Learning Bounded Context-Free-Grammar via LSTM and the Transformer: Difference and Explanations

Hui Shi [1], **Sicun Gao** [1], **Yuandong Tian** [2], **Xinyun Chen** [3], **Jishen Zhao**[1]

[1]University of California San Diego, [2]Facebook AI Research, [3]University of California, Berkeley
{hshi, jzhao, sicung}@ucsd.edu, yuandong@fb.com, xinyun.chen@berkeley.edu

Abstract

Long Short-Term Memory (LSTM) and Transformers are two popular neural architectures used for natural language processing tasks. Theoretical results show that both are Turing-complete and can represent any context-free language (CFL). In practice, it is often observed that Transformer models have better representation power than LSTM. But the reason is barely understood. We study such practical differences between LSTM and Transformer and propose an explanation based on their latent space decomposition patterns. To achieve this goal, we introduce an oracle training paradigm, which forces the decomposition of the latent representation of LSTM and the Transformer, and supervises with the transitions of the Pushdown Automaton (PDA) of the corresponding CFL. With the forced decomposition, we show that the performance upper bounds of LSTM and Transformer in learning CFL are close: both of them can simulate a stack and perform stack operation along with state transitions. However, the absence of forced decomposition leads to the failure of LSTM models to capture the stack and stack operations, while having a marginal impact on the Transformer model. Lastly, we connect the experiment on the prototypical PDA to a real-world parsing task to re-verify the conclusions. [1]

Introduction

The LSTM network has achieved great success in various natural language processing (NLP) tasks (Sutskever, Vinyals, and Le 2014; Wang et al. 2016), and in recent years, the Transformer network keeps breaking the record of state-of-the-art performances established by LSTM-based models in translation (Vaswani et al. 2017), question-answering (Devlin et al. 2018), and so on(Dehghani et al. 2018; Brown et al. 2020). Besides exploring the capacity boundary of the Transformer network, there is an increasing interest in investigating the representation power of the Transformer network and explaining its advantage over the LSTM models theoretically and empirically.

Existing analysis has proven that both LSTM (Siegelmann and Sontag 1995) and the Transformer network (Pérez, Marinković, and Barceló 2019) are Turing-Complete. However, much empirical evidence shows both models are far from perfect in imitating even simple Turing machines (Dehghani et al. 2018; Joulin and Mikolov 2015). Several explanations of the performance gap between practice and theory are 1) some theoretical proofs relies on infinite precision assumption while the precision in practical computations is limited (Weiss, Goldberg, and Yahav 2018; Korsky and Berwick 2019); 2) given fixed latent space dimension, according to the pigeonhole principle, as input sequence length goes towards infinity, there will be information that can not encode into the latent space or is forgotten (Hahn 2020).

Despite the soundness of the theoretical proofs and explanations, none of them can directly explain the phenomenons in practice: 1) given the same computation precision, Transformer has an advantage over LSTM model in many cases; 2) both Transformer and LSTM fails even when the input sequence is short and their latent space dimension is large.

In this work, we study the empirical representation power of the LSTM and the Transformer networks and investigate the origination of their difference. We compare the models via learning context-free languages. The reason to study CFLs other than regular languages or Turing machines is that the learning of CFLs provides most insights into NLP tasks where understanding the underlying hierarchy of the language is crucial. In the rest of the paper, we will first introduce an oracle training paradigm to predict the status of the PDA that accepts the CFL, and a regularizer to explicitly decompose the latent space of the LSTM and the Transformer such that the PDA state and the positions in the stack are encoded in distinct dimensions. Lastly, the experiment section exhibits the empirical results and leads us to the following conclusions:

- LSTM and Transformer have similar upper bound of empirical representation power in simulating PDAs.
- LSTM fails to factorize its latent space to encode the state and multiple elements of the stack without explicit supervision, which is the pivot to its compromised performance in real-world tasks. Meanwhile, the Transformer is marginally affected by absence of explicit decomposition regularization.
- Language recognition is not a reliable task to compare the empirical capacity of LSTM and Transformer since the results is sensitive to the setting of PDAs, the hyperparameters of the models, and the training methods.

[1]The code is available at https://github.com/shihui2010/learn_cfg_with_neural_network

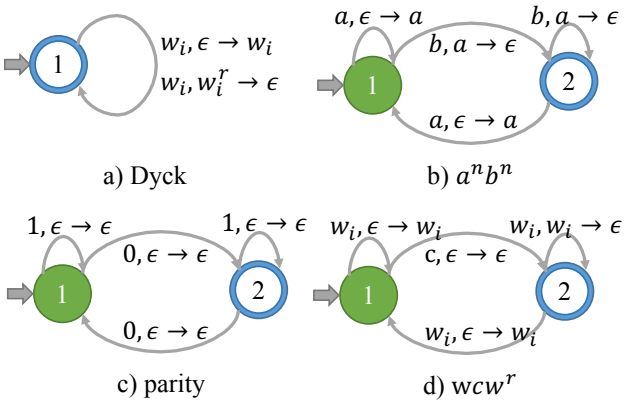


Figure 1: Graphical notations of PDAs for the CFLs.

Preliminaries and Definitions

Context-Free language

A context-free grammar $\mathcal{G}$ is defined by a collection of non-terminal symbols (*i.e.* variables), terminal symbols (*i.e.* the alphabet of $\mathcal{G}$), production rules and a start symbol E. The context-free language of $\mathcal{G}$ is the set of strings that can be derived from the start symbol by iteratively applying the production rules until there are no nonterminal symbols.

A Pushdown Automaton (PDA) is a state machine with a stack that reads an input symbol and the top element in a stack at each step and performs the state transition and some stack operations (push/pop). Formally, a PDA can be defined as a tuple of $< Q, \Sigma, S, \delta, q_0, I, F >$. Q is a finite set of states, and $q_0 \in Q$ is the initial state; Σ is the alphabet of the inputs, S is the stack symbols and $I \in S$ is the initial stack top symbol; $F \in Q$ is a set of accepting states. δ is a transition function $\delta(q \in Q, x \in (\Sigma \cup \{\epsilon\}), s \in S) \to q', s'$, where ϵ denotes an empty symbol and s denotes the top element in the stack S. The transition function implies the stack operation. For example, let $*$ be a wildcard for arbitrary state or symbol, $\delta(*, \epsilon, *)$ represents transition that consumes no input symbols[2]; $\delta(*, *, \epsilon) \to *, s'$ where $s' \neq \epsilon$ is a stack push operation, and $\delta(*, *, s) \to *, \epsilon$ where $s \neq \epsilon$ is a stack pop operation.

A PDA can be equivalently expressed by some CFG and vice versa (Schützenberger 1963; Chomsky 1962). In learning CFGs, the neural networks are expected to learn the equivalent PDAs instead of the production rules.

