Self-Attention Networks Can Process Bounded Hierarchical Languages

Shunyu Yao[†] **Binghui Peng**[‡] **Christos Papadimitriou**[‡] **Karthik Narasimhan**[†]
[†]Princeton University [‡]Columbia University
{shunyuy, karthikn}@princeton.edu
{bp2601, christos}@columbia.edu

Abstract

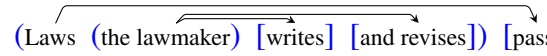
Despite their impressive performance in NLP, self-attention networks were recently proved to be limited for processing formal languages with hierarchical structure, such as Dyck_k, the language consisting of well-nested parentheses of k types. This suggested that natural language can be approximated well with models that are too weak for formal languages, or that the role of hierarchy and recursion in natural language might be limited. We qualify this implication by proving that self-attention networks can process $\mathsf{Dyck}_{k,D}$, the subset of Dyck_k with depth bounded by D, which arguably better captures the bounded hierarchical structure of natural language. Specifically, we construct a hard-attention network with $D+1$ layers and $O(\log k)$ memory size (per token per layer) that recognizes $\mathsf{Dyck}_{k,D}$, and a soft-attention network with two layers and $O(\log k)$ memory size that generates $\mathsf{Dyck}_{k,D}$. Experiments show that self-attention networks trained on $\mathsf{Dyck}_{k,D}$ generalize to longer inputs with near-perfect accuracy, and also verify the theoretical memory advantage of self-attention networks over recurrent networks.[1]

1 Introduction

Transformers (Vaswani et al., 2017) are now the undisputed champions across several benchmark leaderboards in NLP. The major innovation of this architecture, *self-attention*, processes input tokens in a distributed way, enabling efficient parallel computation as well as long-range dependency modelling. The empirical success of self-attention in NLP has led to a growing interest in studying its properties, with an eye towards a better understanding of the nature and characteristics of natural language (Tran et al., 2018; Papadimitriou and Jurafsky, 2020).

In particular, it was recently shown that self-attention networks *cannot* process various kinds of formal languages (Hahn, 2020; Bhattamishra et al., 2020a), among which particularly notable is Dyck_k, the language of well-balanced brackets of k types. By the Chomsky-Schützenberger Theorem (Chomsky and Schützenberger, 1959), any context-free language can be obtained from a Dyck_k language through intersections with regular languages and homomorphisms. In other words, this simple language contains the essence of all context-free languages, i.e. hierarchical structure, center embedding, and recursion – features which have been long claimed to be at the foundation of human language syntax (Chomsky, 1956).

Consider for example the long-range and nested dependencies in English subject-verb agreement:



The sentence structure is captured by Dyck_2 string (()[][])[]. Given the state-of-the-art performance of Transformers in parsing natural language (Zhang et al., 2020; He and Choi, 2019), the Dyck_k blind spot seems very suggestive. If the world's best NLP models cannot deal with this simple language — generated by a grammar with $k+2$ rules and recognized by a single-state pushdown automaton — does this not mean that the role of hierarchy and recursion in natural language must be limited? This question has of course, been extensively debated by linguists on the basis of both theoretical and psycholinguistic evidence (Hauser et al., 2002; Frank et al., 2012; Nelson et al., 2017; Brennan and Hale, 2019; Frank and Christiansen, 2018).

So, what can self-attention networks tell us about natural language and recursion? Here we provide a new twist to this question by considering $\mathsf{Dyck}_{k,D}$, the subset of Dyck_k with nesting depth at most D, and show that Transformers can process

[1]Code is available at https://github.com/princeton-nlp/dyck-transformer.

Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, pages 3770–3785
August 1–6, 2021.

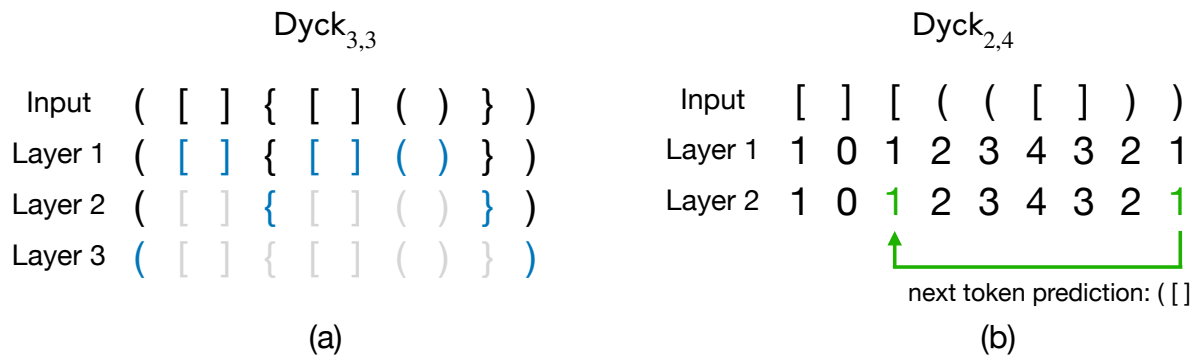


Figure 1: Illustrations of our self-attention network constructions to recognize and generate $\text{Dyck}_{k,D}$. In construction (a), at each layer, the innermost brackets attend to their matching brackets and "cancel" each other, yielding "shallower" spans for successive layers to process. In construction (b), the first layer computes the depth of each token by attending to all previous tokens, while the second layer uses depth information to find the most recent unclosed open bractket in the history.

it. $\text{Dyck}_{k,D}$ models bounded (or finite) recursion, thus captures the hierarchical structure of human language much more realistically. For example, center-embedding depth of natural language sentences is known to rarely exceed three (Karlsson, 2007; Jin et al., 2018), and while pragmatics, discourse, and narrative can result in deeper recursion in language (Levinson, 2014), there is arguably a relatively small limit to the depth as well.

In particular, we prove that self-attention networks can both recognize and generate $\text{Dyck}_{k,D}$, with two conceptually simple yet different constructions (Figure 1). The first network requires $D+1$ layers and a memory size of $O(\log k)$ (per layer per token) to recognize $\text{Dyck}_{k,D}$, using a distributed mechanism of parenthesis matching. The second network has two layers and memory size $O(\log k)$. It works by attending to all previous tokens to count the depth for each token in the first layer, and then uses this depth information to attend to the most recent unclosed open bracket in the second layer. Our constructions help reconcile the result in Hahn (2020) with the success of Transformers in handling natural languages.

Our proof requires certain assumptions about the positional encodings, an issue that is often considered in empirical papers (Ke et al., 2021; Shaw et al., 2018; Wang et al., 2020; Shiv and Quirk, 2019) but not in the more theoretical literature. First, positional encodings must have $\log n$ bits when the input length is n, as otherwise different positions would share the same representation. More importantly, positional encodings should support easy position comparisons, since token order is vital in formal language processing. Our experiments show that two standard practices, namely learnable or fixed sine/cosine positional encodings, cannot generalize well on $\text{Dyck}_{k,D}$ beyond the training input lengths. In contrast, using a single fixed scalar monotonic positional encoding such as pos/n achieves near-perfect accuracy even on inputs significantly longer than the training ones. Our findings provide a novel perspective on the function of positional encodings, and implies that different applications of self-attention networks (in this case, natural vs. formal language) may require different model choices.

Our theoretical results also bring about interesting comparisons to recurrent networks (e.g. RNNs, LSTMs) in terms of the resource need to process hierarchical structure. While recurrent networks with finite precision need at least $\Omega(D \log k)$ memory to process $\text{Dyck}_{k,D}$ (Hewitt et al., 2020), our second construction requires only $O(\log k)$ memory but a $O(\log n)$ precision. In experiments where precision is not an issue for practical input lengths ($< 10^4$), we confirm that a Transformer requires less memory than a LSTM to reach high test accuracies. This may help explain why Transformers outperform RNNs/LSTMs in syntactical tasks in NLP, and shed light into fundamental differences between recurrent and non-recurrent sequence processing.

2 Related work

Our work primarily relates to the ongoing effort of characterizing theoretical abilities (Pérez et al., 2019; Bhattamishra et al., 2020b; Yun et al., 2020)

and limitations of self-attention networks, particularly through formal hierarchical structures like Dyck_k. Hahn (2020) proves that (even with positional encodings) hard-attention Transformers cannot model Dyck_k, and soft-attention Transformers with bounded Lipschitz continuity cannot model Dyck_k with perfect cross entropy. Bhattamishra et al. (2020a) prove a soft-attention network with positional masking (but no positional encodings) can solve Dyck_1 but not Dyck_2. Despite the *expressivity* issues theoretically posed by the above work, empirical findings have shown Transformers can *learn* Dyck_k from finite samples and outperform LSTM (Ebrahimi et al., 2020). Our work addresses the theory-practice discrepancy by using positional encodings and modeling $\text{Dyck}_{k,D}$.

