundamental differences in these models' ability to extrapolate. Investigating models' hidden states under principal component analysis (PCA) reveals that +EOS models unnecessarily stratify their hidden states by linear position while −EOS models do so less, likely contributing to −EOS models' improved performance (Figure 1). We call these stratified hidden states *length manifolds*.

Second, we use SCAN—a synthetic sequence-to-sequence dataset meant to assess systematic generalization. We focus on the split of the dataset made to test length extrapolation, showing that models in the +EOS condition perform up to 40% worse than those in the −EOS condition in an oracle evaluation setting where models are provided the correct output length. Visualizing the hidden states of both models using PCA reveals that +EOS models fail once they put high probability on the EOS token because their hidden states remain in place, exhibiting what we term a *length attractor*. −EOS models do not predict the EOS token, and their hidden states are able to move throughout hidden state space during the entire generation process, likely contributing to their improved performance.

Finally, we investigate extrapolation in a human language task—translation from German into English. We use an oracle evaluation where sequence lengths are chosen to maximize the BLEU score of generated translations. In contrast to our other experiments, for this more complex NLP task, we see that the −EOS condition less consistently contributes to extrapolation ability.

2 Related Work

The Difficulty of Extrapolation. Evidence from a number of settings shows that processing sequences longer than those seen at training time is a challenge for neural models (Dubois et al., 2020; Ruis et al., 2020; Hupkes et al., 2020; Klinger et al., 2020). This difficulty has been observed in datasets designed to test the ability of models to compositionally generalize, such as SCAN (Lake and Baroni, 2018), where the best performing neural models do not even exceed 20% accuracy on generating sequences of out-of-domain lengths, whereas in-domain performance is 100%. Extrapolation has also been a challenge for neural machine translation; Murray and Chiang (2018) identifies models producing translations that are too short as one of the main challenges for neural MT.

There are a number of reasons why extrapolation is challenging. At the highest level, some believe extrapolation requires understanding the global structure of a task, which standard neural architectures may have trouble emulating from a modest number of in-domain samples (Mitchell et al., 2018; Marcus, 2018; Gordon et al., 2019). Others focus more on implementation-level modeling problems, such as the EOS problem, where models tend to predict sequences should end too early, and try to address it with architectural changes (Dubois et al., 2020). Our work focuses on the latter issue; we show how the EOS decision affects not just a model's tendency for producing EOS tokens too early, but also the representations and dynamics underlying these decisions.

Addressing Extrapolation Issues Previous approaches to address the poor extrapolation ability of standard sequence models often focus on architectural changes. They include incorporating explicit memory (Graves et al., 2014) or specialized attention mechanisms (Dubois et al., 2020). Others have proposed search-based approaches—modifying beam search to prevent EOS from being assigned too much probability mass (Murray and Chiang, 2018), or searching through programs to produce sequences rather than having neural models produce sequences themselves (Nye et al.,

2020). Extrapolative performance has recently come up in the design of positional embeddings for Transformers (Vaswani et al., 2017). While recent Transformer models like BERT use learned positional embeddings (Devlin et al., 2019), when they were first introduced, position was represented through a combination of sinusoids of different periods. These sinusoids were introduced with the hope that their periodicity would allow for better length extrapolation (Vaswani et al., 2017). In this study we provide insight into popular architectures and the EOS decision instead of pursuing new architectures.

Length in Sequence Models Historically, for discrete symbol probabilistic sequence models, no EOS token was included in the standard presentations of HMMs (Rabiner, 1989) or in most following work in speech. However, the importance of having an EOS token in the event space of probabilistic sequence models was emphasized by NLP researchers in the late 1990s (e.g. Collins (1999) Section 2.4.1) and have been used ever since, including in neural sequence models. Requiring neural models to predict these EOS tokens means that these models' representations must track sequence length. There is empirical evidence that the hidden states of LSTM sequence-to-sequence models trained to perform machine translation models track sequence length by implementing something akin to a counter that increments during encoding and decrements during decoding (Shi et al., 2016). These results are consistent with theoretical and empirical findings that show that LSTMs can efficiently implement counting mechanisms (Weiss et al., 2018; Suzgun et al., 2019a; Merrill, 2020). Our experiments will show that tracking absolute token position by implementing something akin to these counters makes extrapolation difficult.

3 Experimental Setup

In all of our experiments, we compare models trained under two conditions: +EOS, where EOS tokens are appended to each training example, as is standard; and −EOS, where no EOS tokens are appended to training examples.

For our Dyck-(k,m) experiments (Section 4), we train on sequences with lengths from a set $\mathcal{L}_{\text{train}}$ and test on sequences with lengths from a set $\mathcal{L}_{\text{test}}$, where $\max \mathcal{L}_{\text{train}} < \min \mathcal{L}_{\text{test}}$.

For our SCAN and German-English MT experiments (Sections 5 and 6), we train on sequences

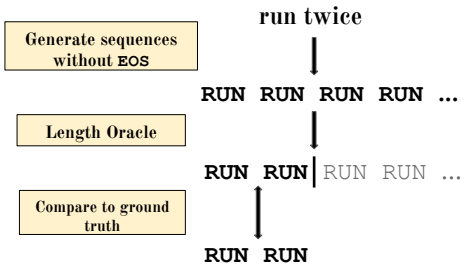


Figure 2: Above is an example of how **length oracle evaluation** works with the SCAN task. Models (both +EOS and −EOS) generate sequences without producing EOS tokens. The length oracle decides the optimal place to end the sequences, and compares the result to the ground truth.

of tokens shorter than some length ℓ, and test on sequences longer than ℓ. We refer to ℓ as a **length cut-off**, and a train-test split created in this manner is referred to as a **length-split**. We evaluate these models differently in each experiment.

Because we are interested in assessing the extrapolative ability of models apart from predicting where sequences end, we use what we refer to as a **length oracle evaluation**. It is performed by first allowing −EOS and +EOS models to output sequences longer than any gold targets, and then choosing the best position to end the sequence. For the +EOS models, this requires preventing the models from emitting EOS tokens during generation by setting the probability mass on the EOS token to 0. We call these models −EOS+Oracle and +EOS+Oracle respectively. Using these oracle metrics ensures that models are only assessed on knowing what content to generate rather than where to end sequences.

4 Experiment 1: Dyck-(k,m)

We conduct our first experiments with a simple language that gives us the opportunity to train an accurate, interpretable language model. This allows us to develop an intuition for how the EOS token affects the representations that models learn.