In this study, we are interested in bounded CFGs and PDAs where the recursion depth is finite and a stack with finite size should be adequate for process the CFLs. The reason is two-fold. First, we agree with the existing theoretical analysis that encoding an unbounded stack into finite space is the bottleneck for LSTM and Transformer, thus we focus on the investigation of their realistic representation power before they reach the theoretical upper bound (*e.g.* number of recursions $\to \infty$). Second, in natural languages and even programming language, the nesting of the production rules are very limited (*e.g.* < 100), so it's important to understand LSTM and Transformer's behavior under bounded CFGs.

Four canonical PDAs (shown in Figure 1) are introduced as follows:

Definition 0.1 (Dyck-(k, m))**.** The language of paired parentheses of k types up to m recursions. Dyck is the prototypical language to any context-free languages (Kozen 1997).

Definition 0.2 ($a^n b^n$)**.** The set of strings consisting of a and b, and every occurrence of n consecutive as is followed by exact n successive bs.

Definition 0.3 (Parity)**.** The binary strings that containing an even number of 0s. Parity can be expressed by a Deterministic Finite Machine (DFA). To formalize parity in PDA, we denote all parity transition function with no stack operation $\delta(*, *, \epsilon) \to *, \epsilon$.

Definition 0.4 ($(wcw^r)^n$-(k, m))**.** wcw^r-(k, m) generates strings that starts with a sub-string ω followed by a character c, then followed by the reverse of ω. The chars $w \in \omega$ comes from a vocabulary Ω ($c \notin \Omega$) of size k. The string ω contains no more than m chars. The $(wcw^r)^n$-(k, m) generates strings that contains a to n substrings from wcw^r-(k, m).

Long-short Term Memory network

Given the input sequence embeddings $\mathbf{X} = \{x_t\}_{t=1}^n$ and initial hidden state and cell state $(h_0, c_0) \in \mathbb{R}^d$, LSTM (Hochreiter and Schmidhuber 1997) produces the latent representation of the sequences $\{h_t\}_{t=1}^n$, where $h_t \in \mathbb{R}^d$, as follows:

$$
\begin{aligned}
f_t &= \sigma(\mathbf{W}_f h_{t-1} + \mathbf{U}_f x_t + b_f) \\
i_t &= \sigma(\mathbf{W}_i h_{t-1} + \mathbf{U}_i x_t + b_i) \\
o_t &= \sigma(\mathbf{W}_o h_{t-1} + \mathbf{U}_o x_t + b_o) \\
\tilde{c}_t &= tanh(\mathbf{W}_c h_{t-1} + \mathbf{U}_c x_t + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot c_t \\
h_t &= o_t \odot tanh(c_t)
\end{aligned}
\tag{1}
$$

Transformer network

The Transformer network (Vaswani et al. 2017) process the sequence via multi-head attention. Specifically, let the $att(Q, K, V)$ represent the scaled dot-product attention function over query $Q \in \mathbb{R}^{r_0 \times r_1}$, key $K \in \mathbb{R}^{r_0 \times r_1}$, and value $V \in \mathbb{R}^{r_0 \times d}$, defined as:

$$
att(Q, K, V) = softmax(\frac{QK^\top}{\sqrt{r_1}}V) \tag{2}
$$

Given sequence $\mathbf{X}$, the Transformer encodes the sequences via multi-head attention followed by a point-wise feed-forward network (denote as $FFN(\cdot)$).

$$
\begin{aligned}
h_t &= FFN([head_1; \cdot; head_k]); \\
head_i &= att(y\mathbf{W}_i^Q, y\mathbf{W}_i^K, y_t\mathbf{W}_i^V)
\end{aligned}
\tag{3}
$$

To distinguish the order of input symbols, the y_t is produced by summing a positional encoding with input encoding:

$$
\begin{aligned}
p_{t,2j} &= sin(t/10000^{2j/r_1}) \\
p_{t,2j+1} &= cos(t/10000^{2j/r_1}) \\
y_t &= x_t + p_t
\end{aligned}
\tag{4}
$$

[2]For instance, the *reduce* operation in shift-reduce parsing, as shown in Table 2

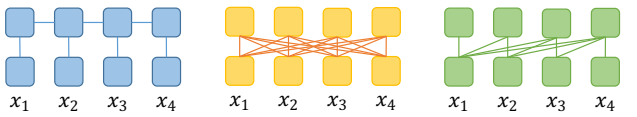


Figure 2: The information flow in LSTM, transformer encoder and decoder.

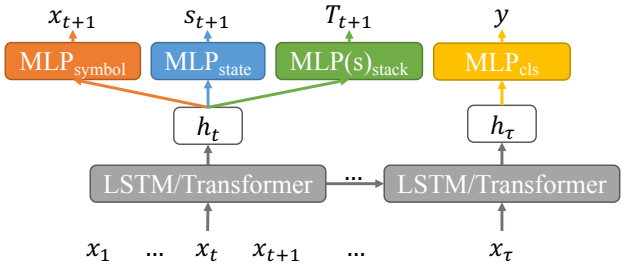


Figure 3: Oracle training.

We distinguish the *Transformer encoder* that allows attention between any pairs of input symbols and the *Transformer decoder* that allows multi-head attentions to compute only on past symbols. The comparison is illustrated in Figure 2.

Oracle Training

In this section, we introduce the oracle training method assuming complete PDA transition steps are exposed to the model to provide the densest supervision signal.

Formally, given symbol sequences $\{X^{(i)}\}_{i=1}^n$ accepted by a PDA, the *oracle* includes $\{X^{(i)}\}_{i=1}^n$, the states of DPA $\{S^{(i)}\}_{i=1}^n$ and the stack status $\{\mathcal{T}^{(i)}\}_{i=1}^n$ at each step while processing the symbol sequences. The oracle training forces the models to predict not only the next symbols as in the language model, but also the internal state and stack status. Hereinafter, for simplicity, we omit the superscript i that denotes i-th sample. Let $X = \{x_t\}_{t=1}^\tau$, $S = \{s_t\}_{t=1}^\tau$, and $\mathcal{T} = \{T_t\}_{t=1}^\tau$, and the $T_t[j]$ being the j-th item in the stack at step t.

Figure 3 shows the generic architecture for oracle training. Several multi-layer perceptron (MLP) networks are employed to independently predict the symbol, state, and stack from the latent representation h_t. For stack status, a non-full stack is padded with the empty token ϵ. Therefore, the models always predict a constant number of symbols for the stack and predict ϵs in the correct positions to indicate a non-full stack. We also include a language recognition task in which the model predicts if the sequences are accepted by the PDA. The language recognition task is widely used in arguing the inability of LSTM and Transformer in recognizing CFLs. Though from the PDA's perspective, the recognition is equivalent to the task of learning the transition, while in the experiment, we show both models benefit greatly from dense supervision in the oracle training, compared to the sparse supervision in the language recognition.