A parallel line of work with much lengthier tradition (Elman, 1990; Das et al., 1992; Steijvers and Grünwald, 1996) investigates the abilities and limitations of recurrent networks to process hierarchical structures. In particular, RNNs or LSTMs are proved capable of solving context-free languages like Dyck_k given infinite precision (Korsky and Berwick, 2019) or external memory (Suzgun et al., 2019; Merrill et al., 2020). However, Merrill et al. (2020) also prove RNNs/LSTMs cannot process Dyck_k without such assumptions, which aligns with experimental findings that recurrent networks perform or generalize poorly on Dyck_k (Bernardy, 2018; Sennhauser and Berwick, 2018; Yu et al., 2019). Hewitt et al. (2020) propose to consider $\text{Dyck}_{k,D}$ as it better captures natural language, and show finite-precision RNNs can solve $\text{Dyck}_{k,D}$ with $\Theta(D \log k)$ memory.

For the broader NLP community, our results also contribute to settling whether self-attention networks are restricted to model hierarchical structures due to non-recurrence, a concern (Tran et al., 2018) often turned into proposals to equip Transformers with recurrence (Dehghani et al., 2019; Shen et al., 2018; Chen et al., 2018; Hao et al., 2019). On one hand, Transformers are shown to encode syntactic (Lin et al., 2019; Tenney et al., 2019; Manning et al., 2020) and word order (Yang et al., 2019) information, and dominate syntactical tasks in NLP such as constituency (Zhang et al., 2020) and dependency (He and Choi, 2019) parsing. On the other hand, on several linguistically-motivated tasks like English subject-verb agreement (Tran et al., 2018), recurrent models are reported to outperform Transformers. Our results help address the issue by confirming that distributed and recurrent sequence processing can both model hierarchical structure, albeit with different mechanisms and tradeoffs.

3 Preliminaries

3.1 Dyck Languages

Consider the vocabulary of k types of open and close brackets $\Sigma = \cup_{i\in[k]}\{\langle_i, \rangle_i\}$, and define $\text{Dyck}_k \subset \gamma\Sigma^*\omega$ (γ, ω being special start and end tokens) to be the formal language of well-nested brackets of k types. It is generated starting from $\gamma X \omega$ through the following context-free grammar:

$$X \to \epsilon \mid \langle_i \; X \; \rangle_i \; X \quad (i \in [k]) \tag{1}$$

where ϵ denotes the empty string.

Intuitively, Dyck_k can be recognized by sequential scanning with a stack (i.e., a pushdown automaton). Open brackets are pushed into the stack, while a close bracket causes the stack to pop, and the popped open bracket is compared with the current close bracket (they should be of the same type). The *depth* of a string $w_{1:n}$ at position i is the stack size after scanning $w_{1:i}$, that is, the number of open brackets left in the stack:

$$d(w_{1:i}) = \mathsf{count}(w_{1:i}, \langle) - \mathsf{count}(w_{1:i}, \rangle) \tag{2}$$

Finally, we define $\text{Dyck}_{k,D}$ to be the subset of Dyck_k strings with depth bounded by D:

$$\text{Dyck}_{k,D} = \left\{ w_{1:n} \in \text{Dyck}_k \;\middle|\; \max_{i\in[n]} d(w_{1:i}) \le D \right\}$$

That is, a string in $\text{Dyck}_{k,D}$ only requires a stack with bounded size D to process.

3.2 Self-attention Networks

We consider the encoder part of the original Transformer (Vaswani et al., 2017), which has multiple layers of two blocks each: (i) a self-attention block and (ii) a feed-forward network (FFN). For an input string $w_{1:n} \in \Sigma^*$, each input token w_i is converted into a token embedding via $f_e : \Sigma \to \mathbb{R}^{d_{\text{model}}}$, then added with a position encoding $\mathbf{p}_i \in \mathbb{R}^{d_{\text{model}}}$. Let $\mathbf{x}_{i,\ell} \in \mathbb{R}^{d_{\text{model}}}$ be the i-th representation of the ℓ-th layer ($i \in [n], \ell \in [L]$). Then

$$\mathbf{x}_{i,0} = f_e(w_i) + \mathbf{p}_i \tag{3}$$
$$\mathbf{a}_{i,\ell} = \text{Att}_\ell(Q_\ell(\mathbf{x}_i), K_\ell(\mathbf{x}), V_\ell(\mathbf{x})) \tag{4}$$
$$\mathbf{x}_{i,\ell+1} = F_\ell(\mathbf{a}_{i,\ell}) \tag{5}$$

Attention In each head of a self-attention block, the input vectors $\mathbf{x}_{1:n}$ undergo linear transforms Q, K, V yielding query, key, and value vectors. They are taken as input to a self-attention module, whose t-th output, $\mathrm{Att}(Q\mathbf{x}_i, K\mathbf{x}, V\mathbf{x})$, is a vector $\mathbf{a}_i = \sum_{j\in[T]} \alpha_j V\mathbf{x}_j$, where $\alpha_{1:n} = \mathrm{softmax}(\langle Q\mathbf{x}_i, K\mathbf{x}_1\rangle, \cdots, \langle Q\mathbf{x}_i, K\mathbf{x}_n\rangle)$. The final attention output is the concatenation of multi-head attention outputs. We also consider variants of the basic model along these directions:

(i) *Hard attention*, as opposed to *soft attention* described above, where hardmax is used in place for softmax (i.e. $\mathrm{Att}(Q\mathbf{x}_i, K\mathbf{x}, V\mathbf{x}) = V\mathbf{x}_{j'}$ where $j' = \arg\max_j \langle Q\mathbf{x}_i, K\mathbf{x}_j\rangle$). Though impractical for NLP, it has been used to model formal languages (Hahn, 2020).

(ii) *Positional masking*, where $\alpha_{1:i}$ (past) or $\alpha_{i:n}$ (future) is masked for position i. Future-positional masking is usually used to train auto-regressive models like GPT-2 (Radford et al., 2019).

Feed-forward network A feed-forward network F transforms each self-attention output vector $\mathbf{a}_i \to F(\mathbf{a}_i)$ individually. It is usually implemented as a multi-layer perceptron (MLP) with ReLU activations. *Residual connections* (He et al., 2016) and *layer normalization* (Ba et al., 2016) are two optional components to aid learning.

Positional encodings Vaswani et al. (2017) proposes two kinds of positional encoding: (i) Fourier features (Rahimi and Recht, 2007), i.e. sine/cosine values of different frequencies; (ii) learnable features for each position. In this work we propose to use a single scalar i/n to encode position $i \in [n]$, and show that it helps process formal languages like $\mathsf{Dyck}_{k,D}$, both theoretically and empirically.

Precision and memory size We define *precision* to be the number of binary bits used to represent each scalar, and *memory size per layer* (d_{model}) to be the number of scalars used to represent each token at each layer. The *memory size* ($L \cdot d_{\text{model}}$) is the total memory used for each token.

3.3 Language Generation and Recognition

For a Transformer with L layers and input $w_{1:i}$, we can use a decoder (MLP + softmax) on the final token output $\mathbf{x}_{i,L}$ to predict w_{i+1}. This defines a *language model* $f_\theta(w_{i+1}|w_i)$ where θ denotes Transformer and decoder parameters. We follow previous work (Hewitt et al., 2020) to define how a language model can generate a formal language:

Definition 3.1 (Language generation)**.** *Language model f_θ over $\Sigma^\star$ generates a language $\mathcal{L} \subseteq \Sigma^\star$ if there exists $\epsilon > 0$ such that $\mathcal{L} = \{w_{1:n} \in \Sigma^\star \mid \forall i \in [n], f_\theta(w_i|w_{1:i-1}) \geq \epsilon\}$.*

We also consider language recognition by a *language classifier* $g_\theta(w_{1:i})$, where a decoder on $\mathbf{x}_{i,L}$ instead predicts a binary label.

Definition 3.2 (Language recognition)**.** *Language classifier g_θ over $\Sigma^\star$ recognizes a language $\mathcal{L} \subseteq \Sigma^\star$ if $\mathcal{L} = \{w_{1:n} \in \Sigma^\star \mid g_\theta(w_{1:n}) = 1\}$.*

4 Theoretical Results

In this section we state our theoretical results along with some remarks. Proof sketches are provided in the next section, and details in Appendix A,B,C.

Theorem 4.1 (Hard-attention, $\mathsf{Dyck}_{k,D}$ recognition)**.** *For all $k, D \in \mathbb{N}^+$, there exists a $(D+1)$-layer hard-attention network that can recognize $\mathsf{Dyck}_{k,D}$. It uses both future and past positional masking heads, positional encoding of the form i/n for position i, $O(\log k)$ memory size per layer, and $O(\log n)$ precision, where n is the input length.*

Theorem 4.2 (Soft-attention, $\mathsf{Dyck}_{k,D}$ generation)**.** *For all $k, D \in \mathbb{N}^+$, there exists a 2-layer soft-attention network that can generate $\mathsf{Dyck}_{k,D}$. It uses future positional masking, positional encoding of form i/n for position i, $O(\log k)$ memory size per layer, and $O(\log n)$ precision, where n is the input length. The feed-forward networks use residual connection and layer normalization.*

Theorem 4.3 (Precision lower bound)**.** *For all $k \in \mathbb{N}^+$, no hard-attention network with $o(\log n)$ precision can recognize $\mathsf{Dyck}_{k,2}$ where n is the input length.*

Required precision Both constructions require a precision increasing with input length, as indicated by Theorem 4.3. The proof of the lower bound is inspired by the proof in Hahn (2020), but several technical improvements are necessary; see Appendix C. Intuitively, a vector with a fixed dimension and $o(\log n)$ precision cannot even represent n positions uniquely. The required precision is not unreasonable, since $\log n$ is a small overhead to the n tokens the system has to store.