The language we study is Dyck-(k,m) (Hewitt et al., 2020). It is a formal language that consists of strings of well-nested, balanced brackets. There are k types of brackets, and a maximum nesting depth of m. For our experiments, we set $k = 2$, and vary m. Accepting this language involves implementing a bounded stack of up to size m, and allowing for pushing and popping any of k distinct elements (the bracket types) to and from the stack. We use the

sequence:	(	[	(	)	[	[	(	)	]	]	]	)
stack state:	(	([	([(	([	([[	([[[	([[[(	([[[	([[	([	(	

Table 1: A sample sequence and its stack states from Dyck-(2,6) and Dyck-(2,8), but not Dyck-(2,4) because the maximum nesting depth is 5.

m	$\mathcal{L}_{\text{train}}$	$\mathcal{L}_{\text{test}}$	+EOS	−EOS
4	[88, 116]	[950, 1050]	0.60	**0.86**
6	[184, 228]	[1840, 2280]	0.68	**0.98**
8	[328, 396]	[3280, 3960]	0.68	**0.96**

Table 2: Dyck-(k,m) bracket closing metric results, median of 5 independent training runs.

term **stack state** to refer to what this stack looks like at a given point in processing a sequence. An example sequence and its stack states are given in Table 1.

4.1 Methodology

We train our +EOS and −EOS models on a language modeling task using samples from Dyck-(2,4); Dyck-(2,6); and Dyck-(2,8). Minimum training sample lengths are chosen such that sequences of that length contain stack states of all possible depths at least three times in expectation. We include enough training samples to reliably achieve perfect bracket-closing accuracy on an in-domain evaluation.[1] Including *only* long sequences of brackets means that models only see evidence that they can end sequences (in the form of an EOS token) after a long stretch of symbols, not whenever their stack states are empty (and the −EOS model sees no evidence of ending at all). We endeavor to address this shortcoming in Section 4.3.

The out-of-domain samples (with lengths in $\mathcal{L}_{\text{test}}$) are 10 times longer than in-domain ones (with lengths in $\mathcal{L}_{\text{train}}$). (Table 2 gives the lengths for each condition.) The models we use are single layer LSTMs with $5m$ hidden states, and we train each model such that it has perfect in-domain length accuracy on a held-out validation set to ensure that we learn the in-domain data distribution well, rather than sacrifice in-domain performance for potential gains in extrapolative performance. Additionally, such in-domain accuracy is only possible because we are bounding the stack depth of our Dyck-(k,m) samples. If we were using the unbounded Dyck-k languages (with $k > 1$), we would not be able to see good in-domain performance, and thus we would not expect to see acceptable out-of domain performance either (Suzgun et al., 2019b,a).

For evaluation, we use the same metric as Hewitt et al. (2020) as it evaluates the crux of the task: maintaining a long-term stack-like memory of brackets. In principle, we want to ensure that whenever a model can close a bracket, it puts more probability mass on the correct one. Our metric is as follows: whenever it is acceptable for the model to close a bracket, we compute how often our models assign an large majority (more than 80%) of the probability mass assigned to *any* of the closing brackets to the *correct* closing bracket. For example, if 20% of the probability mass is put on either closing bracket, the model gets credit if it puts more than 16% of the mass on the correct bracket. Because in each sample there are more bracket pairs that are (sequentially) close together than far, we compute this accuracy separately for each distance between the open and close bracket. Our final score is then the mean of all these accuracies.

We also report a standard metric: perplexity for sequences of in-domain and out-of-domain lengths. We choose perplexity because it incorporates the probabilistic nature of the task—we cannot evaluate using exact match because at almost all decoding steps (other than when the stack has m brackets on it) there are two equally valid predictions that the model can make.

4.2 Results

Recall that we train all models to obtain perfect in-domain bracket closing accuracy on a held-out validation set. However, the +EOS models' performance is severely degraded on out-of-domain sequences 10 times longer than in-domain ones, while the −EOS models still perform well across all conditions (Table 2). The perplexity results mirror the results of the bracket closing task (Appendix B).

Because the Dyck-(k,m) is relatively simple, the model's hidden states are interpretable. We run the model on the training data, stack the models' hidden states, extract their top two principal components, and note that they form clusters (Figure 3) that encode stack states (Appendix; Figure 5). We observe that as both models process sequences, they hop between the clusters. However, in the +EOS model, the hidden states move from one side of each cluster to the other during processing, while this does not happen in the −EOS model.

[1] The code to generate these samples is available at https://github.com/bnewm0609/eos-decision.

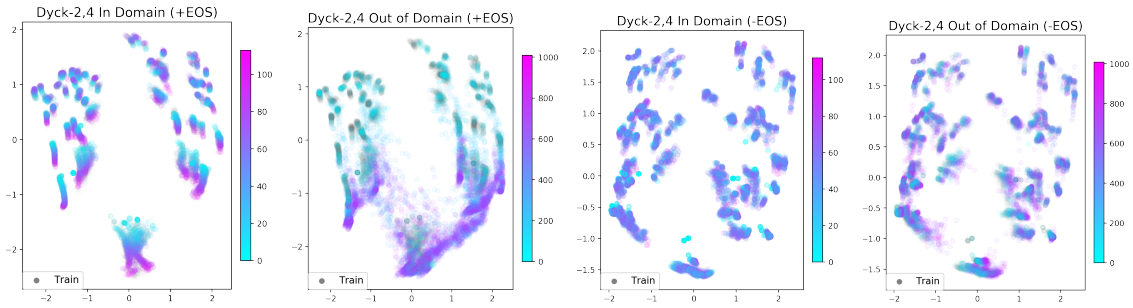


Figure 3: The top two principal components explain 67.52% of the variance for the +EOS model and 62.27%of the variance for the −EOS hidden states. The color scale corresponds to index in the sequence. The bottom-most cluster (centered around $(0, -2.0)$ in the +EOS plot and $(0, -1.5)$ in the −EOS plot) contains all of the hidden states where the stack is empty and the sequence can end.

Because of this, we refer to these elongated clusters as **length manifolds**.

As the +EOS model processes inputs past the maximum length it has seen during training, the representations lie past the boundary of the clusters, which we observe leads to a degradation of the model's ability to transition between states. We hypothesize this is because the recurrent dynamics break near the edges of the training-time length manifolds. We can see this degradation of the model dynamics in the out-of-domain +EOS plot (Figure 3). The hidden states at positions further in the sequences congregate at the tips of the length manifolds, or completely deviate from the training data. Contrast this plot to the in-domain −EOS plot, where there are no length manifolds visible and the out-of-domain −EOS plot, where we can see some length manifolds, but less evidence of the degradation of predictions (Table 2).