We also introduce two vital model configurations: **forced decomposition** (Figure 4) and **latent decomposition** (Figure 5). In forced decomposition, the latent representation vector h_t is split into $m+1$ segments, where m is the maximum stack size, to predict the PDA state and m elements in the stack separately. Since each position in the stack shares the same set of stack symbols, we let the stack predictors MLP_{stack} share parameters. In contrast, the latent decomposition uses whole h_t but individual MLPs for prediction, and the m stack predictors are independently trained. In both configurations, the next symbol is predicted based on complete h_t because the valid symbol for the next step depends on both the current state and stack.

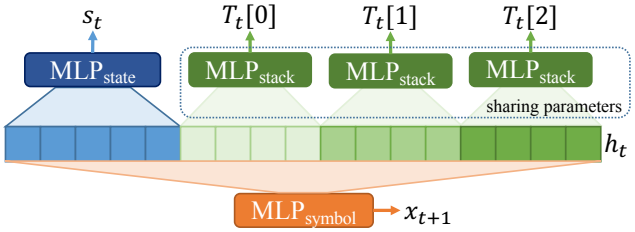


Figure 4: Forced decomposition of h_t.

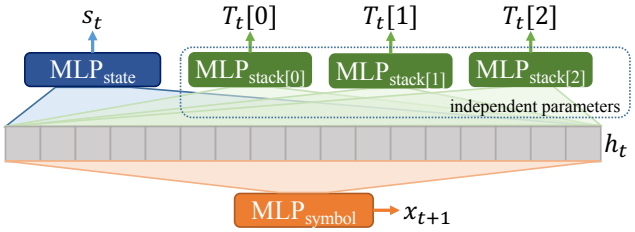


Figure 5: Latent decomposition of h_t (with stack size of 3).

Predictions of symbol, state, and stack are all trained with cross-entropy loss, and the three-part losses are summed [3] :

$$\mathcal{L}_{oracle} = \mathcal{L}_{symbol} + \mathcal{L}_{state} + \mathcal{L}_{stack} \quad (5)$$

Experiments

Canonical PDAs

In this section, we evaluate the representation power of LSTM and Transformer by simulating canonical PDAs.

Data generation. For PDAs introduced and their hyperparameters $\{k, m, n\}$ when applicable, we generate the 50k sequences for training and another 50k for testing except for wcw^r(n=2, m=2, k=*) which enumerate all accepted sequences. For each sequence, the ground-truth PDA annotates the sequence and produces the state and stack labels for oracle training. Meanwhile, a corrupted sequence that is not accepted by the PDA is generated for the classification subtask. For language modelling, we compute the valid symbols for each step t given sequence $x_1, \cdots, x_{t-1}$ and denote this validity over alphabet as *LM mask*.

[3] An option is to assign weights to each term on the right hand side, our additional experiment in the Appendix shows that the choice of loss weights does not influence the main conclusion in the experiment.

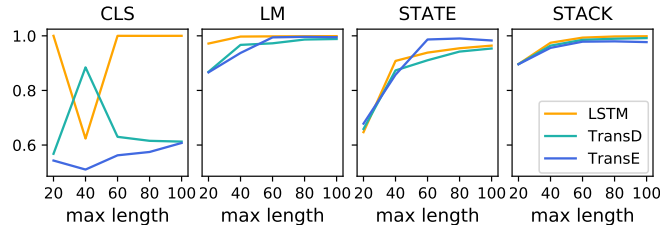


Figure 6: Performance on $a^n b^n$

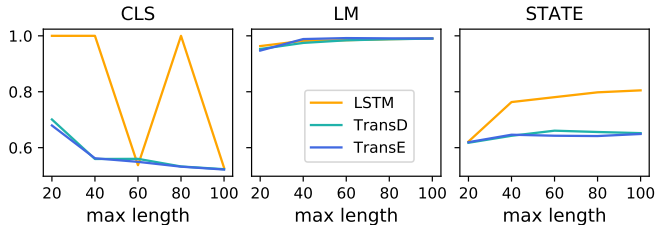


Figure 7: Performance on parity

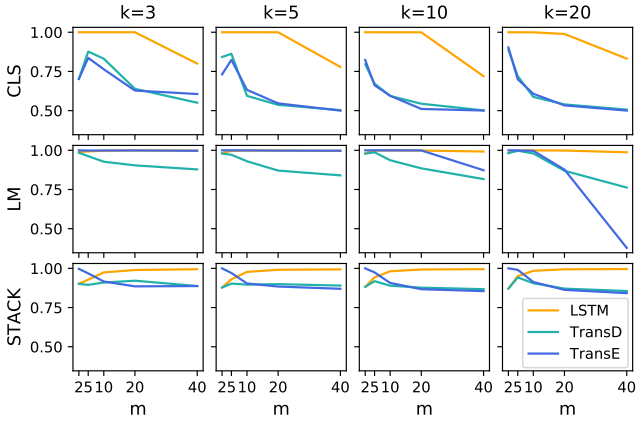


Figure 8: Performance on Dyck.

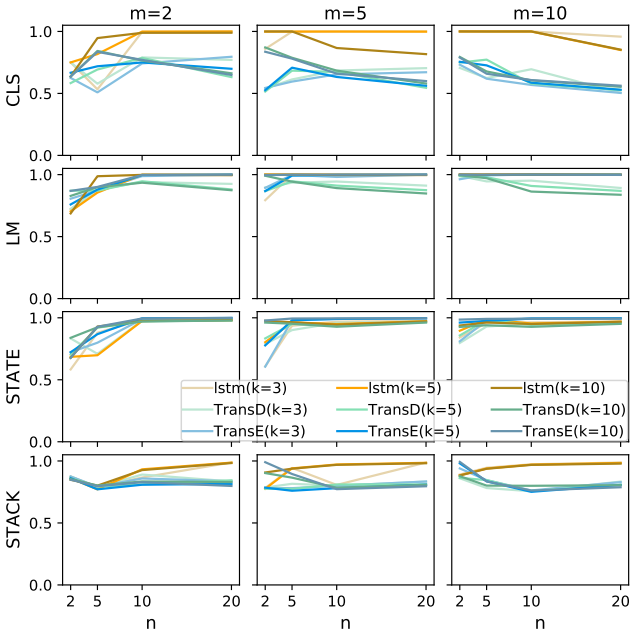


Figure 9: Performance on wcw^r.

Model configurations. For each PDA task, we set the hidden size in the LSTM model and the latent dimension in the Transformer model to be $\alpha * (|Q| + m * |S|)$, where α is the scale factor, $|Q|$ is the size of states, m is the maximum recursion (*i.e.* maximum stack size), and $|S|$ is the size of stack symbols[4]. We always let $\alpha \geq 1$ such that the latent vector h_t will always have enough dimensions to encode both the state and stack. For the Transformer encoder and decoder, the number of attention heads is 8. For all models, unless specified otherwise, the number of layers is 1 and $\alpha = 1$. We set all MLP modules to be a two-layer feed-forward network with sigmoid as an activation function. The hidden dimension of the MLPs is twice of their input feature dimension. For both models, we use an embedding layer to encode input symbols to $\mathbb{R}^{2|\Sigma|}$. For the Transformer model, the filter size for the position-wise feed-forward network is 32, and we apply a dropout layer with a rate of 0.1.

