Evaluating Solutions to Theoretical Limitations of Self-Attention for DYCK -Languages

Diana Steffen dsteffen@ethz.ch
Benjamin Glaus bglaus@ethz.ch
Orhan Saeedi osaeedi@ethz.ch

Abstract

In this work we are reproducing the limited ability of transformers to recognize the hierarchical languages 1-DYCK , 2-DYCK and DYCK-(1,2) if the number of heads and layers does not increase with longer sequence lengths. We find that indeed a transformer can recognize short sequences well, while accuracy and cross-entropy deteriorate for longer sequences. In a second step, we construct and implement a transformer which incorporates proposed solutions to the problem for 1-DYCK and DYCK-(1,D) and experimentally verify the improved performance. Our experiments show that the 2-layered transformer results in perfect accuracy recognizing 1-DYCK and DYCK-(1,D) but with high cross-entropy. Training in combination with applying layer-normalization leads to good results for both metrics.

1 Introduction

Motivation Transformers are remarkably effective in a wide range of NLP tasks. However, the differences in their abilities to model different syntactic properties are still largely unknown. Hahn (2020) shows that as input length increases, accuracy in accepting simple formal languages decreases when using a transformer with self-attention. He argued that transformers are limited in correctly modeling periodic finite-state languages (i.e. PARITY) and hierarchical structures (i.e. DYCK) if the number of layers or attention heads does not increase with the length of the input sequence.
There were several approaches to overcome these theoretical limitations of self-attention. Chiang and Cholak (2022) present a solution that detects PARITY with perfect accuracy. They present an explicit construction showing that there are indeed transformers that can detect PARITY with perfect accuracy for any length. Furthermore, they show that the cross entropy can be arbitrarily close to one by adding layer normalization and multiplying attention logits by $\log n$. They have proposed solutions to the theoretical limitations pointed out by Hahn, but only for periodic finite-state languages. However, Hahn's lemma also applies to hierarchical structures such as DYCK (Chomsky and Schützenberger, 1959). This language represents a basic problem underlying all nonregular context-free languages: Correctly closing brackets.

Contribution First, we experimentally reproduced the theoretical constraints highlighted by Hahn. While Hahn only proved his results for 2-DYCK our experiments indicate that the limitations also apply for other DYCK -languages. Our reproduction of Hahn's setup shows that a simple transformer with a fixed number of layers and heads also has problems learning simpler languages like 1-DYCK and DYCK-(1,D) .

Second, we explore which of the methods proposed by Chiang and Cholak (2022) are applicable to hierarchical structures.

To achieve these two things we extended the publicly provided implementation of Chiang et al. We generated the dataset ourselves. In their first solution Chiang and Cholak (2022) come up with a possible construction of the word embedding, positional encoding and weights, to show there exists a transformer that can have perfect accuracy for periodic finite-state languages. We came up with an explicit construction of a transformer that recognizes 1-DYCK and DYCK-(1,D) with perfect accuracy for arbitrary lengths and which can learn these languages much better than a standard transformer. Later, we also experimentally explored if using layer normalization can be useful to classify these two DYCK languages.

Related work Building on the work of Hahn (2020), multiple recent studies investigate the power and limitation of transformers in language modeling. Hao et al. (2022) confirm the results of Hahn and strengthen the findings that transformers with generalized unique hard attention are not able to recognize N-DYCK ($N \geq 1$) or PARITY , by providing a simpler and more general proof based on the set of families of Boolean circuits of both constant depth and polynomial size (AC^0). The work of Yao et al. (2021) offers ideas explaining why transformers perform in language modeling tasks regardless of the mentioned limitations, by constructing attention networks recognizing and generating DYCK-(N,D) , a language similar to DYCK with the level of nestedness limited to D, suggesting that the hierarchical structure in natural language is also bounded. Various experiments have been studied for variations of the DYCK -language: Bhattamishra et al. (2020) provide a systematic study on 25 carefully chosen languages on the ability of transformers to model regular and counter languages. The investgated languages include 1-DYCK , PARITY and SHUFFLE-DYCK , which is similar to 2-DYCK but ignoring the order of parentheses. For SHUFFLE-DYCK they find that small-sized transformers can generalize well. Ebrahimi et al. (2020) study how self-attention networks can recognize N-DYCK languages, and empirically show that adding a starting symbol to the vocabulary enables a 2-headed transformer-encoder to learn N-DYCK -languages and to generalize well to longer sequences. The authors also compare their two networks with more traditional LSTMs for $N = 1, 2, 3, 4$. With a different approach, Weiss et al. (2021) propose a computational model for transformer-encoders in the form of a programming language called RASP (Restricted Access Sequence Processing language), together with RASP programs for selected DYCK -languages. The authors show through their model that transformers can fully recognize N-DYCK for all N. Other approaches try to learn DYCK -languages with similar architectures, like RNN's (DuSell and Chiang, 2020, 2021; Hewitt et al., 2020; Suzgun et al., 2019).