Comparison to recurrent processing Hewitt et al. (2020) constructs a 1-layer RNN to generate $\mathsf{Dyck}_{k,D}$ with $\Theta(D \log k)$ memory, and proves it is optimal for any recurrent network. Thus Theorem 4.2 establishes a memory advantage of self-attention networks over recurrent ones. However,

this is based on two tradeoffs: (i) *Precision*. Hewitt et al. (2020) assumes $O(1)$ precision while we require $O(\log n)$. (ii) *Runtime*. Runtime of recurrent and self-attention networks usually scale linearly and quadratically in n, respectively.

Comparison between two constructions Theorem 4.2 requires fewer layers (2 vs. D) and memory size ($O(\log k)$ vs. $O(D \log k)$) than Theorem 4.1, thanks to the use of soft-attention, residual connection and layer normalization. Though the two constructions are more suited to the tasks of recognition and generation respectively (Section 5), each of them can also be modified for the other task.

Connection to Dyck_k In Hahn (2020) it is shown that no hard-attention network can recognize Dyck_k even for $k = 1$. Theorem 4.1 establishes that this impossibility can be circumvented by bounding the depth of the Dyck language. Hahn (2020) also points out soft-attention networks can be limited due to bounded Lipschitz continuity. In fact, our Theorem 4.2 construction can also work on Dyck_k with some additional assumptions (e.g. feed n also in input embeddings), and we circumvent the impossibility by using laying normalization, which may have an $O(n)$ Lipschitz constant. More details are in Appendix B.4.

5 Constructions

5.1 $(D+1)$-layer Hard-Attention Network

Our insight underlying the construction in Theorem 4.1 is that, by recursively removing matched brackets from innermost positions to outside, each token only needs to attend to nearest unmatched brackets to find its matching bracket or detect error within D layers. Specifically, at each layer $\ell \le D$, each token will be in one of three states (Figure 2 (c)): (i) *Matched*, (ii) *Error*, (iii) *Unmatched*, and we leverage hard-attention to implement a dynamic state updating process to recognize $\mathsf{Dyck}_{k,D}$.

Representation For an input $w_{1:n} \in \gamma\Sigma^*\omega$, the representation at position i of layer ℓ has five parts $\mathbf{x}_{i,\ell} = [\mathbf{t}_i, o_i, p_i, m_{i,\ell}, e_{i,\ell}]$: (i) a *bracket type embedding* $\mathbf{t}_i \in \mathbb{R}^{\lceil \log k \rceil}$ that denotes which bracket type ($1 \cdots k$) the token is (or if the token is start/end token); (ii) a *bracket openness* bit $o_i \in \{0, 1\}$, where 1 denotes open brackets (or start token) and 0 denotes close one (or end token); (iii) a *positional encoding* scalar $p_i = i/n$; (iv) a *match* bit $m_{i,\ell} \in \{0, 1\}$, where 1 denotes matched and 0 unmatched; (v) an *error* bit $e_{i,\ell} \in \{0, 1\}$, where 1 denotes error and 0 no error. Token identity parts $\mathbf{t}_i, o_i, p_i$ are maintained unchanged throughout layers. The *match* and *error* bits are initialized as $e_{i,0} = m_{i,0} = 0$.

The first D layers have identical self-attention blocks and feed-forward networks, detailed below.

Attention Consider the ℓ-th self-attention layer ($\ell \in [D]$), and denote $\mathbf{x}_i = \mathbf{x}_{i,\ell-1}$, $m_i = m_{i,\ell-1}$, $\mathbf{a}_i = \mathbf{a}_{i,\ell}$, $\mathbf{y}_i = \mathbf{x}_{i,\ell}$ for short. We have 3 attention heads: (i) an identity head $\mathrm{Att}^{\mathsf{id}}$, where each token only attends to itself with attention output $\mathbf{a}_i^{\mathrm{id}} = \mathbf{x}_i$; (ii) a left head $\mathrm{Att}^{\mathsf{left}}$ with future positional masking; (iii) a right head $\mathrm{Att}^{\mathsf{right}}$ with past positional masking. The query, key, and value vectors for $\mathrm{Att}^{\mathsf{left}}$ are defined as $Q\mathbf{x}_i = 1 \in \mathbb{R}$, $K\mathbf{x}_i = p_i - m_i \in \mathbb{R}$, $V\mathbf{x}_i = \mathbf{x}_i \in \mathbb{R}^{d_{\text{model}}}$, so that

$$\mathbf{a}_i^{\mathsf{left}} = \mathbf{x}_{j_1}, \quad j_1 = \arg\max_{j<i}(j/n - m_j)$$

is the representation of the nearest unmatched token to i on its left side. Similarly

$$\mathbf{a}_i^{\mathsf{right}} = \mathbf{x}_{j_2}, \quad j_2 = \arg\max_{j>i}(1 - j/n - m_j)$$

is the representation of the nearest unmatched token to i on its right side. The attention output for position i is the concatenation of these three outputs: $\mathbf{a}_i = [\mathbf{a}_i^{\mathsf{id}}, \mathbf{a}_i^{\mathsf{left}}, \mathbf{a}_i^{\mathsf{right}}] = [\mathbf{x}_i, \mathbf{x}_{j_1}, \mathbf{x}_{j_2}]$.

Feed-forward network (FFN) Following the notation above, the feed-forward network $F : \mathbf{a}_i \to \mathbf{y}_i$ serves to update each position's state using information from $\mathbf{x}_{j_1}, \mathbf{x}_{j_2}$. The high level logic (Figure 2 (c)) is that, if w_i is an open bracket, its potential matching half should be $w_j = w_{j_2}$ ($j_2 > i$), otherwise it should be $w_j = w_{j_1}$ ($j_1 < i$). If w_i and w_j are one open and one close, they either match (same type) or cause error (different types). If w_i and w_j are both open or both close, no state update is done for position i. Besides, token identity parts $\mathbf{t}_i, o_i, p_i$ are copied from $\mathbf{a}_i^{\mathsf{id}}$ to pass on. The idea can be translated into a language of logical operations ($\wedge, \vee, \neg$) plus a $\textsc{same}(\mathbf{t}, \mathbf{t}')$ operation, which returns 1 if vectors $\mathbf{t} = \mathbf{t}'$ and 0 otherwise:

$$\begin{aligned}
\mathbf{y}_i &= [\mathbf{t}_i, o_i, p_i, m_i', e_i'] \\
m_i' &= m_i \vee (o_i \wedge \neg o_{j_2} \wedge s_1) \vee (\neg o_i \wedge o_{j_1} \wedge s_2) \\
e_i' &= e_i \vee (o_i \wedge \neg o_{j_2} \wedge \neg s_1) \vee (\neg o_i \wedge o_{j_1} \wedge \neg s_2) \\
s_1 &= \textsc{same}(\mathbf{t}_i, \mathbf{t}_{j_1}) \quad s_2 = \textsc{same}(\mathbf{t}_i, \mathbf{t}_{j_2})
\end{aligned}$$

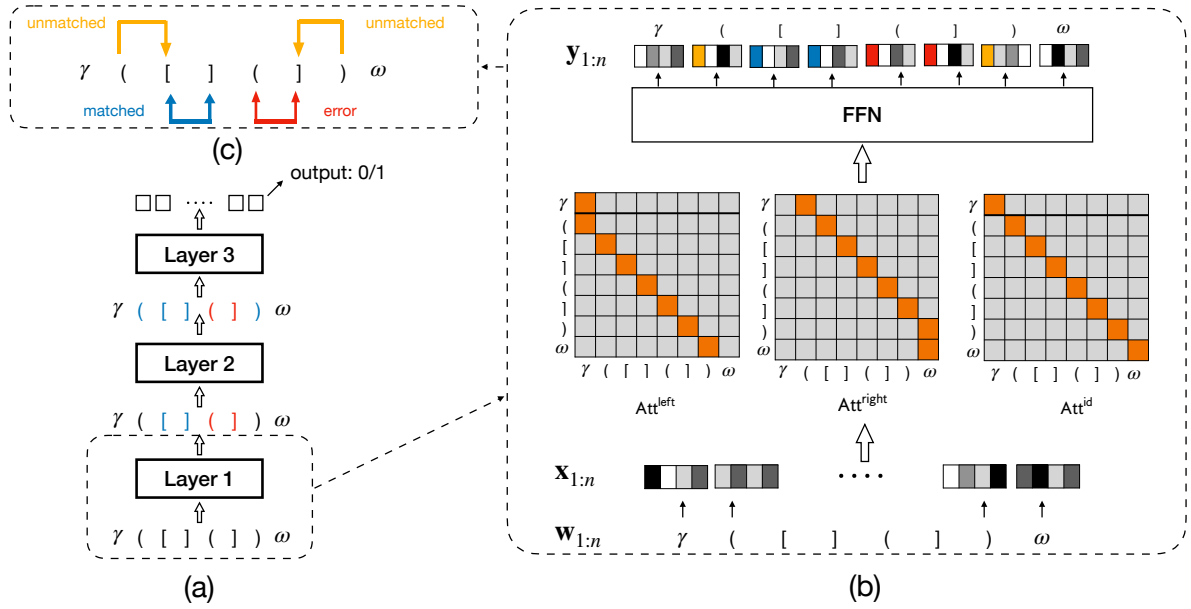


Figure 2: Our construction for Theorem 4.1. (a) The network has multiple identical layers to match brackets and detect errors. (b) Each layer consists of three hard-attention heads so that a token attends to itself and the nearest unmatched tokens on both sides, and uses representations from these positions to update its state. (c) Each position can be in three states: matched, error, or unmatched.