Training. Models are trained with Adam optimizer with a learning rate of 0.001 on the AWS platform. The models are trained for 200 epochs or up to convergence. We introduce *two-phase training* for language recognition tasks, *i.e.* classify whether the sequences are accepted by the PDA. In phase 0, the models are initialized and solely trained by a classification task, while in phase 1, the models are retrained on classification tasks after being trained with oracle training.

Metrics. 1) Classification accuracy: portion of the sequences that are correctly accepted/rejected. 2) LM accuracy: percentage of predicted symbols that are valid according to the LM mask. 3) State accuracy: accuracy in predicting the current PDA state shown only the sequence of symbols. 4) Stack accuracy: accuracy in predicting current stack status, stack symbols (including empty symbol), and their position in the stack shown only the sequence of symbols.

Overall results. Figure 6, 7, 8, 9 shows the LSTM and Transformer performance over multiple configuration of PDAs [5]. There are few observations from the results: 1) LSTM has higher accuracy in learning PDAs, as it generally achieves higher accuracy in both state and stack predictions, especially shown in Figure 7, 9. The results shows that when CFGs are bounded, LSTM does not need external memory to simulate the stack. 2) For predicting the state, Transformer decoder behaves in a similar way to LSTM, as shown in state accuracy in Figure 6, 9. In $a^n b^n$and $(wcw^r)^n$, the Transformer encoder slightly outperforms LSTM and Transformer decoder. This indicates that though past information should be adequate to determine the current status of PDA, it's still benefits the neural models to foresee the future sequences. 3) Both language recognition task and language modeling

[4]For PDAs introduced, we set $S = \Sigma \bigcup \{I, \epsilon\}$

[5]These figs shows phase 0 classification accuracy

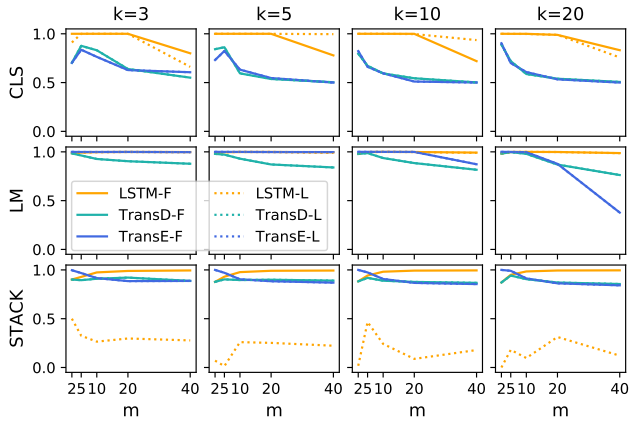


Figure 10: Performance on Dyck with forced (solid lines) and latent (dotted lines) decomposition.

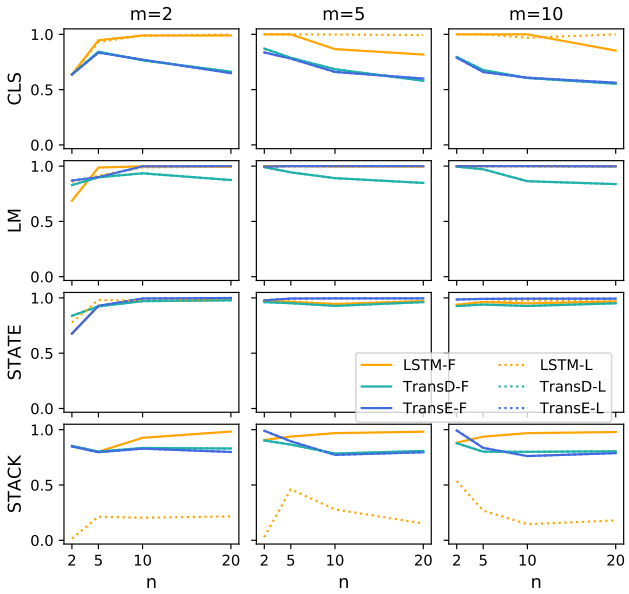
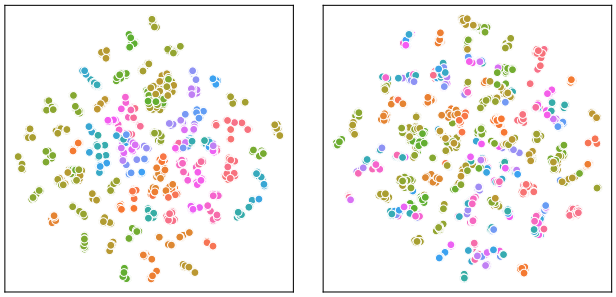


Figure 11: Performance on wcw^r with forced (solid lines) and latent (dotted lines) decomposition ($k = 10$).



a) Forced b) Latent

Figure 12: t-SNE analysis on LSTM hidden states trained on Dyck (k=3,m=5).

solely should not be used to examine the capability of neural models in learning CFG, since they do not fully reflect the capability of models to learn the internal dynamics in state transition and stack operation (Figure 7), and reversely learning the precise PDA does not necessarily lead to perfection in language recognition and language modeling (Figure 8, 9).

Decomposition Hardship of LSTM. Though LSTM and Transformer show comparable representation power under forced decomposition setting, the disadvantage of LSTM in the latent decomposition training is significant, as can be viewed from Figure 10, 11, meanwhile, Transformer models are roughly as good as they are trained with forced decomposition. The failure of LSTM in stack prediction is key to its disadvantage in many tasks (Devlin et al. 2018). Figure 12 shows the two-component t-SNE (Van der Maaten and Hinton 2008) results of the LSTM hidden states h_t. The different colors represent distinct stack status in the oracle. As shown, the hidden states in forced decomposition tend to separately encode different stack status, while in the latent decomposition, there are much more clusters containing multiple colors, which causes the failure of stack predictor to correctly predict the stack status, and also prevents LSTM itself to learn the correct stack operations.

Scaling factor. Generally, a larger number of parameters brings higher representation power. We verified the conclusion that LSTM and Transformer have similar representation power of learning CFG on larger models. Figure 13 shows the performance of four-layer LSTM and Transformer model with scaling factor $\alpha = 4$. Viewing from the stack prediction accuracy, the conclusion still holds. We also examined the conclusion on factorization on larger scale models shown in Figure 14. The full results of the larger models can be found in the Appendix.