We hypothesize that these length manifolds form because +EOS models need to predict where each sequence ends in order to confidently put probability mass on the EOS token.[2] Our main takeaway is then that this length tracking has negative effects when the model is used to predict longer sequences.

4.3 Predicting the End of Sequences

Knowing where a Dyck-(k,m) sequence can end is an important part of being able to accept strings from the language. While we focus more on the ability of models to extrapolate in this work, and our evaluations do not depend on the ability of models to end sequences, we do observe in our plots of hidden states that the cluster of hidden states corresponding to the empty stack state (where sequences can end) is linearly separable from the other clusters (Figure 3). This linear-separability suggests we can train a linear classifier on top of model hidden states, a probe, to predict where sequences can end (Conneau et al., 2018; Ettinger et al., 2018).

m	+EOS		−EOS	
	ID	OOD	ID	OOD
4	1.0	0.90	1.0	0.92
6	1.0	0.95	1.0	1.0
8	1.0	0.97	1.0	1.0

Table 3: Accuracies for predicting when Dyck-$(2,m)$ sequences can end. Each entry is the median of five runs. Note that even though +EOS model is trained with an EOS token, it does not receive signal that it can end at other empty stack states, so such a probe is needed for models to learn when to end sequences.

We train a binary classifier to predict if a sequence can end at every token position. The input to this classifier at time-step t is the sequence models' hidden state at t, and it is trained to predict 1 if the sequence can end and 0 otherwise. We find that this simple method is able to predict where sequences end with high accuracy for in-domain sequences, and higher accuracy for the −EOS models compared to the +EOS models for the longer out-of-domain sequences (Table 3). This distinction mirrors the disorganization of the hidden states in the +EOS models hidden states when they extrapolate (Figure 3).

5 Experiment 2: SCAN

While the Dyck-(k,m) experiments give us good intuitions for how and why EOS tokens affect extrapolation, we study SCAN to extend to sequence-to-sequence modeling, and since it is a well-studied

[2]Close brackets are more likely than open brackets at training time near the maximum training length, since all sequences must end in the empty stack state; this may explain why −EOS models still show some length-tracking behavior.

dataset in the context of length extrapolation.

SCAN is a synthetic sequence-to-sequence dataset meant to simulate data used to train an instruction-following robot (Lake and Baroni, 2018). The inputs are templated instructions while the outputs correspond to the sequences of actions the robot should take. For example, the input instruction "walk left twice" maps to the output action sequence `TURN_LEFT WALK TURN_LEFT WALK`. There are 13 types in the input vocabulary—five actions (turn, run, walk, jump, look); six modifiers (left, right, opposite, around, twice, thrice); and two conjunctions (and, after), and different combinations of these tokens map to different numbers of actions. For example, "turn left twice" maps to `TURN_LEFT TURN_LEFT` while "turn left thrice" maps to `TURN_LEFT TURN_LEFT TURN_LEFT`. This is important for our exploration of models' abilities to extrapolate because it means that while the lengths of the input instructions range from one to nine tokens, the output action sequences lengths vary from one to 48 tokens.

5.1 Length splits in SCAN

Lake and Baroni (2018) define a 22-token length split of the dataset, in which models are trained on instructions with sequences whose outputs have 1–22 tokens (16990 samples) and are evaluated on sequences whose outputs have 22–48 tokens (3920 samples). Previous work has noted the difficulty of this length split—the best approach where sequences are generated by a neural model achieves 20.8% accuracy (Lake and Baroni, 2018).[3] Specifically, this extra difficulty stems from the fact that neural models only perform the SCAN task well when they are trained with sufficient examples of certain input sequence templates (Loula et al., 2018). In particular, the Lake and Baroni (2018) length split lacks a template from the training data: one of the form "walk/jump/look/run around left/right thrice." Alone, this template results in an output sequence of 24 actions, and thus is not included in the training set. Approximately 80% of the test set is comprised of sequences with this template, but models have not seen any examples of it at training time, so they perform poorly. Thus, we create additional length splits that have more examples of this template, and evaluate using them in addition to the standard 22-token length split of SCAN.

[3]Note that Nye et al. (2020) are able to achieve 100% accuracy on the SCAN length split, though their set-up is different—their neural model searchers for the SCAN grammar rather than generating sequences by token.

5.2 Methodology

We generate the 10 possible length splits of the data with length cutoffs ranging from 24 tokens to 40 (See the header of Table 4). Not every integer is represented because SCAN action sequences do not have all possible lengths in this range. For each split, we train the LSTM sequence-to-sequence model that Lake and Baroni (2018) found to have the best in-domain performance. Like in the previous experiment, we train two models—a `+EOS` model with an `EOS` token appended to each training sample output, and a `-EOS` model without them. We use the standard evaluation of Lake and Baroni (2018): greedy decoding and evaluating based on exact match. Because we are interested in how well our models can extrapolate separately from their ability to know when an output should end, we use force our models to generate to the correct sequence length before comparing their predictions to the gold outputs. For the `+EOS` model, this entails preventing the model from predicting `EOS` tokens by setting that token's probability to 0 during decoding. This gives us the `-EOS+Oracle` and `+EOS+Oracle` evaluations. We additionally report the exact match accuracy when the `+EOS` model is allowed to emit an `EOS` token (`+EOS`), which is the standard evaluation for this task (Lake and Baroni, 2018).

5.3 Results

We find that the `-EOS+Oracle` models consistently outperform `+EOS+Oracle` models across all length splits (Table 4). We also observe that after including sequences up to length 26, the models have seen enough of the new template to perform with accuracy $\geq 80\%$ on the rest of the long sequences. However, the question of what the `-EOS` model is doing that allows it to succeed remains. The `+EOS` model fails in the non-oracle setting by predicting that sequences should end before they do, and the `+EOS+Oracle` model fails because once it decodes past its maximum training sequence length, it tends to either repeat the last token produced or emit unrelated tokens. The `-EOS+Oracle` model succeeds, however, by repeating the last few tokens when necessary, so as to complete the last portion of a *thrice* command for example. (See Table 5 for two such examples).