As we show in Appendix A, a multi-layer perception with ReLU activations can simulate all operations $(\wedge, \vee, \neg, \text{SAME})$, thus the existence of our desired FFN.

Final layer At the $(D+1)$-th layer, the self attention is designed as $Q\mathbf{x}_i = 1 \in \mathbb{R}$, $K\mathbf{x}_i = e_i + 1 - m_i \in \mathbb{R}$, $V\mathbf{x}_i = (e_i, m_i) \in \mathbb{R}^2$. If all brackets are matched without error ($(e_i, m_i) = (0, 1)$), all keys would be 0, and the attention output of the last token $\mathbf{a}_n$ would be $(0, 1)$. If any bracket finds error ($e_i = 1$) or is not matched ($m_i = 0$), the key would be at least 1 and $\mathbf{a}_n$ would not be $(0, 1)$. An FNN that emulates $(a, b) \mapsto \neg a \wedge b$ will deliver y_n as the recognition answer.

5.2 Two-layer Soft-Attention Network

Our Theorem 4.2 construction takes advantage of soft attention, residual connection, and layer normalization to calculate each token depth and translate it into a vector form at the first layer. Using the depth information, at the second layer each w_i can attend to the stack-top open bracket at the position, in order to decide if open brackets or which type of close brackets can be generated as the next token (Figure 3).

Representation The representation at position i, layer ℓ has four parts $\mathbf{x}_{i,\ell} = [\mathbf{t}_i, o_i, p_i, \mathbf{d}_{i,\ell}]$, with bracket type embedding $\mathbf{t}_i$, bracket openness bit o_i, position encoding p_i already specified in Section 5.1. The last part $\mathbf{d}_{i,\ell} \in \mathbb{R}^2$ is used to store depth information for position i, and initialized as $\mathbf{d}_{i,0} = (0, 0)$.

First Layer – Depth Counting The first self-attention layer has two heads, where an Att^{id} head is still used to inherit $\mathbf{t}_i, o_i, \mathbf{p}_i$, and a future positional masking head[2] Att^{d} aims to count depth with $Q\mathbf{x}_i = K\mathbf{x}_i = 1$ and $V\mathbf{x}_i = 2o_i - 1$, resulting in uniform attention scores and attention output $a_i^d = \sum_{j \le i} \frac{1}{i} \cdot (2o_j - 1) = d(w_{1:i})/i$.

However, our goal is to enable matching based on depth $d_i = d(w_{1:i})$, and the attention output d_i/i isn't readily usable for such a purpose: the denominator i is undesirable, and even a scalar d_i cannot easily attend to the same value using dot-product attention. Thus in the first feed-forward network, we leverage residual connection and layer normalization to transform

$$d_i/i \mapsto \mathbf{d}_i = (\cos(\theta(d_i)), \sin(\theta(d_i))) \quad (6)$$

where $\theta(d) = \arctan\left(\frac{d}{D+2-d}\right)$ has an unique

[2]Here we assume $w_{i+1:n}$ is masked for position i, just for convenience of description.

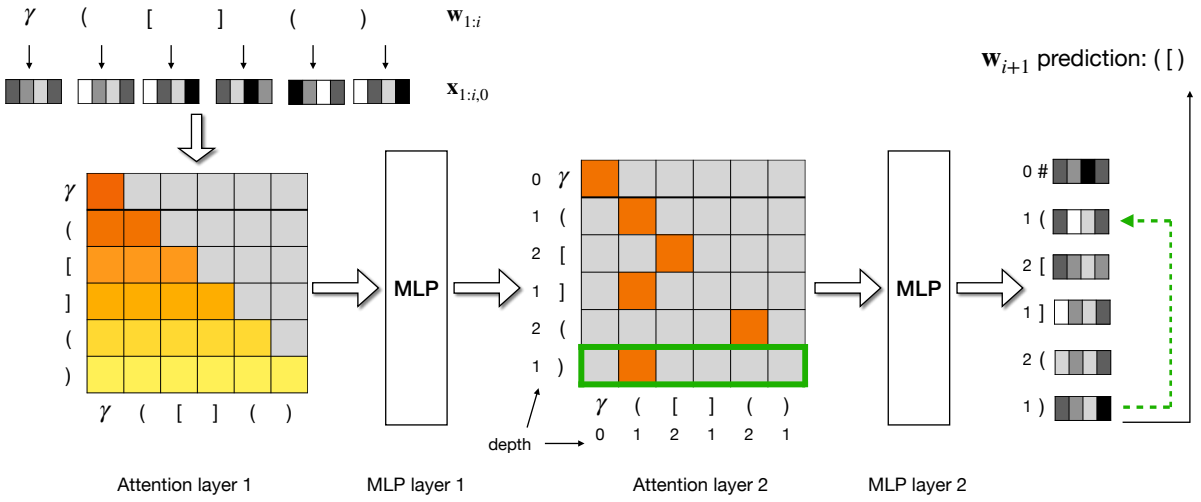


Figure 3: Our construction for Theorem 4.2. The first self-attention layer calculates token depths, while the second layer uses them so that each token attends to the closest unmatched open bracket ign the history, which is useful for next token prediction.

value for every $d \in \{0, \cdots, D+1\}$, so that

$$\mathbf{d}_i \cdot \mathbf{d}_j \begin{cases} = 1 & d_i = d_j \\ < 1 - \frac{1}{10D^2} & d_i \neq d_j \end{cases} \tag{7}$$

The representation by the end of first layer is $\mathbf{x}_{i,1} = [\mathbf{t}_i, o_i, p_i, \mathbf{d}_i]$. The full detail for the first FFN is in Appendix B.1.

Second layer – Depth Matching The second self-attention layer has a depth matching hard-attention head $\text{Att}^{\text{match}}$, with query, key, value vectors as $Q\mathbf{x}_i = [20D^2 \cdot \mathbf{d}_i, 1, 2] \in \mathbb{R}^4$, $K\mathbf{x}_i = [\mathbf{d}_i, p_i, o_i] \in \mathbb{R}^4$, $V\mathbf{x}_i = \mathbf{x}_i$, so that attention score

$$\langle Q\mathbf{x}_i, K\mathbf{x}_j \rangle = 20D^2 \mathbf{d}_i \cdot \mathbf{d}_j + j/n + 2o_j$$
$$\begin{cases} = 20D^2 + 2 + j/n & d_i = d_j, o_j = 1 \\ \leq 20D^2 + 1 & \text{otherwise} \end{cases}$$

would achieve its maximum when w_j ($j \leq i$) is the open bracket (or start token) closest to w_i with $d_j = d_i$. The attention output is $\mathbf{a}_i = [\mathbf{a}_i^{\text{id}}, \mathbf{a}_i^{\text{match}}] = [\mathbf{x}_i, \mathbf{x}_j]$ where $j = \max\{j \leq i | d_i = d_j \wedge o_j = 1\}$.

With such a $[\mathbf{x}_i, \mathbf{x}_j]$, the second-layer FFN can readily predict what w_{i+1} could be. It could be any open bracket when $d_i < D$ (i.e. $\cos(\theta(d_i)) > \cos(\theta(D))$), and it could be a close bracket with type as $\mathbf{t}_j$ (or end token if w_j is start token). The detailed construction for such a FFN is in Appendix B.2.

On Dyck_k Generation In fact, this theoretical construction can also generate Dyck_k, as intuitively the $O(\log n)$ precision assumption allows counting depth up to $O(n)$. But it involves extra conditions like feeding n into network input, and may not be effectively learned in practice. Please refer to details in Appendix B.4.

Connection to Empirical Findings Our theoretical construction explains the observation in Ebrahimi et al. (2020): the second layer of a two-layer Transformer trained on Dyck_k often produces virtually hard attention, where tokens attend to the stack-top open bracket (or start token). It also explains why such a pattern is found less systematically as input depth increases, as (6) is hard to learn and generalize to unbounded depth in practice.

6 Experiments

Our constructions show the *existence* of self-attention networks that are capable of recognizing and generating $\text{Dyck}_{k,D}$. Now we bridge theoretical insights into experiments, and study whether such networks can be *learned* from finite samples and *generalize* to longer input. The answer is affirmative when the right positional encodings and memory size are chosen according to our theory.