Why language recognition accuracy is not a good indicator. We detail the four-tier reasons for our earlier conclusion that language recognition accuracy is unfair in judging the models' capability of learning PDA. Firstly, we explain the observations from Figure 8, 9 that perfection in state and stack prediction does not lead to accurate recognition: the language recognition requires a higher level of reasoning than just learning a set of PDA transition functions. A rejected sequence for PDA might due to the current symbol not accepted given the state and stack top, popping an empty stack, or exceeding the maximum recursion. Thus the classification decision boundary might be inseparable for an MLP without special design. Furthermore, for any models, the error message might occur at random steps, and the models have to preserve and pass the message to the last step for the classification prediction in the general training paradigm for language recognition. This is mainly why the classification accuracy of LSTM declines much abruptly than of the Transformers since Transformer can pass information between any pairs of inputs, thus the comparison of LSTM and Transformer will be sensitive to sequence lengths. Besides sequence length, the model size also influences the recognition

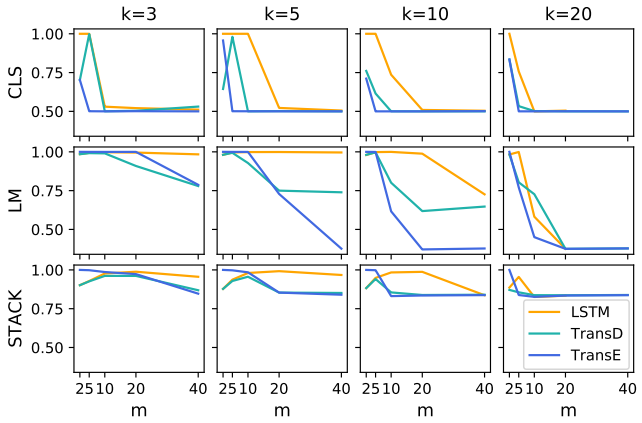


Figure 13: Performance of four-layer models on Dyck with $\alpha = 4$

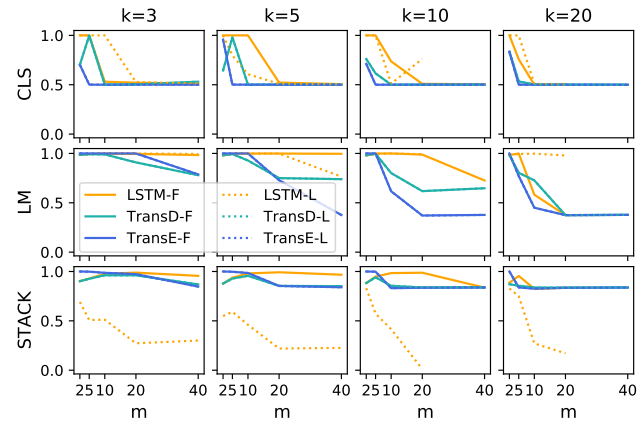


Figure 14: Four-layer models ($\alpha = 4$) on Dyck with forced (solid lines) and latent (dotted lines) decomposition.

accuracy differently. Comparing the classification accuracy of Figure 8 and Figure 13, the relative advantage of two models are reversed when models scale up. Lastly, Transformer models may recognize language in some manners that are dissimilar to PDAs. To prove this, we show the classification accuracy of phase 0 and phase 1 in Figure 15, 16, 17. LSTM generally improves after oracle training (phase 1), especially in Dyck (Figure 17), while the Transformer models suffers from oracle training (Figure 15-16).

Shift-Reduce Parsing

Dataset. SCAN (Lake and Baroni 2018) is a semantic parsing dataset consisting of commands in natural language and sequences of actions to execute the commands. Tab. 1 shows the generation rules producing SCAN commands. The dataset contains 16728 training samples and 16728 test samples. In our experiment, instead of parsing to the sequence of actions, we parse the linguistic command according to its CFG production rules (Tab. 1) using shift-reduce parsing. Since there are production rules in the CFG that map a single nonterminal variable to another one, the corresponding PDA is nondeterministic. To facilitate the training, we insert a special token <*reduce*> to make the process deterministic. Tab. 2 illustrates an example of the annotation of the sequence with stack status and the padding process to insert <*reduce*> tokens for reduction operations that do not consume symbol from the input sequence. The equivalent PDA of SCAN's CFG has only one state q_0. Alphabet Σ covers English words in the linguistic commands and the <*reduce*> token, and the stack symbols are $\{C, CP, S, V, VP, D, U\}$.

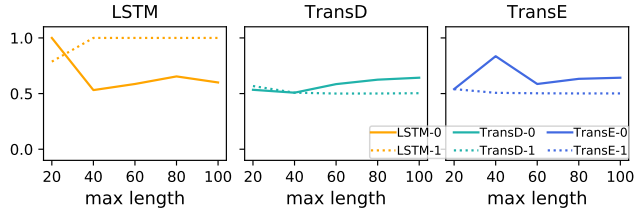


Figure 15: Two-phase classification accuracy on $a^n b^n$.

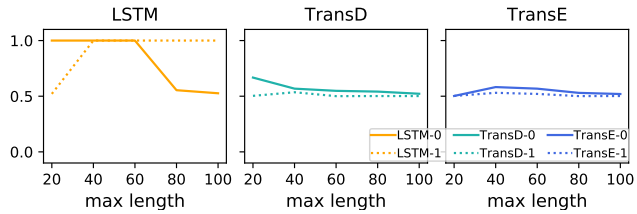


Figure 16: Two-phase classification accuracy on parity.

Models and metrics. The model configurations are the same as in learning the canonical PDAs. We compute perplexity to evaluate language modeling and compute the accuracy of the stack prediction as the parsing accuracy.

Results. Tab. 3 sums up the results of LSTM and Transformer on SCAN dataset, which is consistent with the observations in Sec. : LSTM and Transformer decoder perform similarly in language modeling and parsing when decomposition is forced, and the parsing accuracy of LSTM suffers heavily from latent decomposition setting. The Transformer encoder can see the whole sequence at each step, so it achieves an almost lower bound of perplexity and has the highest parsing accuracy.

Related Works

There are many theoretical analyses on the representation power of LSTM and the Transformer and their comparisons that motivate this work to re-examine their representation power from an empirical aspect. Siegelmann and Sontag (1995) firstly established the theory that given infinite precision and adequate number of hidden units RNNs are Turing-Complete, and Hölldobler, Kalinke, and Lehmann (1997) has designed an one-unit vanilla RNN counter for recognizing counter languages (*e.g.* $a^n b^n$ and $a^n b^n c^n$) with finite range of n. Recently, Hewitt et al. (2020) proposed the construction of RNN that performs stack operations as in PDA and encodes PDA stack within hidden states of RNN without external memory. Pérez, Marinković, and Barceló (2019)

C	:=	CP S | CP V | S
CP	:=	S and | S after
S	:=	V | V twice | V thrice
V	:=	VP D| VP left | VP right | D
VP	:=	D opposite | D around
D	:=	U | U left | U right
U	:=	walk | look | run | jump | turn

Table 1: Domain Specific Language for SCAN linguistic commands.

step	stack	unconsumed symbols	transition δ
0	[]	jump left and turn opposite left	–
1	[U]	left and turn opposite left	$\delta(left, \epsilon) \rightarrow U$
2	[D]	and turn opposite left	$\delta(and, U) \rightarrow D$
3	[V]	and turn opposite left	$\delta(\epsilon, D) \rightarrow V$
4	[S]	and turn opposite left	$\delta(\epsilon, V) \rightarrow S$
5	[CP]	turn opposite left	$\delta(and, S) \rightarrow CP$
6	[CP, D]	opposite left	$\delta(turn, \epsilon) \rightarrow U$
7	[CP, VP]	left	$\delta(opposite, D) \rightarrow VP$
8	[CP, V]		$\delta(left, VP) \rightarrow V$
9	[C]		$\delta(\epsilon, [CP, V]) \rightarrow C$
Padded Sequence			
[jump, left, and, <reduce>, <reduce>, turn, opposite, left, <reduce>]			

Table 2: Shift-reduce parsing of SCAN commands.