We compute the top two principal components

	ℓ *(length cutoff)*	22	24	25	26	27	28	30	32	33	36	40
LSTM	*+EOS*	*0.16*	*0.08*	*0.26*	*0.61*	*0.72*	*0.64*	*0.60*	*0.67*	*0.45*	*0.47*	*0.85*
	+EOS+Oracle	0.18	0.46	0.47	0.71	0.82	0.77	0.80	0.84	0.74	0.81	0.95
	−EOS+Oracle	**0.61**	**0.57**	**0.54**	**0.83**	**0.92**	**0.97**	**0.90**	**1.00**	**0.99**	**0.98**	**1.00**
Transformer	*+EOS*	*0.00*	*0.05*	*0.04*	*0.00*	*0.09*	*0.00*	*0.09*	*0.35*	*0.00*	*0.00*	*0.00*
	+EOS+Oracle	0.53	0.51	**0.69**	0.76	0.74	0.57	0.78	0.66	0.77	**1.00**	0.97
	−EOS+Oracle	**0.58**	**0.54**	0.67	**0.82**	**0.88**	**0.85**	**0.89**	**0.82**	**1.00**	**1.00**	**1.00**

Table 4: Exact match accuracies on length splits. Reported results are the median of 5 runs. The +EOS rows (italicized) are not comparable to the other rows because they are not evaluated in an an oracle setting—they are provided for reference.

Instruction	Model	Predicted Actions
walk around left twice and walk around left thrice	+EOS+Oracle	TURN_LEFT WALK ... WALK WALK JUMP WALK WALK JUMP WALK WALK JUMP
	−EOS+Oracle	TURN_LEFT ... WALK TURN_LEFT WALK TURN_LEFT WALK TURN_LEFT WALK TURN_LEFT WALK
run around left twice and run around right thrice	+EOS+Oracle	TURN_LEFT ... TURN_RIGHT RUN RUN
	−EOS+Oracle	TURN_LEFT ... TURN_RIGHT RUN TURN_RIGHT

Table 5: Two examples of errors a +EOS model makes from the 22-token length split: generating irrelevant actions and repeating the last action. Red tokens are incorrect, blue are correct. See the appendix for more examples.

of the decoder hidden states and plot them as we did for the Dyck-(k,m) models. While the top two principal components explain a modest amount of the variance (27.78% for −EOS and 29.83% for +EOS), they are still interpretable (Figure 4). In particular, we can see that the recurrent dynamics break differently in this case compared to the Dyck-$(2,4)$ case. Here, they break once the +EOS model puts a plurality of probability mass on the EOS token (4b). Once this happens, the hidden states remain in the cluster of EOS states (4a, pink cluster). The −EOS condition does not have such an attractor cluster (4d), so these same hidden states are able to transition between clusters associated with various token identities (4c). This freedom likely aids extrapolative performance.

A figure with hidden states colored by length is available in Appendix C. Both the +EOS and −EOS show evidence of tracking token position because knowing how many actions have been taken is an important component of the SCAN task (for example, to follow an instruction like "turn left twice and walk around right thrice", a model needs to recall how many times it has turned left in order to correctly continue on to walking around thrice).

The results we see here shed some light on the length generalization work by Lake and Baroni (2018). They note that when they prevent a model from predicting EOS (our +EOS+Oracle metric), they achieved 60.2% exact match accuracy on the 22-token length split with a GRU with a small hidden dimension. This number is comparable to the number we find using −EOS+Oracle metric. The success of the low-dimensional GRU may very well be related to its failure to implement counters as efficiently as LSTMs (Weiss et al., 2018)—if the model cannot count well, then it may not learn length attractors.

5.4 Transformer-based models

All of our experiments so far have used LSTMs, but transformers are the current state of the art in many NLP tasks. They also can be trained with the standard fixed sinusoidal positional embeddings, which Vaswani et al. (2017) suggest might help with extrapolation due to their periodicity. We train transformer +EOS and −EOS models with sinusoidal positional embeddings on the SCAN length splits and observe that +EOS models perform about as poorly as +EOS models in the LSTM case as well (Table 4). Despite using periodic positional embeddings, +EOS models are not able to extrapolate well in this setting.

6 Experiment 3: Machine Translation

Our final experiment focuses on how well the extrapolation we see in the case of SCAN scales to a human-language task—translation from German to

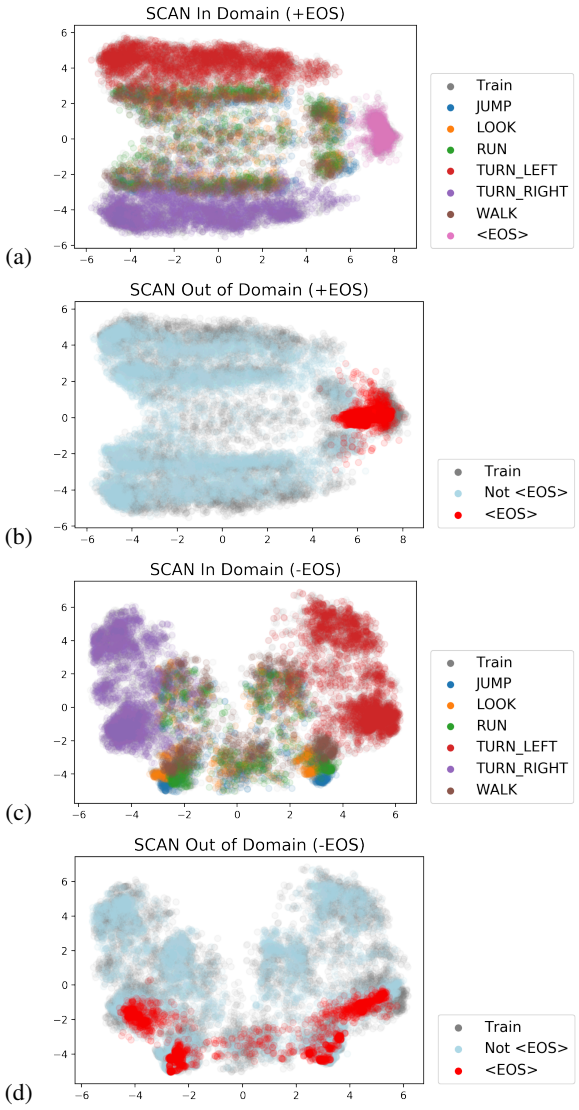


Figure 4: The top two principal components of the hidden states for the +EOS and −EOS LSTM models trained on the SCAN 22-token length split. 4a and 4c color hidden states by the identity of the gold output tokens for the +EOS and −EOS conditions respectively. 4b and 4d color out-of-domain sequences by whether the +EOS model puts a plurality of probability mass is put on the EOS token (<EOS>, red) or on any other token (Not <EOS>, light blue). It is important to note that in both plots, the colors are derived from the +EOS model as we want to investigate how the −EOS model performs in the places where the +EOS model errs.