We first present results on $\text{Dyck}_{8,10}$ (Section 6.1) as an example $\text{Dyck}_{k,D}$ language to investigate the effect of different positional encoding schemes, number of layers, and hidden size on the Transformer performance, and to compare with the LSTM performance. We then extend the Transformer vs. LSTM comparison on more $\text{Dyck}_{k,D}$ languages ($k \in \{2, 8, 32, 128\}$, $D \in \{3, 5, 10, 15\}$) in Section 6.2. Finally, we apply

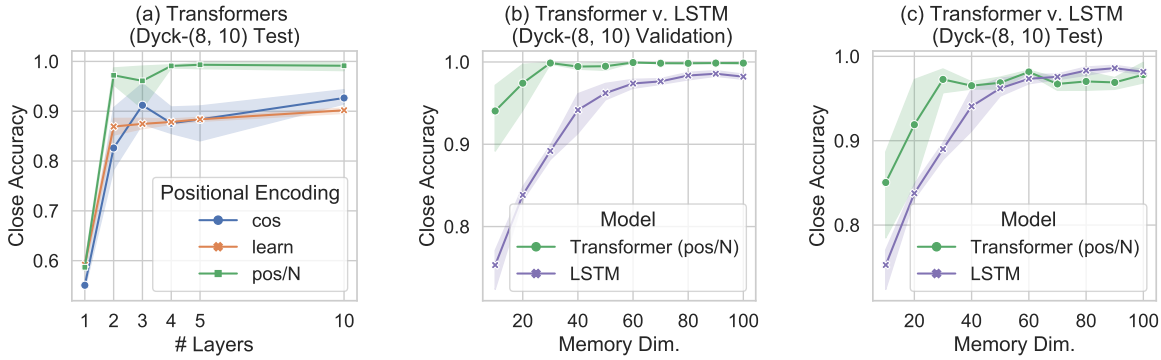


Figure 4: Results on $\text{Dyck}_{8,10}$ validation set (same input lengths as training) and test set (longer inputs). **(a)** compares Transformers of different layers ($L \in \{1, 2, 3, 4, 5, 10\}$) and with different positional encodings (**COS**, **LEARN**,**POS/N**) on the test set. **(b)** and **(c)** compare a 2-layer Transformer (**POS/N**) with a 1-layer LSTM over varying memory sizes on the validation and test sets respectively.

the novel scalar positional encoding to natural language modeling with some preliminary findings (Section 6.3).

6.1 Evaluation on $\text{Dyck}_{8,10}$

Setup For $\text{Dyck}_{8,10}$, we generate training and validation sets with input length $n \leq 700$, and test set with length $700 < n \leq 1400$. We train randomly initialized Transformers using the Huggingface library (Wolf et al., 2019), with one future positional masking head, $L \in \{1, 2, 3, 4, 5, 10\}$ layers, and a default memory size $d_{\text{model}} = 30$. We search for learning rates in $\{0.01, 0.001\}$, run each model with 3 trials, and report the average accuracy of generating *close* brackets, the major challenge of $\text{Dyck}_{k,D}$. More setup details are in Appendix D.1.

Positional Encodings We compare 3 types of positional encodings: (i) Fourier features (**COS**); (ii) learnable features (**LEARN**); (iii) a scalar $i/6000$ for position i (**POS/N**). Note that (i, ii) are original proposals in Vaswani et al. (2017), where positional encoding vectors are *added* to the token embeddings, while our proposal (iii) encodes the position as a *fixed* scalar *separated* from token embeddings.

On the validation set of $\text{Dyck}_{8,10}$ (see Appendix D.2), all three models achieve near-perfect accuracy with $L \geq 2$ layers. On the test set (Figure 4(a)) however, only **POS/N** maintains near-perfect accuracy, even with $L = 10$ layers. Meanwhile, **LEARN** and **COS** fail to generalize, because encodings for position $700 < i \leq 1400$ are not learned (for **LEARN**) or experienced (for **COS**) during training. The result validates our theoretical construction, and points to the need for *separate* and *systemic* positional encodings for processing long and order-sensitive sequences like $\text{Dyck}_{k,D}$.

Memory Size and Comparison with LSTM We compare a two-layer Transformer (**POS/N**) with a one-layer LSTM[3] (Hochreiter and Schmidhuber, 1997) using varying per-layer memory sizes $d_{\text{model}} \in \{10, 20, \cdots, 100\}$. As Figure 4 (b) shows, the Transformer consistently outperforms the LSTM on the validation set. On the test set (Figure 4 (c)), the Transformer and the LSTM first achieve a $> 90\%$ accuracy using $d_{\text{model}} = 20$ and 40 respectively, and an accuracy of $> 95\%$ with $d_{\text{model}} = 30$ and 50, respectively. These findings agree with our theoretical characterization that self-attention networks have a memory advantage over recurrent ones.

6.2 Evaluation on More $\text{Dyck}_{k,D}$ Languages

Setup In order to generalize some of the above results, we generate a wide range of $\text{Dyck}_{k,D}$ languages with different vocabulary sizes ($k \in \{2, 8, 32, 128\}$) and recursion bounds ($D \in \{3, 5, 10, 15\}$). We continue to compare the one-layer LSTM versus the two-layer Transformer (**POS/N**). For each model on each language, we perform a hyperparameter search for learning rate in $\{0.01, 0.001\}$ and memory size $d_{\text{model}} \in \{10, 30, 50\}$, and report results from the best setting based on two trials for each setting.

[3]LSTMs only need one layer to process $\text{Dyck}_{k,D}$ (Hewitt et al., 2020), while Transformers at least need two in our constructions. We also experimented with two-layer LSTMs but did not find improved performance.

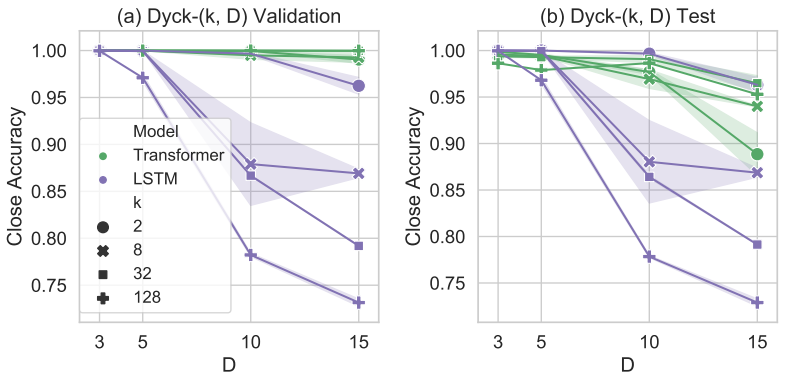


Figure 5: Results on more $\text{Dyck}_{k,D}$ languages.

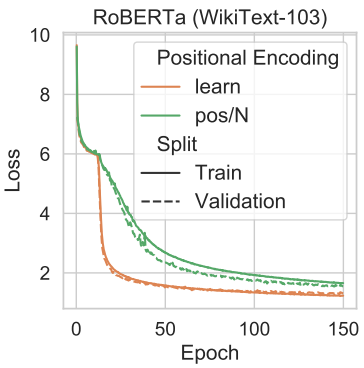


Figure 6: Results on WikiText-103.

Results The validation and test accuracy of the models are reported in Figure 5, and more fine-grained results for each $d_{\text{model}} \in \{10, 30, 50\}$ are in Appendix D.2. The Transformer attains a $> 99.9\%$ validation accuracy and a $> 94\%$ test accuracy across all languages, strengthening the main claim that self-attention networks can learn $\text{Dyck}_{k,D}$ languages and generalize to longer input. On the other hand, the validation and test accuracy of the LSTM model are less than 80% when the vocabulary size and recursion depth are large, i.e. $(k, D) \in \{(32, 15), (128, 10), (128, 15)\}$[4], which reconfirms Transformers' memory advantage under limited memory ($d_{\text{model}} \leq 50$).

6.3 Evaluation on WikiText-103

In Section 6.1, we show a Transformer with the scalar positional encoding scheme (**POS/N**) can learn $\text{Dyck}_{k,D}$ and generalize to longer input, while traditional positional encoding schemes ((**COS**), (**LEARN**)) lead to degraded test performance. To investigate whether such a novel scheme is also useful in NLP tasks, we train two RoBERTa[5] models (**POS/N**, **LEARN**) from scratch on the WikiText-103 dataset (Merity et al., 2017) for 150 epochs.

Figure 6 shows the masked language modeling loss on both training and validation sets. By the end of the training, **POS/N** has a slightly larger validation loss (1.55) than **LEARN** (1.31). But throughout the optimization, **POS/N** shows a gradual decrease of loss while **LEARN** has a sudden drop of loss around 20-30 epochs. We believe it will be interesting for future work to explore how **POS/N** performs on different downstream tasks, and why **POS/N** seems slightly worse than **LEARN** (at least on this MLM task), though theoretically it provides the complete positional information for Transformers. These topics will contribute to a deeper understanding of positional encodings and how Transformers leverage positional information to succeed on different tasks.

7 Discussion

In this paper, we theoretically and experimentally demonstrate that self-attention networks can process bounded hierarchical languages $\text{Dyck}_{k,D}$, even with a memory advantage over recurrent networks, despite performing distributed processing of sequences without explicit recursive elements. Our results may explain their widespread success at modeling long pieces of text with hierarchical structures and long-range, nested dependencies, including coreference, discourse and narratives. We hope these insights can enhance knowledge about the nature of recurrence and parallelism in sequence processing, and lead to better NLP models.