In our work we concentrate on reproducing the theoretical limitations found in Hahn (2020) for 1-DYCK , DYCK-(1,D) , and 2-DYCK , and on applying the solutions proposed in Chiang and Cholak (2022) to overcome the limitations recognizing those languages.

2 Background

In this section, we will provide a short overview on transformers and present relevant definitions of formal languages.

2.1 Transformers

Before transformers, sequence-to-sequence models consisting of an encoder-decoder recurrent network architecture with attention mechanisms were widely used for natural language tasks. In 2017, Vaswani et al. (2017) proposed a different approach, relying only on the attention mechanism. This architecture has proven to result in better quality results for machine translation tasks, to be better parallelizable and thus needing less training time, and generalizing well to different tasks. The authors used multi-head attention to jointly attend to information from different positions. This transformer architecture represents the basis of the discussion in this work. The following definitions are heavily adopted from Chiang and Cholak (2022).

Input Layer The input to the network is a word $w \in \Sigma^*$. Let w_i be the i-th symbol of w. The input layer then consists of a word embedding and a positional encoding, which are added together to compute the input vector:

$$\mathbf{a}^{0,i} = \mathrm{WE}(w_i) + \mathrm{PE}(i),$$

where WE : $\Sigma \to \mathbb{R}^d$ is the word embedding and PE : $\mathbb{N} \to \mathbb{R}^d$ is the positional encoding.

Encoder layer This mechanism consists of L layers each of which has a self-attention sublayer followed by a position-wise feedforward sublayer. For $l = 1, \ldots, L$ we have

$$\mathbf{q}^{l,h,i} = \mathbf{W}^{Q,l,h}\mathbf{a}^{l-1,i}$$

$$\mathbf{K}^{l,h} = \left[\mathbf{W}^{K,l,h}\mathbf{a}^{l-1,0} \ldots \mathbf{W}^{K,l,h}\mathbf{a}^{l-1,n}\right]^T$$

$$\mathbf{V}^{l,h} = \left[\mathbf{W}^{V,l,h}\mathbf{a}^{l-1,0} \ldots \mathbf{W}^{V,l,h}\mathbf{a}^{l-1,n}\right]^T$$

$$\mathbf{c}^{l,i} = \mathrm{LN}\left(\sum_{h=1}^{H} \mathrm{Att}(\mathbf{q}^{l,h,i}, \mathbf{K}^{l,h}, \mathbf{V}^{l,h}) + \mathbf{a}^{l-1,i}\right)$$

$$\mathbf{h}^{l,i} = \max\left(0, \mathbf{W}^{F,l,1}\mathbf{c}^{l,i} + \mathbf{b}^{F,l,1}\right)$$

$$\mathbf{a}^{l,i} = \mathrm{LN}\left(\mathbf{W}^{F,l,2}\mathbf{h}^{l,i} + \mathbf{b}^{F,l,2} + \mathbf{c}^{l,i}\right)$$

where $h = 1, \ldots, H$ and $i = 1, \ldots, n$. The function Att represents either hard or soft attention. The function LN is the layer normalization as defined by Chiang and Cholak (2022).