English from the WMT2009 challenge. This data is much more complex than the SCAN task and has many more subtle markers for length, which might act as proxies for the EOS token, meaning that removing EOS tokens from the training data has a smaller impact on the models' extrapolative abilities. We find that there is very little difference between the +EOS and −EOS models' performance on out-of-domain lengths compared to SCAN, and while −EOS perform better in out-of-domain settings more often than +EOS models, removing the EOS token does not conclusively help with extrapolation.

6.1 Methodology

We use a subset of 500,000 German to English sentences from the WMT2009 challenge Europarl training set. The median English sentence length has 24 tokens; we create three length splits: with $\ell = 10$, 15, and 25, giving us approximately 61, 126, and 268 thousand training examples respectively. We train +EOS and −EOS LSTM models with 2 layers for the encoder and decoder with 500 and 1000-dimensional hidden states as well as Transformer models, the result of a hyperparameter search described in Appendix A.3. Additionally, the training data in both conditions is modified by removing any final punctuation from the sentences that could serve as explicit markers for length. Finally, our evaluations are very similar to the ones in our SCAN experiments, but our metric is BLEU rather than exact match accuracy (Papineni et al., 2002).

We report standard BLEU for the +EOS condition, and two length oracles, where we prevent the +EOS models from producing EOS tokens and force sequences produced by all models to stop decoding at the length that maximizes their BLEU score. In practice, computing the BLEU score for all sequence lengths is expensive, so we consider only lengths within a window of 7 tokens on either side of the gold target length, which gets us within ∼1% of the true oracle score. We train our models using OpenNMT (Klein et al., 2017) and calculate BLEU using the sacreBLEU package for reproducibility (Post, 2018).

6.2 Results

We observe a slight increase in extrapolative performance for −EOS models over +EOS models for LSTMs in the 15 and 10-token length splits, and transformers in the 15 and 25-token length splits, but have no consistent takeaways (Table 6).

We also report in-domain BLEU scores. There is some variation between these, but mostly less than between the out-of-domain scores, which may suggest that the difference of extrapolative performance in those models is meaningful. Additionally, we do not report plots of the top two principal components for these models because they only explain

ℓ	10		15		25	
LSTM	ID	OOD	ID	OOD	ID	OOD
+EOS	*25.25*	*1.75*	*25.27*	*7.24*	*28.14*	*16.19*
+EOS+Oracle	26.42	4.64	**26.43**	11.75	**29.01**	**20.34**
-EOS+Oracle	**26.59**	**5.14**	25.84	**12.53**	28.70	20.12
Δ	0.17	0.5	-0.59	0.78	-0.31	-0.22
Transformer						
+EOS	*24.91*	*1.27*	*25.67*	*5.16*	*28.75*	*13.32*
+EOS+Oracle	26.15	**4.87**	26.33	10.37	**29.29**	17.14
-EOS+Oracle	**26.73**	4.65	**26.81**	**11.65**	29.13	**17.39**
Δ	0.58	-0.22	0.51	1.32	0.16	0.25

Table 6: German-to-English translation BLEU scores. ID is on held-out data with the same length as the training data and OOD is data longer than that seen at training.

3% of the variance and are not visually interesting.

We speculate that we do not see the −EOS models consistently outperforming the +EOS ones because are likely more subtle indicators of length that models in both conditions pick up on, rendering the presence of EOS tokens less relevant. Further analysis should look to mitigate these length cues as well as investigate additional length splits.

7 Discussion

For most purposes of generative modeling, it is necessary to know when to end the generative process in order to use a model; put another way, a −EOS model is not usable by itself. The immediate engineering question is whether there exists a way to learn a distribution over when to end the generative process that does not have the same negative effects as the EOS decision.

In our Dyck-(k,m) experiments, we observed that even in −EOS models, there exists a linear separator in hidden state space between points where the generative process is and isn't allowed to terminate. We explore training probe to find this linear separator, and are able to predict where sequences end with high accuracy (Table 3). In our other experiments, we did not see such a simple possible solution, but can speculate as to what it would require. In particular, the length manifold and length attractor behaviors seem to indicate that length extrapolation fails because the conditions for stopping are tracked in these models more or less in terms of absolute linear position. As such, it is possible that a successful parameterization may make use of an implicit checklist model (Kiddon et al., 2016), that checks off which parts of the input have been accounted for in the output.

Can we use the same neural architectures, in the +EOS setting, while achieving better length extrapolation? Our PCA experiments seem to indicate that −EOS models' length tracking is (at least) linearly decodable. This suggests that training models with an adversarial loss against predicting the position of the token may encourage +EOS models to track length in a way that allows for extrapolation.

8 Conclusion

In this work, we studied a decision often overlooked in NLP: modeling the probability of ending the generative process through a special token in a neural decoder's output vocabulary. We trained neural models to predict this special EOS token and studied how this objective affected their behavior and representations across three diverse tasks. Our quantitative evaluations took place in an oracle setting in which we forced all models to generate until the optimal sequence length at test time. Under this setting, we consistently found that networks trained to predict the EOS token (+EOS) had a worse length-extrapolative ability than those not trained to (−EOS). Examining the hidden states of +EOS and −EOS networks, we observed that +EOS hidden states exhibited *length manifolds* and *length attractor* behaviors that inhibit extrapolation, and which otherwise identical −EOS networks do not exhibit. Thus, we argue that training to predict the EOS token causes current models to track generated sequence length in their hidden states in a manner that does not extrapolate out-of-domain. When EOS tokens were first introduced to NLP ensure probabilistic discrete sequence models maintained well-formed distributions over strings, these models, with a small sets of hidden states, did not readily pick up on these length correlations. However, when this NLP technique was ported to more expressive neural networks, it hid potentially useful length-extrapolative inductive biases. We see the evidence presented here as a call to explore alternative ways to parameterize the end of the generative process.

Acknowledgments

The authors thank Nelson Liu and Atticus Geiger for comments on early drafts, and to our reviewers whose helpful comments improved the clarity of this work. JH was supported by an NSF Graduate Research Fellowship under grant number DGE-1656518.

References

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

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

Alexis Conneau, German Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. 2018. What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2126–2136, Melbourne, Australia. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Minneapolis, Minnesota. Association for Computational Linguistics.

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.

Allyson Ettinger, Ahmed Elgohary, Colin Phillips, and Philip Resnik. 2018. Assessing composition in sentence vector representations. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1790–1801, Santa Fe, New Mexico, USA. 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*.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401*.

John Hewitt, Michael Hahn, Surya Ganguli, Percy Liang, and Christopher D. Manning. 2020. RNNs can generate bounded hierarchical languages with optimal memory. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics.