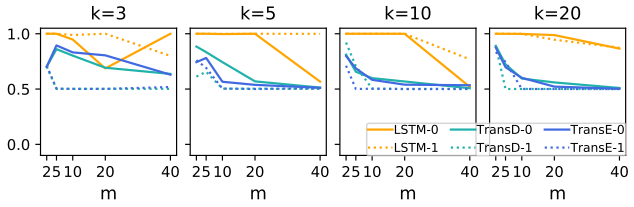


Figure 17: Two-phase classification accuracy on Dyck.

Model	Perplexity	Accuracy
LSTM($\alpha = 1$, Forced)	2.705	91.30
LSTM($\alpha = 1$, Latent)	2.705	16.61
LSTM($\alpha = 4$, Forced)	2.706	91.30
LSTM($\alpha = 4$, Latent)	2.708	35.00
Transformer(D, $\alpha = 1$, Forced)	2.710	91.02
Transformer(D, $\alpha = 1$, Latent)	2.713	76.61
Transformer(D, $\alpha = 4$, Forced)	2.708	91.30
Transformer(D, $\alpha = 4$, Latent)	2.710	90.56
Transformer(E, $\alpha = 1$, Forced)	1.020	99.84
Transformer(E, $\alpha = 1$, Latent)	1.014	72.39
Transformer(E, $\alpha = 4$, Forced)	1.002	99.99
Transformer(E, $\alpha = 4$, Latent)	1.001	99.29

Table 3: Perplexity and parsing accuracy on SCAN. **E** denotes Transformer encoder and **D** denotes decoder.

proofs that with arbitrary precision, the Transformer network could simulate the single execution step for Turing machine, then by induction the Transformer is Turing-Complete. The majority of theoretical analysis emphasis that limited computation precision may break the proofs and compromise the performance in practice. Nevertheless, the models are not supervised with either step-by-step execution of the Turing Machine or the actual counters, which might be the crux to the failures of both models in reality.

This work also closely relates to previous attempts to connecting LSTM and transformer model with a specific type of languages, *e.g.* languages from Chomsky's hierarchy (Chomsky 1956) and counter languages. 1) For regular languages (representable by DFAs), Michalenko et al. (2019) shows the empirical ability of LSTM to represent DFAs and Rabusseau, Li, and Precup (2019) has proposed a construction method of RNNs from a weighted DFA. 2) For context-free languages (representable by PDAs), Sennhauser and Berwick (2018) observed that CFGs are hardly learnable by LSTM models. on the other hand, Bhattamishra, Ahuja, and Goyal (2020b) showed LSTM could learn CFGs with bounded recursion depth but the performance will be limited for infinite recursion. 3) For counter languages, the Transformer network (Bhattamishra, Ahuja, and Goyal 2020a) and LSTM Suzgun et al. (2019a) have been trained to predict the outputs of a dynamic counter. However, their results disagree on the capability of LSTM in representing DYCK languages. In our work, we focus on bounded CFG since the capacity of learning regular languages is widely agreed upon while there are disputes on the CFG level.

Finally, observing the defects of both LSTM and Transformer in learning CFG and algorithmic tasks, many works propose to use external memory to enhance the LSTM model (Joulin and Mikolov 2015; Das, Giles, and Sun 1992; Suzgun et al. 2019b), introduce recurrence in the Transformer network (Dehghani et al. 2018), and design specialized architectures (Graves, Wayne, and Danihelka 2014; Hao et al. 2018; Sukhbaatar et al. 2015; Stogin et al. 2020). Though it's commonly believed that LSTM with finite memory, *i.e.* hidden states, can not handle CFGs which requires infinite stack spaces, we investigate the capacity of LSTM and transformer in bounded CFGs that requires finite-size stack and conclude that finite memory is not the bottleneck of LSTM capacity in learning CFGs.

Conclusion

We illustrate that only the state and stack prediction accuracy trained with dense supervision and explicit decomposition regularizer are the fair and stable metric to compare the empirical representation power of LSTM and the Transformer network. Then we conclude that both LSTM and Transformer network can simulate context-free languages with bounded recursion with a similar representation power, and unveiled the disadvantage of LSTM model in practice is from its inability to decompose the latent representation space.

References

Bhattamishra, S.; Ahuja, K.; and Goyal, N. 2020a. On the Ability of Self-Attention Networks to Recognize Counter Languages. *arXiv preprint arXiv:2009.11264*.

Bhattamishra, S.; Ahuja, K.; and Goyal, N. 2020b. On the Practical Ability of Recurrent Neural Networks to Recognize Hierarchical Languages. *arXiv preprint arXiv:2011.03965*.

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Chomsky, N. 1956. Three models for the description of language. *IRE Transactions on information theory*, 2(3): 113–124.

Chomsky, N. 1962. Context-free grammars and pushdown storage. Vol. 65 of Quarterly Progress Report.

Das, S.; Giles, C. L.; and Sun, G.-Z. 1992. Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *Proceedings of The Fourteenth Annual Conference of Cognitive Science Society. Indiana University*, 14. Citeseer.

Dehghani, M.; Gouws, S.; Vinyals, O.; Uszkoreit, J.; and Kaiser, Ł. 2018. Universal transformers. *arXiv preprint arXiv:1807.03819*.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Graves, A.; Wayne, G.; and Danihelka, I. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Hahn, M. 2020. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8: 156–171.

Hao, Y.; Merrill, W.; Angluin, D.; Frank, R.; Amsel, N.; Benz, A.; and Mendelsohn, S. 2018. Context-free transductions with neural stacks. *arXiv preprint arXiv:1809.02836*.

Hewitt, J.; Hahn, M.; Ganguli, S.; Liang, P.; and Manning, C. D. 2020. RNNs can generate bounded hierarchical languages with optimal memory. *arXiv preprint arXiv:2010.07515*.

Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural computation*, 9(8): 1735–1780.

Hölldobler, S.; Kalinke, Y.; and Lehmann, H. 1997. Designing a counter: Another case study of dynamics and activation landscapes in recurrent networks. In *Annual Conference on Artificial Intelligence*, 313–324. Springer.

Joulin, A.; and Mikolov, T. 2015. Inferring algorithmic patterns with stack-augmented recurrent nets. *arXiv preprint arXiv:1503.01007*.

Kendall, A.; Gal, Y.; and Cipolla, R. 2018. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 7482–7491.

Korsky, S. A.; and Berwick, R. C. 2019. On the computational power of rnns. *arXiv preprint arXiv:1906.06349*.