Acknowledgement

We thank Xi Chen, members of the Princeton NLP Group, and anonymous reviewers for suggestions and comments.

Ethical Consideration

Our work is mainly theoretical with no foreseeable ethical issues.

[4]Note that Hewitt et al. (2020) only reports $D \in \{3, 5\}$.

[5]We also tried language modeling with GPT-2 models, and **POS/N** has slightly larger train/validation losses than **LEARN** throughout the training. Interestingly, using no positional encoding leads to the same loss curves as **LEARN**, as positional masking leaks positional information.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Jean-Phillipe Bernardy. 2018. Can recurrent neural networks learn nested recursion? In *Linguistic Issues in Language Technology, Volume 16, 2018*. CSLI Publications.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. 2020a. On the ability of self-attention networks to recognize counter languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7096–7116.

Satwik Bhattamishra, Arkil Patel, and Navin Goyal. 2020b. On the computational power of transformers and its implications in sequence modeling. In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pages 455–475, Online. Association for Computational Linguistics.

Jonathan R Brennan and John T Hale. 2019. Hierarchical structure guides rapid linguistic predictions during naturalistic listening. *PloS one*, 14(1):e0207741.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. 2018. The best of both worlds: Combining recent advances in neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 76–86, Melbourne, Australia. Association for Computational Linguistics.

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

Noam Chomsky and Marcel P Schützenberger. 1959. The algebraic theory of context-free languages. In *Studies in Logic and the Foundations of Mathematics*, volume 26, pages 118–161. Elsevier.

Sreerupa Das, C Lee Giles, and Guo-Zheng Sun. 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*, page 14. Citeseer.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Javid Ebrahimi, Dhruv Gelda, and Wei Zhang. 2020. How can self-attention networks recognize Dyck-n languages? In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4301–4306, Online. Association for Computational Linguistics.

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

Stefan L Frank, Rens Bod, and Morten H Christiansen. 2012. How hierarchical is language use? *Proceedings of the Royal Society B: Biological Sciences*, 279(1747):4522–4531.

Stefan L Frank and Morten H Christiansen. 2018. Hierarchical and sequential processing of language: A response to: Ding, melloni, tian, and poeppel (2017). rule-based and word-level statistics-based processing of language: insights from neuroscience. language, cognition and neuroscience. *Language, Cognition and Neuroscience*, 33(9):1213–1218.

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

Jie Hao, Xing Wang, Baosong Yang, Longyue Wang, Jinfeng Zhang, and Zhaopeng Tu. 2019. Modeling recurrence for transformer. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1198–1207, Minneapolis, Minnesota. Association for Computational Linguistics.

Marc D Hauser, Noam Chomsky, and W Tecumseh Fitch. 2002. The faculty of language: what is it, who has it, and how did it evolve? *science*, 298(5598):1569–1579.

Han He and Jinho D Choi. 2019. Establishing strong baselines for the new decade: Sequence tagging, syntactic and semantic parsing with bert. *arXiv preprint arXiv:1908.04943*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society.

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 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1978–2010, Online. Association for Computational Linguistics.

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

Lifeng Jin, Finale Doshi-Velez, Timothy Miller, William Schuler, and Lane Schwartz. 2018. Unsupervised grammar induction with depth-bounded PCFG. *Transactions of the Association for Computational Linguistics*, 6:211–224.

Fred Karlsson. 2007. Constraints on multiple center-embedding of clauses. *Journal of Linguistics*, pages 365–392.

Guolin Ke, Di He, and Tie-Yan Liu. 2021. Rethinking the positional encoding in language pre-training. In *International Conference on Learning Representations, (ICLR 2021)*.

Samuel A Korsky and Robert C Berwick. 2019. On the computational power of rnns. *arXiv preprint arXiv:1906.06349*.

Stephen C Levinson. 2014. Pragmatics as the origin of recursion. In *Language and recursion*, pages 3–13. Springer.

Yongjie Lin, Yi Chern Tan, and Robert Frank. 2019. Open sesame: Getting inside BERT's linguistic knowledge. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 241–253, Florence, Italy. Association for Computational Linguistics.

Christopher D Manning, Kevin Clark, John Hewitt, Urvashi Khandelwal, and Omer Levy. 2020. Emergent linguistic structure in artificial neural networks trained by self-supervision. *Proceedings of the National Academy of Sciences*, 117(48):30046–30054.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

William Merrill, Gail Weiss, Yoav Goldberg, Roy Schwartz, Noah A. Smith, and Eran Yahav. 2020. A formal hierarchy of RNN architectures. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 443–459, Online. Association for Computational Linguistics.

Matthew J Nelson, Imen El Karoui, Kristof Giber, Xiaofang Yang, Laurent Cohen, Hilda Koopman, Sydney S Cash, Lionel Naccache, John T Hale, Christophe Pallier, et al. 2017. Neurophysiological dynamics of phrase-structure building during sentence processing. *Proceedings of the National Academy of Sciences*, 114(18):E3669–E3678.

Isabel Papadimitriou and Dan Jurafsky. 2020. Learning Music Helps You Read: Using transfer to study linguistic structure in language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6829–6839, Online. Association for Computational Linguistics.

Jorge Pérez, Javier Marinkovic, and Pablo Barceló. 2019. On the turing completeness of modern neural network architectures. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Ali Rahimi and Benjamin Recht. 2007. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 1177–1184. Curran Associates, Inc.

Luzi Sennhauser and Robert Berwick. 2018. Evaluating the ability of LSTMs to learn context-free grammars. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 115–124, Brussels, Belgium. Association for Computational Linguistics.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. 2018. Disan: Directional self-attention network for rnn/cnn-free language understanding. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 5446–5455. AAAI Press.

Vighnesh Leonardo Shiv and Chris Quirk. 2019. Novel positional encodings to enable tree-based transformers. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 12058–12068.

Mark Steijvers and Peter Grünwald. 1996. A recurrent network that performs a context-sensitive prediction task. In *Proceedings of the 18th annual conference of the cognitive science society*, pages 335–339.

Mirac Suzgun, Sebastian Gehrmann, Yonatan Belinkov, and Stuart M Shieber. 2019. Memory-augmented recurrent neural networks can learn generalized dyck languages. *arXiv preprint arXiv:1911.03329*.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.

Ke Tran, Arianna Bisazza, and Christof Monz. 2018. The importance of being recurrent for modeling hierarchical structure. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4731–4736, Brussels, Belgium. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Benyou Wang, Donghao Zhao, Christina Lioma, Qiuchi Li, Peng Zhang, and Jakob Grue Simonsen. 2020. Encoding word order in complex embeddings. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Baosong Yang, Longyue Wang, Derek F. Wong, Lidia S. Chao, and Zhaopeng Tu. 2019. Assessing the ability of self-attention networks to learn word order. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3635–3644, Florence, Italy. Association for Computational Linguistics.

Xiang Yu, Ngoc Thang Vu, and Jonas Kuhn. 2019. Learning the Dyck language with attention-based Seq2Seq models. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 138–146, Florence, Italy. Association for Computational Linguistics.

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J. Reddi, and Sanjiv Kumar. 2020. Are transformers universal approximators of sequence-to-sequence functions? In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Yu Zhang, Houquan Zhou, and Zhenghua Li. 2020. Fast and accurate neural CRF constituency parsing. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4046–4053. ijcai.org.

A Construction Details of Section 5.1

We provide missing details on the construction of $(D+1)$-layer Transformer with hard attention. In particular, we prove that neural networks are capable of simulating logic gates: AND, OR, NOT, SAME and arithmic gates: GREATERTHAN and EQUAL gate. For input $x, y \in \mathbb{R}$, the GREATERTHAN satisfies that $\text{GREATERTHAN}(x, y) = 1$ if $x \geq y + c$ and $\text{GREATERTHAN}(x, y) = 0$ when $x < y$; the EQUAL gate satisfies $\text{EQUAL}(x, y) = 1$ if $x = y$ and $\text{EQUAL}(x, y) = 0$ when $x < y - c$ or $x > y + c$. Here c is a constant independent of x, y.

Lemma A.1. *A constant layer neural network can simulate logic gates:* AND, OR, NOT, SAME *and arithmic gates:* GREATERTHAN, EQUAL.

Proof. Our construction is as follows.

(1) AND gate. Given input $x_1, \ldots, x_m \in \{0, 1\}$, we compute $z = \max\{x_1 + \cdots + x_m - m + 1, 0\}$. We conclude that $z = 1$ iff $x_1 = \cdots = x_m = 1$ and $z = 0$ otherwise.

(2) NOT gate. Given input $x \in \{0, 1\}$, it suffices to compute $z = \max\{1 - x, 0\}$.

(3) OR gate. Given input $x_1, \ldots, x_m \in \{0, 1\}$, we compute $z = \max\{1 - \max\{1 - x_1 - \cdots - x_m, 0\}, 0\}$. It is easy to see that $z = 1$ iff one of $x_i = 1$ ($i \in [m]$) and $z = 0$ otherwise.