Positional Masking When computing $c^{l,i}$, just before applying the softmax, a mask can be applied to limit the input elements that the self-attention mechanism can attend to. In autoregressive models, a mask disables all elements above the diagonal of the input matrix. This ensures that the self-attention mechanism cannot look forward into the sequence. We rewrite $K^{l,h}$ and $V^{l,h}$ to:

$$\mathbf{K}^{l,h,i}_{\text{masked}} = \left[\mathbf{W}^{K,l,h}\mathbf{a}^{l-1,0} \ldots \mathbf{W}^{K,l,h}\mathbf{a}^{l-1,i}\right]^T$$
$$\mathbf{V}^{l,h,i}_{\text{masked}} = \left[\mathbf{W}^{V,l,h}\mathbf{a}^{l-1,0} \ldots \mathbf{W}^{V,l,h}\mathbf{a}^{l-1,i}\right]^T$$

Output Layer As a final step, the transformer linearly projects the last computed attention to a scalar and applies a sigmoid function.

$$y = \sigma(\mathbf{W}^{L+1}\mathbf{a}^{L,0} + \mathbf{b}^{L+1})$$

where $\mathbf{W}^{L+1} \in \mathbb{R}^{1\times d}$ and $\mathbf{b} \in \mathbb{R}$. The network accepts w if and only if the output probability y is bigger than $\frac{1}{2}$.

2.2 DYCK -Languages

In the following, we will present some definitions of formal languages important to the topic. We adopt the language definitions presented in Hao et al. (2022):

N-DYCK The language N-DYCK is the set of strings over an alphabet of n types of pairs of brackets that are correctly nested and matched. For example, 2-DYCK over the alphabet {(,), [,]} can be described by a context free grammar with productions S $\rightarrow$ ϵ, S $\rightarrow$ (S), S $\rightarrow$ [S], and S $\rightarrow$ SS.

N-SHUFFLE-DYCK The language N-SHUFFLE-DYCK is the shuffle (arbitrary interleaving) of strings from n versions of 1-DYCK each using a different type of bracket pair.

DYCK-(N,D) The language DYCK-(N,D) is the set of strings in N-DYCK in which the depth of nesting of brackets never exceeds D.

3 Method

This section presents our approach at applying the solutions proposed by Chiang and Cholak (2022) to overcome the theoretical limitations of self-attention in modeling the 2-DYCK -language presented by Hahn (2020).

Construction of Dataset To generate the languages described before, we used a simple stack-based approach for the languages N-DYCK and DYCK-(N,D) . For N-DYCK , we uniformly draw one of the types of brackets and open the bracket. We place the closing bracket on top of a stack. As long as we have not reached the word size, we also randomly decide whether to close the bracket which is on top of the stack (if any) or open another bracket. For DYCK-(N,D) , we additionally put a constraint on the size of the stack.
To create words that are not in the DYCK -language of interest, we create a valid DYCK word as specified and change only one letter to any other letter of the alphabet. This is by construction a non-valid DYCK word. With this procedure we try to come as close to the idea of Hahn as possible. We see that this does not cover all possible variants of the DYCK -complement. For this reason, we also implemented a method to create any words of the corresponding DYCK -complement. These words are then used to test if our construction can fully learn the complement of 1-DYCK or DYCK-(1,D) .

Reproducing Limitations For the well-known soft attention transformer (Vaswani et al., 2017) Hahn gave a theoretical proof for 2-DYCK being not representable, which implies that soft attention transformers cannot model any N-DYCK -language for $N \geq 2$. Furthermore, Hahn proved that hard attention transformers, introduced by Pérez et al. (2019), cannot represent any N-DYCK -language for $N \geq 1$. In order to have a baseline and to reproduce Hahn's theoretical results for 2-DYCK in practice, we implemented two networks, one using hard-attention and one using soft-attention. Both transformers were heavily based on the transformer implementation from Vaswani et al. (2017). In contrast to Chiang and Cholak (2022) we also adopted the positional encoding from Vaswani for this part of our project

$$\text{PE}(i, 2j) = \sin\left(\frac{i}{10000^{\frac{2j}{d}}}\right)$$
$$\text{PE}(i, 2j+1) = \cos\left(\frac{i}{10000^{\frac{2j}{d}}}\right)$$

for $j = [1, d]$. We ran several experiments on recognizing increasing-length sequences of 1-DYCK , DYCK-(1,D) , and 2-DYCK examples to observe how the accuracy developed.