Kozen, D. C. 1997. The Chomsky—Schützenberger Theorem. In *Automata and Computability*, 198–200. Springer.

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

Michalenko, J. J.; Shah, A.; Verma, A.; Baraniuk, R. G.; Chaudhuri, S.; and Patel, A. B. 2019. Representing formal languages: A comparison between finite automata and recurrent neural networks. *arXiv preprint arXiv:1902.10297*.

Pérez, J.; Marinković, J.; and Barceló, P. 2019. On the turing completeness of modern neural network architectures. *arXiv preprint arXiv:1901.03429*.

Rabusseau, G.; Li, T.; and Precup, D. 2019. Connecting weighted automata and recurrent neural networks through spectral learning. In *The 22nd International Conference on Artificial Intelligence and Statistics*, 1630–1639. PMLR.

Schützenberger, M. P. 1963. On context-free languages and push-down automata. *Information and control*, 6(3): 246–264.

Sennhauser, L.; and Berwick, R. C. 2018. Evaluating the ability of LSTMs to learn context-free grammars. *arXiv preprint arXiv:1811.02611*.

Siegelmann, H. T.; and Sontag, E. D. 1995. On the computational power of neural nets. *Journal of computer and system sciences*, 50(1): 132–150.

Stogin, D.; Mali, A.; Giles, D.; et al. 2020. Provably stable interpretable encodings of context free grammars in rnns with a differentiable stack. *arXiv preprint arXiv:2006.03651*.

Sukhbaatar, S.; Szlam, A.; Weston, J.; and Fergus, R. 2015. End-to-end memory networks. *arXiv preprint arXiv:1503.08895*.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*.

Suzgun, M.; Gehrmann, S.; Belinkov, Y.; and Shieber, S. M. 2019a. LSTM networks can perform dynamic counting. *arXiv preprint arXiv:1906.03648*.

Suzgun, M.; Gehrmann, S.; Belinkov, Y.; and Shieber, S. M. 2019b. Memory-augmented recurrent neural networks can learn generalized Dyck languages. *arXiv preprint arXiv:1911.03329*.

Van der Maaten, L.; and Hinton, G. 2008. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11).

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Wang, Y.; Huang, M.; Zhu, X.; and Zhao, L. 2016. Attention-based LSTM for aspect-level sentiment classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, 606–615.

Weiss, G.; Goldberg, Y.; and Yahav, E. 2018. On the practical computational power of finite precision RNNs for language recognition. *arXiv preprint arXiv:1805.04908*.

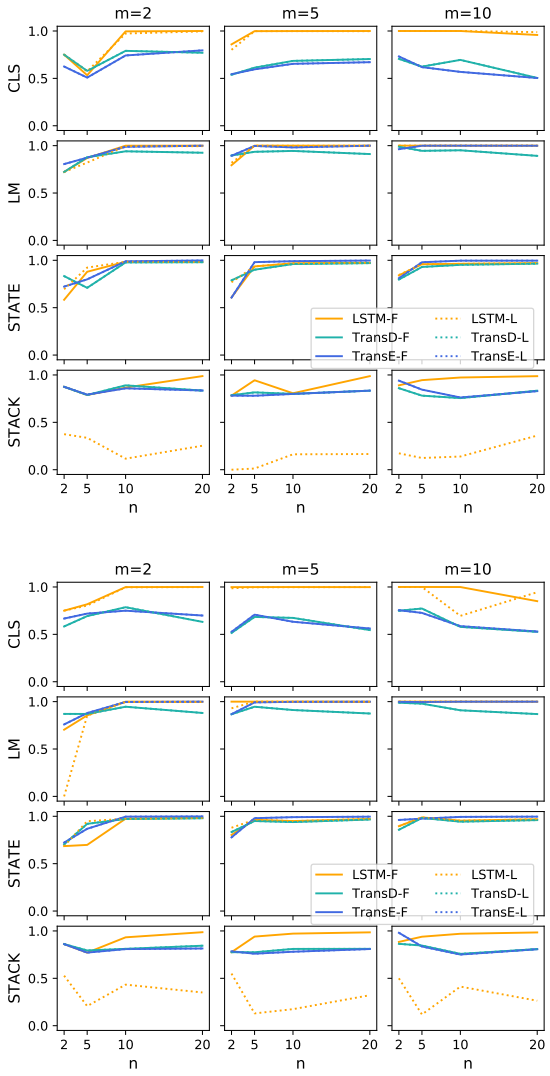


Figure 18: Performance on $(wcw^r)^n$. Upper figure shows results with $k = 3$, and the lower shows results with $k = 5$.

Additional Factorization Results

Figure 18, 19, 23 show additional results for factorization. Unless specified, the model are configured with a single layer and $\alpha = 1$. All the results support the conclusion that LSTM without forced factorization hardly learns to simulate the PDA stack properly.

Sensitivity to Loss Weights

Loss function: In Equation (5), the weights of $\mathcal{L}_{symbol}$, $\mathcal{L}_{state}$, $\mathcal{L}_{stack}$ are set to 1. To confirm the conclusion is general without dependent on the choice of weight values, we examine the model performance using the learnable loss weights (Kendall, Gal, and Cipolla 2018), and the oracle

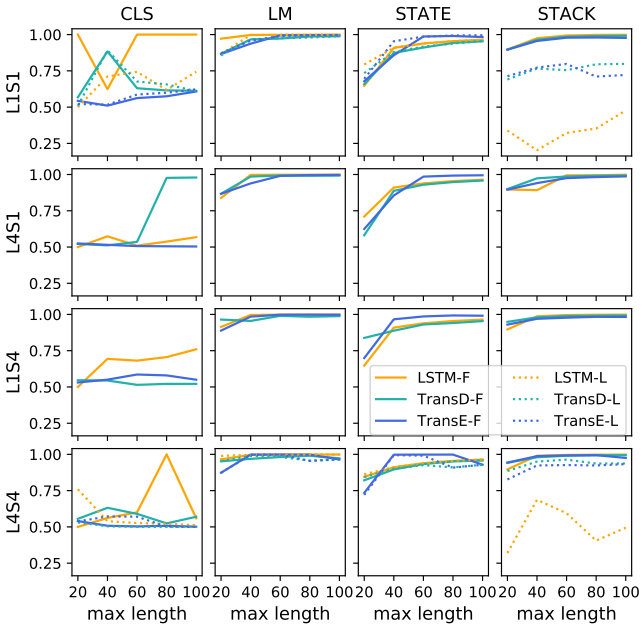


Figure 19: Performance on $a^n b^n$. Y-label shows number of layers and scaling factor α. For example, *L1S4* represents single-layer model with $\alpha = 4$.

training loss has the form:

$$\mathcal{L}_{oracle} = \frac{1}{2\sigma_1^2}\mathcal{L}_{symbol} + \frac{1}{2\sigma_2^2}\mathcal{L}_{state} + \frac{1}{2\sigma_3^2}\mathcal{L}_{stack} + log(\sigma_1\sigma_2\sigma_3) \tag{6}$$

where $\sigma_1, \sigma_2, \sigma_3$ are trainable parameters.