(3) SAME gate. Given input $x_1, \ldots, x_m \in \{0, 1\}$ and $y_1, \ldots, y_m \in \{0, 1\}$. The SAME gate is equivalent to $z = ((x_1 \vee \overline{y}_1) \wedge (\overline{x}_1 \vee y_1)) \vee \cdots \vee ((x_m \vee \overline{y}_m) \wedge (\overline{x}_m \vee y_m))$. We can construct it using logic gates: AND, OR, NOT .

(4) GREATERTHAN gate. Given $x, y \in \mathbb{R}$, compute $z_1 = \frac{1}{c}\max\{c - \max\{x - y, 0\}, 0\}$, we have that $z_1 = 0$ when $x > y + c$ and $z = 1$ when $x \leq y$. Taking $z = \max\{1 - z_1, 0\}$ completes the construction.

(5) EQUAL gate. Given $x, y \in \mathbb{R}$. Let $z_1 = \text{GREATEREQUAL}(x, y)$ and $z_2 = \text{GREATEREQUAL}(y, x)$. It suffices to take $z = \neg z_1 \wedge \neg z_2$. □

With some extra effort, one can extend the construction for recognition task to generation task and prove that a D-layer Transformer is capable of generating $\text{Dyck}_{k,D}$.

Corollary A.2. *$\forall k, D \in \mathbb{N}^+$, there exists a D-layer hard-attention network that can generate $\text{Dyck}_{k,D}$. It uses both a future-position masking head and a past-position masking head, a $O(\log k)$ memory size, and $O(\log n)$ precision for processing input length up to n.*

Soft attention Both Theorem 4.1 and Corollary A.2 can be adapted to soft attention, by setting the temperature parameter η in softmax operator to be sufficient large, say $\eta = \Omega(n \log nD)$. Then one can use soft attention to simulate hard attention. In order to fit the precision, for the soft attention distribution $\mathbf{p} = [p_1, \cdots, p_m]$, we round p_i to the closest multiple of $\frac{1}{Cn}$, where C is a large constant.

B Construction Details of Section 5.2

We provide missing details of the construction in Section 5.2.

B.1 First Layer FFN

Recall the output of the first attention layer is $\mathbf{a}_{i,1} = [\mathbf{t}_i, o_i, p_i, \mathbf{d}_{i,1}]$, where $\mathbf{t}_i$, o_i, p_i are the bracket type embedding, the bracket openness bit and the position encoding. $\mathbf{d}_{i,1} \in \mathbb{R}^2$ contains the information d_i/i, where $d_i = d(w_{1:i})$ equals the depth at position i. For ease of presentation, we assume it also contains an entry with $1/i$, this can be derived with an extra attention head in the first layer or be inherited from an extra position encoding. Define $\theta(d) = \arctan\left(\frac{d}{D+2-d}\right)$. We prove

Lemma B.1. *With residual connection and layer normalization, a two-layer MLP can perform the following transformation*

$$(d_i/i, 1/i) \mapsto \mathbf{d}_i = (\cos(\theta(d_i)), \sin(\theta(d_i)))$$

while keeping $\mathbf{t}_i$, o_i, p_i unchanged.

Proof. Consider the following series of operations.

$$\begin{aligned}
&\left(\mathbf{t}_i, o_i, p_i, \frac{d_i}{i}, \frac{1}{i}, 0, 0\right) \\
\mapsto &\left(\mathbf{0}, 0, 0, -\frac{d_i}{i}, \frac{d_i - D - 2}{i}, \frac{d_i}{i}, \frac{D + 2 - d_i}{i}\right) \\
\mapsto &\left(\mathbf{0}, 0, 0, -\frac{1}{2}\sin(\theta(d_i)), -\frac{1}{2}\cos(\theta(d_i)),\right. \\
&\left.\frac{1}{2}\sin(\theta(d_i)), \frac{1}{2}\cos(\theta(d_i))\right) \\
\mapsto &\left(\mathbf{0}, 0, 0, 0, 0, \frac{1}{2}\sin(\theta(d_i)), \frac{1}{2}\cos(\theta(d_i))\right) \\
\mapsto &\left(\mathbf{t}_i, o_i, p_i, \frac{d_i}{i}, \frac{1}{i}, \frac{1}{2}\sin(\theta(d_i)), \frac{1}{2}\cos(\theta(d_i))\right) \\
\mapsto &\ (\mathbf{t}_i, o_i, p_i, \cos(\theta(d_i)), \sin(\theta(d_i)), 0, 0))
\end{aligned}$$

The first steps can be achieved with a linear transformation, the second step can be achieved by layer

normalization and the third step follows from the ReLU activation gate, the fourth step comes from the residual connection and the last step can be obtained with an extra layer of MLP. We conclude the proof here. □

B.2 Second Layer FFN

We can choose between k open brackets and the matched close bracket, with the exception on a few boundary cases: (1) The depth of the current bracket reaches the maximum; (2) The length of the sequence is about to reach the maximum. Let $\widetilde{\mathbf{m}}_i$ be the bracket type of the matched bracket at position i, we implement the last layer as follow.

$$\mathbf{y}_i = [o_i, \mathbf{z}_i, \overline{\mathbf{z}}_i]$$
$$o_i = \neg(\mathbf{d}_{i_1} = \sin(\theta(D))) \wedge \neg(\mathbf{d}_{i_1} = \sin(\theta(\widetilde{D})))$$
$$\widetilde{D} = \min\{n - i, D + 1\}$$
$$\mathbf{z}_i = \neg(\mathbf{d}_{i_1} = 0) \wedge \widetilde{\mathbf{m}}_i$$
$$\overline{\mathbf{z}}_i = \mathbf{1} - \mathbf{z}_i.$$

We elaborate on a few details here. (1) We can derive the term $\sin(\theta(\widetilde{D}))$ via the similar method in Lemma B.1. (2) Since $|\sin(\theta(i)) - \sin(\theta(j))| = \Omega\left(\frac{1}{D^2}\right)$ holds for any $i \neq j \in \{0, 1, \cdots, D+1\}$, we know that the input gap (i.e. the constant c in Lemma A.1) for of all three EQUAL gates is at least $\Omega\left(\frac{1}{d^2}\right)$. Thus we can apply Lemma A.1. (3) We can obtain $n - i$ by either augmenting the position encoding with n and i, or normalizing $(i/n, 1 - i/n)$ (see Lemma B.1).

Output mechanism The final output is determined by on $V\mathbf{y}_{T+2}$, where $V \in \mathbb{R}^{2k \times 2\lceil \log k \rceil + 1}$ satisfies $V_{i,1} = 0$ and $V_{i,1:}$ is the binary encoding of the i-th close bracket and its complement when $i \in \{1, \cdots, k\}$; $V_{i,1} = \lceil \log k \rceil$ and $V_{i,j} = 0$ when $i \leq \{k+1, \cdots, 2k\}$ and $j > 1$. Let $S \subseteq [2k]$ denote the index of valid output, we conclude that $(V\mathbf{y}_{T+2})_i = \lceil \log k \rceil$ for $i \in S$ and $(V\mathbf{y}_{T+2})_i \leq \lceil \log k \rceil - 1$ for $i \notin S$.

B.3 Extension to Recognition task

Our construction can be adapted to recognition task with some extra efforts.

Corollary B.2. *For all $k, D \in \mathbb{N}^+$, there exists a 3-layer soft-attention network that can generate $\mathsf{Dyck}_{k,D}$. It uses future positional masking, positional encoding of form i/n for position i, $O(\log k)$ memory size per layer, and $O(\log n)$ precision where n is the input length. The feed-forward networks use residual connection and layer normalization.*

B.4 Extension to Dyck_k

We can extend the above construction to recognize language Dyck_k. Our construction bypasses the lower bound in Hahn (2020) since the layer normalization operation is not constant Lipschitz (it can be $O(n)$ in the proof).

Theorem B.3 (Soft-attention, Dyck_k generation)**.** *For all $k \in \mathbb{N}^+$, there exists a 2-layer soft-attention network that can generate Dyck_k. It uses future positional masking, $O(\log k)$ memory size per layer, and $O(\log n)$ precision where n is the input length. The feed-forward networks use residual connection and layer normalization.*

Due to space limits, we omit the detailed proof and only outline the major difference from the proof of Theorem 4.2.

1. We need position encoding i/n^3 instead of i/n, and add an extra position encoding of n.
2. For the first FNN, we replace D with n. In particular, for Lemma B.1, we need an extra input of n/i, this can be derived with either an extra attention head or an extra position encoding.
3. For the second FNN, we make some adjustment to the input of the EQUAL gate, since the gap between two input could be very small, i.e., $O(1/n^2)$. Nevertheless, we can use the same trick of Lemma B.1 to amplify the gap between two input a, b to be of order $\Omega(1)$, the later one suffices to our purpose.

C Theoretical limits for finite position encoding

We prove that a Transformer with finite precision can not recognize $\mathsf{Dyck}_{k,D}$ language. In fact, we show a stronger result: no transformer with $o(\log n)$ precision can recognize $\mathsf{Dyck}_{k,D}$ language of length more than n.