Application of Improvements In a second step, we applied the suggested improvements to self-attention presented in Chiang and Cholak (2022) to our baseline networks, and ran the same experiments on increasing-length sequences of 1-Dyck and Dyck-(1,D) examples. We constructed transformers for both languages, see section 4, and we used layer normalization exactly as described by Ba et al. (2016) and Chiang and Cholak (2022).

Comparison and Discussion In a final step, we analyse the experimental results in order to investigate whether the suggested improvements help overcoming the theoretical limitations. Our results imply that Hahn's lemma concerning soft attention also applies to less complex languages like 1-Dyck and Dyck-(1,D) . Further we show how to use the methods developed for periodic finite-state languages for these two hierarchical languages.

4 Exact Solutions

Following Chiang and Cholak (2022) constructions of a transformer capable of detecting Parity, we present similar transformers for 1-Dyck and Dyck-(1,D). We show by explicit construction that we can recognize both 1-Dyck and Dyck-(1,D) with perfect accuracy. Unlike Chiang and Cholak (2022), we use positional masking in order to process the input sequentially. In the following, we use a masked self-attention mechanism as described earlier. By using matrices $\mathbf{K}^{l,h,i}_{\text{masked}}$ and $\mathbf{V}^{l,h,i}_{\text{masked}}$ within the attention function, our model only attends to input elements that precede the current index i.

4.1 1-Dyck

The input to the network will be a string $w = w_0 w_1 \dots w_n$, where we define $w_0 =$ CLS. We will write 0 for an opening bracket and 1 for a closing bracket. Let w_i be the i-th symbol of sequence w. We will use the notation $w[:i] = w_0 \dots w_i$. We define the encoding of a word w_i as followed:

$$\mathbf{a}^{0,i} = \begin{bmatrix} \mathbb{I}[w_i = 0] \\ \mathbb{I}[w_i = 1] \\ \mathbb{I}[w_i = CLS] \\ \mathbb{I}[i = n] \end{bmatrix}$$

We construct a network with $L = 2$ layers and $H = 1$ head. In the first self-attention layer, we use the attention head to compute the (averaged) difference in the number of closing and opening brackets within the sequence until index i:

$$\frac{1}{i}(\text{count}_0 w[:i] - \text{count}_1 w[:i]) = \frac{1}{i} k(w[:i])$$

To compute this expression, we define the weight matrices to be:

$$W^{Q,1,1} = 0$$
$$W^{K,1,1} = 0$$
$$W^{V,1,1} = \begin{bmatrix} & 0^{4\times 4} & & \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Together with the residual connection, we get:

$$c^{1,i} = \begin{bmatrix} \mathbb{I}[w_i = 0] \\ \mathbb{I}[w_i = 1] \\ \mathbb{I}[w_i = CLS] \\ \mathbb{I}[i = n] \\ \frac{1}{i} k(w[:i]) \\ \frac{1}{i} \end{bmatrix}$$

In the following feedforward sublayer, we compute $\mathbb{I}[k(w[:i]) > 0]$ and $\mathbb{I}[k(w[:i]) = 0]$. These two variables let us know if more brackets have been closed than opened, and if the amount of closed and opened brackets is equivalent.

$$W^{F,1,1} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$
$$b^{F,1,1} = 0^{3\times 1}$$

Giving us:

$$h^{1,i} = \frac{1}{i} \begin{bmatrix} \max(0, k(w[:i]) + 1) \\ \max(0, k(w[:i])) \\ \max(0, k(w[:i]) - 1) \end{bmatrix}$$