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.

Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. 2016. Globally coherent text generation with neural checklist models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 329–339, Austin, Texas. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. 2017. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, Vancouver, Canada. Association for Computational Linguistics.

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.

João Loula, Marco Baroni, and Brenden Lake. 2018. Rearranging the familiar: Testing compositional generalization in recurrent networks. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 108–114, Brussels, Belgium. Association for Computational Linguistics.

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

William Merrill. 2020. On the linguistic capacity of real-time counter automata. *arXiv preprint arXiv:2004.06866*.

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.

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

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels. Association for Computational Linguistics.

Lawrence R Rabiner. 1989. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

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*.

Xing Shi, Kevin Knight, and Deniz Yuret. 2016. Why neural translations are the right length. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2278–2282, Austin, Texas. Association for Computational Linguistics.

Mirac Suzgun, Yonatan Belinkov, Stuart Shieber, and Sebastian Gehrmann. 2019a. LSTM networks can perform dynamic counting. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 44–54, Florence. Association for Computational Linguistics.

Mirac Suzgun, Yonatan Belinkov, and Stuart M. Shieber. 2019b. On evaluating the generalization of LSTM models in formal languages. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2019*, pages 277–286.

Ashish Vaswani, 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.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. On the practical computational power of finite precision RNNs for language recognition. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Melbourne, Australia. Association for Computational Linguistics.

A Experiment Details

A.1 Dyck-(k,m)

The language Dyck-(k,m), for $k, m \in \mathbb{Z}^+$, is a set of strings over the vocabulary consisting of k open brackets, k close brackets, and the `EOS` token. To generate a dataset for Dyck-(k,m), we must thus define a distribution over the language, and sample from it. Defining this distribution characterizes the statistical properties of the language – how deeply nested is it on average, how long are the strings, etc.

At a high level, we wanted our samples from Dyck-(k,m) to be difficult – that is, to require long-term memory, and to traverse from empty memory to its maximum nesting depth (m) and back multiple times in the course of a single sample, to preclude simple heuristics like remembering the first few open brackets to close the last few. For further discussion of evaluation of models on Dyck-k and similar formal languages, see Suzgun et al. (2019b).

The distribution we define is as follows. If the string is balanced, we end the string with probability $1/2$ and open any of the k brackets with probability $1/2$. If the string has more than 0 unclosed open parentheses but fewer than m (the bound), we open a bracket with probability $1/2$ and close the most recently opened bracket with probability $1/2$; if we open a bracket, we choose uniformly at random from the k. If the string has m unclosed open brackets, then it has reached its nesting bound, and we close the most recently opened bracket with probability 1.

Sampling directly from this distribution would lead to exponentially short sequences, which would break our desideratum of difficulty. So, we truncate the distribution by the length of strings, ensuring a minimum (and a maximum) length at training time.

We define the length truncation as follows. Consider that the number of unclosed open brackets at timestep t is some number between 0 and m. At any timestep, we move from s_i to s_{i+1} with probability 1/2 (opening a bracket), or from s_i to s_{i-1} with probability 1/2 (closing a bracket.) Let these states be s_0 to s_m. What we'd like to see is that we eventually move from state s_0 to s_m and back to s_0 multiple times, making sure that the network cannot use positional heuristics to remember the first few brackets in order to close the last few. We ensure this in expectation by making sure that the minimum truncation length of samples is defined so that in expectation, sampled strings traverse the Markov chain from s_0 to s_m and back to s_0 at least three times.

In practice, we set the minimum length of training samples to $6m(m-2)+40$, and the maximum length to $7m(m-2)+60$. Note the quadratic dependence on m; this is because of the expected hitting time of s_0 to s_{m+1} of the $m+1$-state Markov chain wherein there is a $1/2$ probability of moving in each direction. We add constant factors to ensure there are no extremely short sequences for small m; the exact truncation lengths are not as important as the general scaling.

Additionally, as m increases, the number of training examples required for consistently high in-domain performance also increases. We use $10^{\frac{m}{2}+2}$ training examples for each of the values of m. For all values of m, we use 10000 test samples.

Our models are single-layer LSTMs with $5m$ hidden states. We a batch size of $2^{\frac{m}{2}+2}$, and use the Adam optimizer with a learning rate of 0.01 (Kingma and Ba, 2015).

A.2 SCAN

The SCAN model we train is the model that achieves the best accuracy in-domain in the experiments of Lake and Baroni (2018). This is an LSTM sequence-to-sequence model with 2 layers of 200-dimensional hidden units and no dropout or attention. Optimization was done with the Adam optimizer (Kingma and Ba, 2015), learning rate of 0.001. We used OpenNMT to train the transformer models (Klein et al., 2017). These models have 2 layers, 128-dimensional hidden states, 8 attention-heads, and 1024-dimensional feedfoward layers. All other hyperparameters were the defaults suggested by OpenNMT for training transformer sequence-to-sequence models.

A.3 Machine Translation

The German-English MT models we train were also trained with Open NMT. We trained LSTM models with attention and 1000-dimensional and 500-dimensional hidden states and two layers. All other parameters were specified by the OpenNMT defaults as of `OpenNMT-py` version 1.1.1. For the 25-token length split we report results with 1000-dimensional hidden states, and with 500-dimensional hidden states for the 10-length split. For the 15-token length split the `+EOS` model used 1000-dimensional hidden states while the `+EOS+Oracle` and `-EOS+Oracle` models used

500-dimensional hidden states. This difference is a result of running hyperparameter optimization and finding that +EOS baseline performed better with a larger hidden state size. Our transformer models were trained using the hyperparameters suggested by OpenNMT for training transformer sequence-to-sequence models. This includes 6 layers for the decoder and decoder, 512-dimensional hidden states and 2048-dimensional feed-forward layers.