Theorem C.1 (Formal statement of Theorem 4.3)**.** *For any $k \in \mathbb{N}$, using hard attention, no transformer with $o(\log n)$ encoding precision can recognize $\mathsf{Dyck}_{k,2}$ language with input length n.*

Our proof is inspired by Hahn (2020) but with several different technique ingredient: (1) we allow arbitrary attention masking (both future and past

position masking); (2) we allow arbitrary position encoding (3) our lower bounds holds for bounded depth language $\mathsf{Dyck}_{k,D}$; (4) we provide an quantitative bound for precision in terms of input length n. In general, our lower bound is *incomparable* with Hahn (2020), we prove a fine grained bound on the precision requirement for bounded depth language $\mathsf{Dyck}_{k,D}$, while the proof in Hahn (2020) applies only for language with Depth $\Omega(n)$ but allows arbitrary precision on position encoding.

The high level intuition behind our proof is that the attention head can only catch $o(n)$ input positions when we properly fix a small number of symbol in the input sequence. This limits the capability of a Transformer and makes it fail to recognize $\mathsf{Dyck}_{k,D}$ language.

We consider a L-layer transformer and assume $3H$ attention heads in total: H normal attention heads, H attention heads with future position masking, H attention heads with past position masking. To make our hardness result general, we allow residual connection for the attention layer, and we assume the FNN can be arbitrary function defining on the attention outcome. In the proof, we would gradually fix $o(n)$ positions of the input sequence. We only perform the follow two kinds of assignment (1) we assign matching brackets to position $i, i+1$ where i is odd; (2) we assign matching brackets (e.g., we assign '[', '(', ')', ']') to position $i, i+1, i+2, i+3$ for odd i. A partial assignment to the input sequence is said to be *well-aligned* if it follows these two rules. Throughout the proof, we guarantee that for any $i \in [n], \ell \in [L]$, the output of the ℓ-th layer $x_{i,\ell}$ depends only the input symbol at position i. This is clearly satisfied for $\ell = 0$, given the it is composed by position embedding and word embedding only. We gradually fix the input and conduction induction on ℓ. We use c_ℓ to denote the number of positions we fixed before the ℓ-th layer, and we use s_ℓ to denote the number of consecutive assigned blocks of the input sequence. It is clear that $s_\ell \le 2c_\ell$. The following Lemma is key to our analysis. Due to space limits, we omit the detailed proof.

Lemma C.2. *For any $\ell \in \{1, \cdots, L\}$, given a well-aligned partially assigned input sequence, suppose the input of ℓ-th layer $\mathbf{x}_{i,\ell-1}$ depends on the symbol at position i only. Then by fixing $c_\ell H^2 (k+1)^{O(\ell H)} 2^{O(\ell H p)}$ additional positions of the input sequence, we guarantee that the output of ℓ-th layer $\mathbf{x}_{i,\ell}$ also depends solely on the symbol at position i.*

Proof of Theorem C.1. We apply Lemma C.2 and compute the number of positions c_{L+1} we need to restrict, in order to guarantee that the output of L-th layer $x_{i,L+1}$ depends only on the input at position ($i \in [n]$). Since $c_{\ell+1} \le c_\ell H^2 (k+1)^{O(\ell H)} 2^{O(\ell H p)}$ and $c_1 = O(1)$, we have

$$c_{L+1} \lesssim H^{O(L)} (k+1)^{O(L^2 H)} 2^{O(L^2 H p)}.$$

By taking

$$H^{O(L)} (k+1)^{O(L^2 H)} 2^{O(L^2 H p)} \le 0.01n.$$

We know the partial assigned sequence is well-aligned, has depth at most two, and the number of assignment is only 0.01. Thus, we assert that that when $p = o(\log n)$, the output of Transformer is completely determined by the partial assignment and it do not detect whether there exists error in the unassigned positions and thus can not recognize $\mathsf{Dyck}_{k,2}$ language. We conclude the proof here. □

D Experiment Details

D.1 Setup

Data We follow Hewitt et al. (2020) to generate $\mathsf{Dyck}_{k,D}$ by randomly sampling stack decisions (push, pop, or end) and maintaining length conditions (Table 1) for a $O(D^2)$ hitting time of different DFA states. The number of tokens for train, validation, and test set is $2 \times 10^6, 2 \times 10^5, 10^6$ respectively.

D	3	5	10	15
Train/val lengths	1:84	1:180	1:700	1:1620
Test lengths	85:168	181:360	701:1400	1621:3240

Table 1: Input lengths for $\mathsf{Dyck}_{k,D}$ with different D.

Models We use the LSTM model implemented in Hewitt et al. (2020). For Transformer models, we turn off all drop outs as we find them to hurt performance greatly. We also use only 1 head as we find more heads to hurt performance. We use Adam optimizer with initial learning rate being 0.01 or 0.001, and choose the better learning rate in terms of validation accuracy for each experiment. We train for at most 100 epochs but allow early stopping if the validation loss converges.

Metric We follow Hewitt et al. (2020) and use the accuracy of correct close bracket predictions:

$$p(\rangle_j | \rangle) = \frac{p(\rangle_j)}{\sum_i p(\rangle_i)}$$

Let p_l be the empirical probability that the model confidently predicts a close bracket (defined as $p(\rangle_j|\rangle) > .8$), conditioned on it being separated from its open bracket by l tokens. Unlike Hewitt et al. (2020) where $\text{mean}_l p_l$ is reported, we report $\mathbb{E}_l p_l$ for two reasons: (i) when l is large p_l might be only defined by one trail, thus $\text{mean}_l p_l$ amplifies the randomness; (ii) the findings remain similar with either metrics.

D.2 More Results

In Figure 7, we show the validation performance for Transformers of different positional encoding schemes. They all reach near-perfect accuracy when having at least 2 layers.

In Figure 8, we break down the results in Section 6.2 when $d_{\text{model}} \in \{10, 30, 50\}$. We also add results for a five-layer Transformer, which performs similarly as the two-layer Transformer. This shows (i) a two-layer Transformer, as suggested by our theory, is enough to process $\text{Dyck}_{k,D}$, and (ii) Transformers with more layers can also learn to process $\text{Dyck}_{k,D}$ without overfitting or degraded performance.

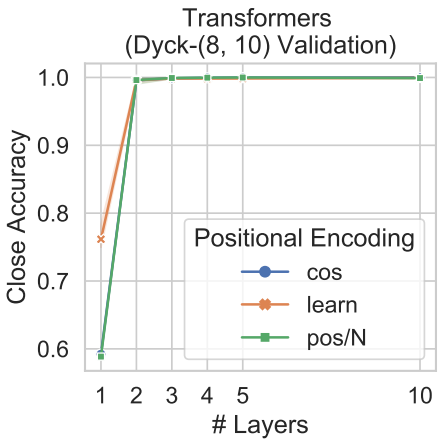


Figure 7: Validation results on $\text{Dyck}_{8,10}$.

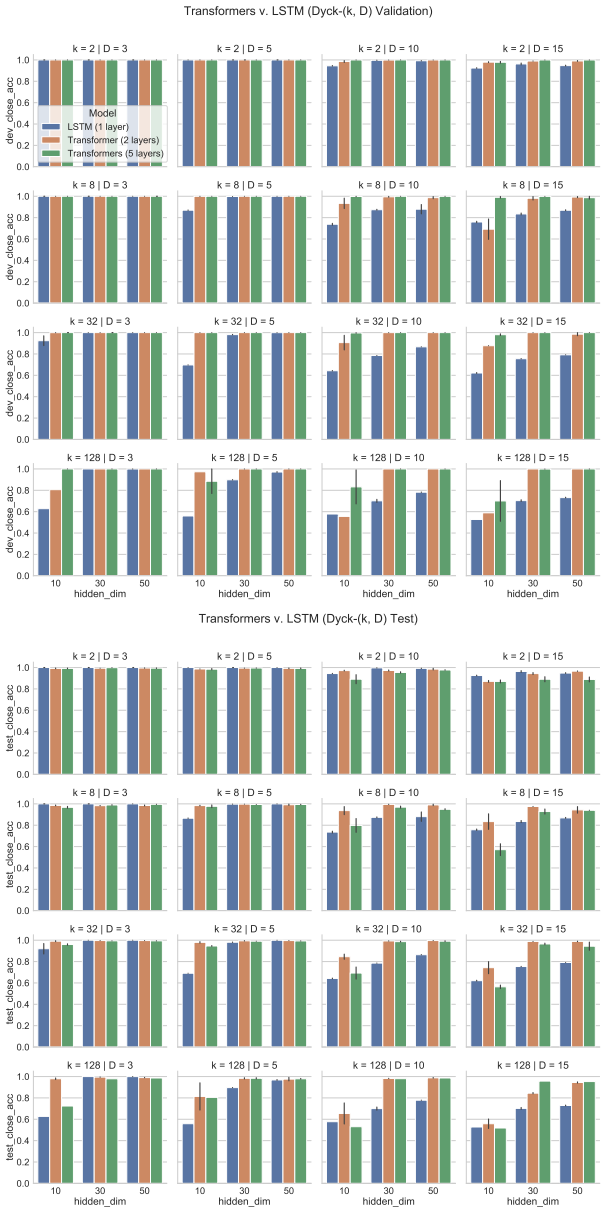


Figure 8: Validation and test results on $\text{Dyck}_{k,D}$ ($k \in \{2, 8, 32, 128\}$ and $D \in \{3, 5, 10, 15\}$). Enlarge for details.