Dataset and training: We used Dyck-(5,*) datasets ($m = 2, 5, 10, 20$). For each model on each dataset, the training is repeated five times.

Results: Figure 20 compares the model performance with fixed or learned weights and illustrates that neither the performance nor the hardship in factorization is sensitive to the choice of loss weights.

Deeper models

In Figure 13, 14, we show that additional layers and extra hidden size can not remedy the hardship in decomposition. Figure 19, 23 provide further evidence. From Figure 19, we notice that the difference between forced and latent factorization is alleviated in Transformer models as the number of layer increases and α grows[6]. However, the gap remains huge for the LSTM model. Besides, the decrease of classification and language model accuracy is generally observed when scaling up the models, which indicates the training difficulty introduced by the deeper and wider model overwhelms the benefit of increased model capacity.

[6]The ablation on factorization is not conducted for L4S1 and L1S4 for any PDA. Also, since the factorization makes most significant difference on learning stack and Parity requires no stack, the comparison of factorization is not made on Parity.

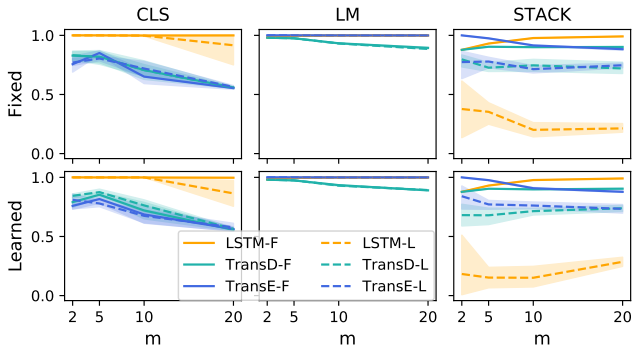


Figure 20: Model performance on Dyck-(5,*). First row shows results with Equation (5), and second row shows results with Equation (6). Lines indicates the average accuracy over 5 runs, and the shadows illustrate the range.

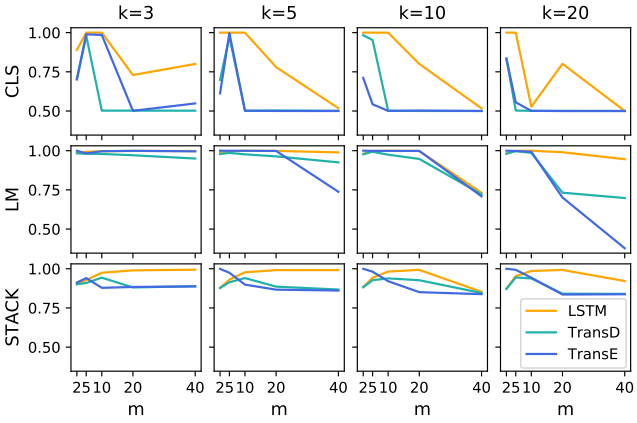


Figure 21: Performance on Dyck ($\alpha = 1$, four layers)

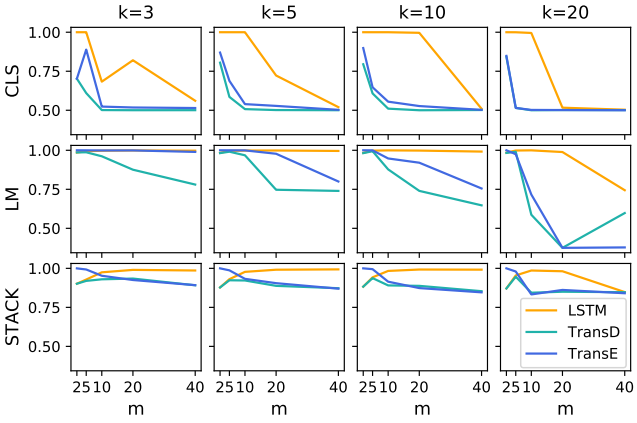


Figure 22: Performance on Dyck ($\alpha = 4$, single layer)

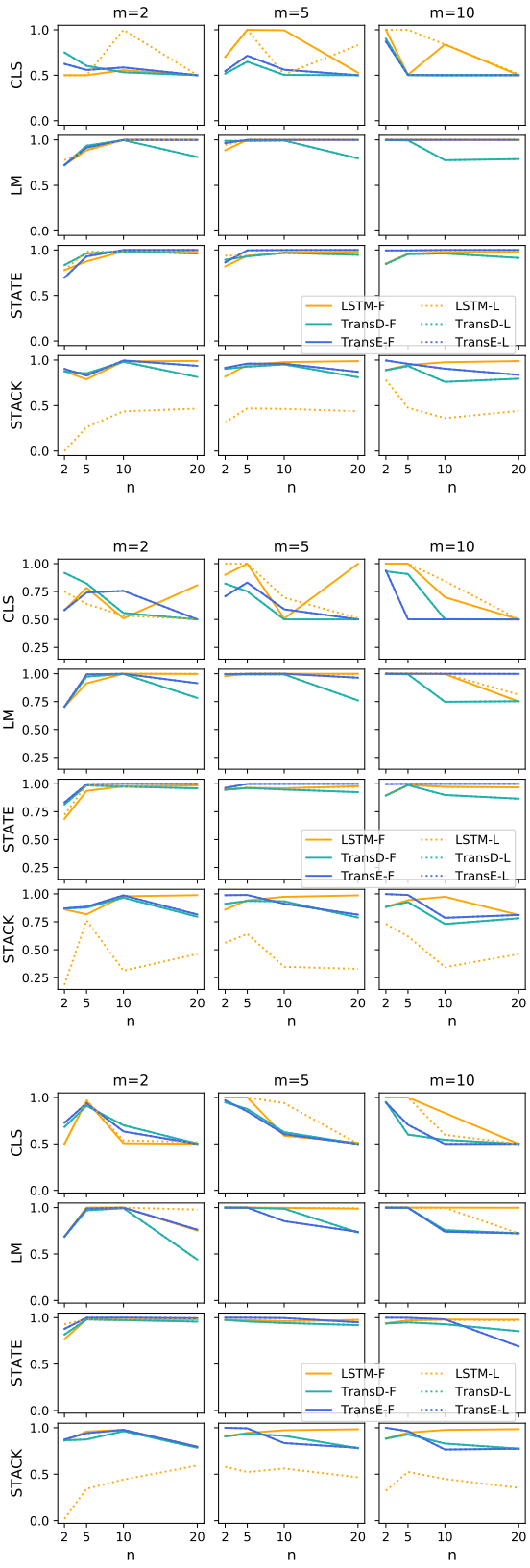


Figure 23: 4-layer model with $\alpha = 4$ on $(wcw^r)^n$. The upper figure shows results on $k = 3$, the middle shows $k = 5$, and the bottom shows $k = 10$.