B Additional Plots and Results for Dyck-2,m Languages

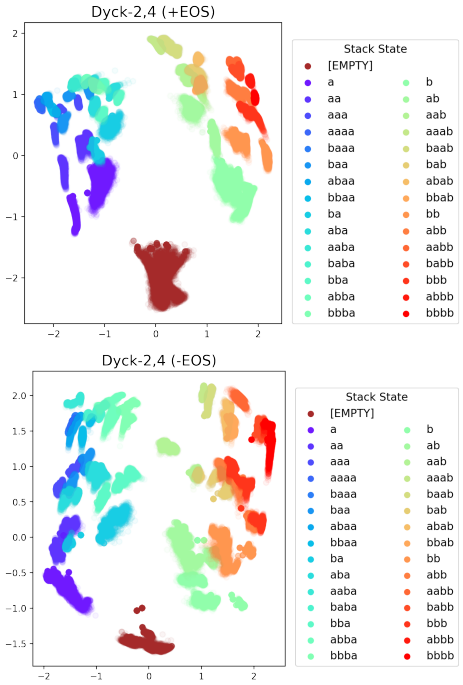


Figure 5: Above we can see a plot of the top two principal components of hidden states from the +EOS and −EOS LSTMs trained on they Dyck-2,4. These hidden states come from the training samples. The hidden states are colored by stack state—at that point in the sequence, what brackets must be closed and in what order. Because this is Dyck-2,4 there are 2 types of brackets which we can call "a" and "b". An "a" in stack state in the legend represents that there is a bracket of type "a" on the stack, and the same for "b". The top of the stack is the right-most character. We can see that the the first principal component roughly represents whether an "a" or "b" is at the top of the stack.

m	+EOS	−EOS	−EOS + Random Cutoff
4	0.60	0.86	**1.0**
6	0.68	0.98	**1.0**
8	0.68	0.96	**1.0**

Table 8: Dyck-$(2,m)$ bracket closing metric results, median of 5 independent training runs.

m	+EOS		−EOS	
	ID	OOD	ID	OOD
4	2.39	7.09	2.38	**2.93**
6	2.52	5.14	2.52	**2.60**
8	2.60	5.24	2.59	**2.68**

Table 7: Perplexities of the +EOS and −EOS models trained on Dyck-2,m languages on in-domain (ID) and out-of-domain (OOD) strings. The perplexities support the bracket closing results—the perplexities are very similar in-domain and out-of-domain for −EOS model but differ substantially for the +EOS model. Reported value is the median of five runs, lower is better.

Our −EOS models were not able to achieve perfect accuracy on the the bracket-closing task on longer sequences. We suspect that this failure is due to the −EOS models picking up on other signals for sequence length that were present in our Dyck-$(2,m)$ samples. Namely, all of our samples ended in an empty bracket state, so the final non-EOS token was always a closing bracket. To address this issue, we explored removing a random number of additional tokens from the end of the −EOS models' training data (−EOS +Random Cutoff). We found that doing this allowed us to perfectly extrapolate to sequences 10x longer (Table 8).

C Additional Plots and Samples for SCAN Task

	run around left after jump around left twice		look around left thrice		run around left thrice after jump around right thrice	
	+EOS	−EOS	+EOS	−EOS	+EOS	−EOS
1	TURN_LEFT	TURN_LEFT	TURN_LEFT	TURN_LEFT	TURN_RIGHT	TURN_RIGHT
2	JUMP	JUMP	LOOK	LOOK	JUMP	JUMP
3	TURN_LEFT	TURN_LEFT	TURN_LEFT	TURN_LEFT	TURN_RIGHT	TURN_RIGHT
4	JUMP	JUMP	LOOK	LOOK	JUMP	JUMP
5	TURN_LEFT	TURN_LEFT	TURN_LEFT	TURN_LEFT	TURN_RIGHT	TURN_RIGHT
6	JUMP	JUMP	LOOK	LOOK	JUMP	JUMP
7	TURN_LEFT	TURN_LEFT	TURN_LEFT	TURN_LEFT	TURN_RIGHT	TURN_RIGHT
8	JUMP	JUMP	LOOK	LOOK	JUMP	JUMP
9	TURN_LEFT	TURN_LEFT	TURN_LEFT	TURN_LEFT	TURN_RIGHT	TURN_RIGHT
10	JUMP	JUMP	LOOK	LOOK	JUMP	JUMP
11	TURN_LEFT	TURN_LEFT	TURN_LEFT	TURN_LEFT	TURN_RIGHT	TURN_RIGHT
12	JUMP	JUMP	LOOK	LOOK	JUMP	JUMP
13	TURN_LEFT	TURN_LEFT	TURN_LEFT	TURN_LEFT	TURN_RIGHT	TURN_RIGHT
14	JUMP	JUMP	LOOK	LOOK	JUMP	JUMP
15	TURN_LEFT	TURN_LEFT	TURN_LEFT	TURN_LEFT	TURN_RIGHT	TURN_RIGHT
16	JUMP	JUMP	LOOK	LOOK	JUMP	JUMP
17	TURN_LEFT	TURN_LEFT	LOOK	TURN_LEFT	TURN_RIGHT	TURN_LEFT
18	JUMP	RUN	JUMP	LOOK	JUMP	RUN
19	TURN_LEFT	TURN_LEFT	LOOK	TURN_LEFT	TURN_LEFT	TURN_LEFT
20	RUN	RUN	JUMP	LOOK	RUN	RUN
21	TURN_LEFT	TURN_LEFT	LOOK	TURN_LEFT	TURN_LEFT	TURN_LEFT
22	RUN	RUN	JUMP	LOOK	RUN	RUN
23	TURN_LEFT	TURN_LEFT	LOOK	TURN_LEFT	TURN_LEFT	TURN_LEFT
24	RUN	RUN	LOOK	LOOK	RUN	RUN
25					TURN_LEFT	TURN_LEFT
26					RUN	RUN
27					TURN_LEFT	TURN_LEFT
28					RUN	RUN
29					TURN_LEFT	TURN_LEFT
30					RUN	RUN
31					TURN_LEFT	TURN_LEFT
32					RUN	RUN
33					RUN	TURN_LEFT
34					RUN	RUN
35					RUN	TURN_LEFT
36					RUN	RUN
37					RUN	TURN_LEFT
38					RUN	RUN
39					RUN	TURN_LEFT
40					RUN	RUN
41					RUN	TURN_LEFT
42					RUN	RUN
43					RUN	TURN_LEFT
44					RUN	RUN
45					RUN	TURN_LEFT
46					RUN	RUN
47					RUN	TURN_LEFT
48					RUN	RUN

Table 9: More illustrative selections of errors the +EOS and −EOS models make when trained on the SCAN 22-token length split. In the left-most column we see that the +EOS model makes a mistake on the conjunction that the −EOS model does not make. In the middle column we see the +EOS model fail on the template that is not seen at training time. In the right-most column we see an example where both models fail: the conjunction of two templates not seen at training time. The +EOS model begins the third "jump around" but does not finish it, while the −EOS model switches completely to the first "turn left".

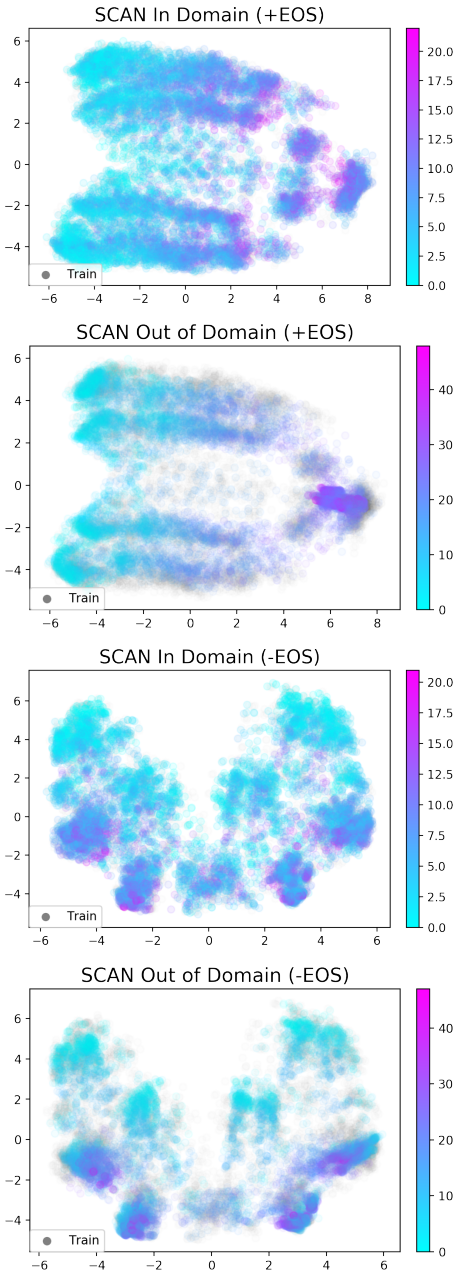
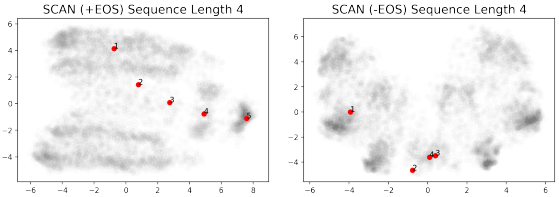
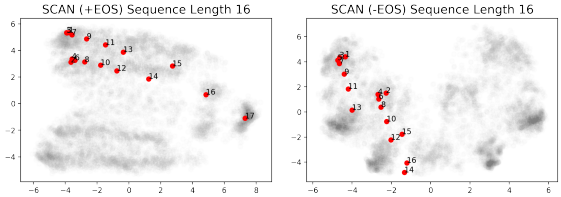


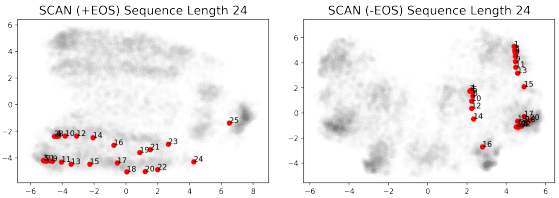
Figure 6: The top two principal components of hidden states from `+EOS` and `−EOS` LSTMs trained on the SCAN 22-token length split. Hidden states are colored by their position in the sequence. We can see that both models track length somewhat as it is required, however the `+EOS` model has an attractor cluster where all points end up while the `−EOS` model does not have such a cluster.



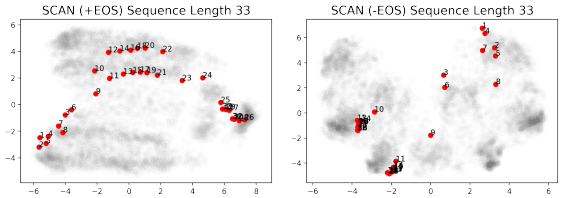
Input: run twice after jump right
Output: TURN_RIGHT JUMP RUN RUN



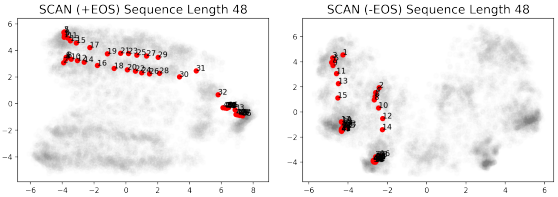
Input: run around right twice
Output: TURN_RIGHT RUN TURN_RIGHT RUN TURN_RIGHT RUN TURN_RIGHT RUN TURN_RIGHT RUN TURN_RIGHT RUN TURN_RIGHT RUN



Input: turn around left twice after run around left twice
Output: TURN_LEFT RUN TURN_LEFT RUN TURN_LEFT RUN TURN_LEFT RUN TURN_LEFT RUN TURN_LEFT RUN TURN_LEFT RUN TURN_LEFT TURN_LEFT TURN_LEFT TURN_LEFT TURN_LEFT TURN_LEFT TURN_LEFT TURN_LEFT



Input: walk opposite left thrice and run around right thrice
Output: TURN_LEFT TURN_LEFT WALK TURN_LEFT TURN_LEFT WALK TURN_LEFT TURN_LEFT WALK TURN_RIGHT RUN TURN_RIGHT RUN TURN_RIGHT RUN TURN_RIGHT RUN TURN_RIGHT RUN TURN_RIGHT RUN TURN_RIGHT RUN TURN_RIGHT RUN TURN_RIGHT RUN TURN_RIGHT RUN TURN_RIGHT RUN



Input: jump around right thrice and walk around right thrice
Output:TURN_RIGHT JUMP TURN_RIGHT JUMP TURN_RIGHT JUMP TURN_RIGHT JUMP TURN_RIGHT JUMP TURN_RIGHT JUMP TURN_RIGHT JUMP TURN_RIGHT JUMP TURN_RIGHT JUMP TURN_RIGHT WALK TURN_RIGHT WALK TURN_RIGHT WALK TURN_RIGHT WALK TURN_RIGHT WALK TURN_RIGHT WALK TURN_RIGHT WALK TURN_RIGHT WALK

Figure 7: Cherry-picked, but representative, examples of the paths of gold (i.e. not decoded) in-domain (length 4 and 16) and out-of-domain (length 24, 33, 48) examples through the hidden state space of LSTMs trained on the SCAN 22-token length split (top two principle components). Note that the EOS token is included in the +EOS plots, giving them one extra point. Interestingly, the hidden states from the OOD sequences arrive at the EOS attractor well after the maximum training length (22) (for example, the sequence of length 48 arrives at the attractor at the hidden state at position 33), suggesting that the +EOS does have some extrapolative abilities, but the EOS attractor is suppressing them.