We then linearly combine those values:

$$W^{F,1,2} = \begin{bmatrix} & 0^{6\times 3} & \\ 0 & 1 & -1 \\ 1 & -2 & 1 \end{bmatrix}$$
$$b^{F,1,2} = 0^{8\times 1}$$

Together with the residual connection, we now have:

$$a^{1,i} = \begin{bmatrix} \mathbb{I}[w_i = 0] \\ \mathbb{I}[w_i = 1] \\ \mathbb{I}[w_i = CLS] \\ \mathbb{I}[i = n] \\ \frac{1}{i} k(w[:i]) \\ \frac{1}{i} \\ \frac{1}{i}\mathbb{I}[k(w[:i]) > 0] \\ \frac{1}{i}\mathbb{I}[k(w[:i]) = 0] \end{bmatrix}$$

In the second layer, we don't use the attention head.

$$W^{Q,2,1} = 0$$
$$W^{K,2,1} = 0$$
$$W^{V,2,1} = 0$$

Due to the residual connection, we have:

$$c^{2,i} = a^{1,i}$$

We use the feedforward sublayer to compute if we have closed all brackets and if the current position is the last one: $\mathbb{I}[k(w[:i]) = 0 \wedge i = n]$.

$$W^{F,2,1} = \begin{bmatrix} -1 & -1 & -1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$
$$b^{F,2,1} = 0$$

Giving us:

$$\begin{aligned} h^{2,i} =& \max\Big(0, \frac{1}{i}\mathbb{I}[k(w[:i]) = 0] + \mathbb{I}[i = n] \\ & - \mathbb{I}[w_i = 0] - \mathbb{I}[w_i = 1] - \mathbb{I}[w_i = \text{CLS}]\Big) \\ =& \frac{1}{i}\mathbb{I}[k(w[:i]) = 0 \wedge i = n] \end{aligned}$$

We again linearly combine this with the previous vector $a^{1,i}$.

$$W^{F,2,2} = \begin{bmatrix} 0^{8\times1} \\ 1 \end{bmatrix}$$
$$b^{F,1,2} = 0^{1\times1}$$

The vector at $a^{2,i}$ position i then is:

$$a^{1,i} = \begin{bmatrix} \mathbb{I}[w_i = 0] \\ \mathbb{I}[w_i = 1] \\ \mathbb{I}[w_i = CLS] \\ \mathbb{I}[i = n] \\ \frac{1}{i}k(w[:i]) \\ \frac{1}{i} \\ \frac{1}{i}\mathbb{I}[k(w[:i]) > 0] \\ \frac{1}{i}\mathbb{I}[k(w[:i]) = 0] \\ \frac{1}{i}\mathbb{I}[k(w[:i]) = 0 \wedge i = n] \end{bmatrix}$$

For an input sequence to be in 1-DYCK, we require that at any position i, the number of closed brackets is at most as large as the number of opened brackets. We see that this is equivalent to:

$$\forall i \leq n. \quad \frac{1}{i}\mathbb{I}[k(w[:i]) > 0] \overset{!}{=} 0$$

Additionally, we require that at the end of the sequence, the number of opened and closed brackets is equivalent. This means that in vector $a^{i,n}$ we require:

$$\frac{1}{i}\mathbb{I}[k(w[:i]) = 0 \wedge i = n] = \begin{cases} 1/i, & \text{if } i = n \\ 0, & \text{otherwise} \end{cases}$$

We can therefore build an output layer that computes:

$$\sum_{i=0}^{n} \frac{1}{i}\mathbb{I}[k(w[:i]) = 0 \wedge i = n] - \frac{1}{i}\mathbb{I}[k(w[:i]) > 0]]$$

Note that this equals $\frac{1}{i} = \frac{1}{n}$ if and only if both conditions are met. Otherwise we know that this term is smaller or equal to 0. Thus, a sigmoid activation function will correctly classify any input sequence.

4.2 DYCK-(1,D)

We define the input string $w = w_0 w_1 \ldots w_n$ equivalent to before with $w_0 = \text{CLS}$. The encoding of a word w_i will contain an additional term for the allowed depth of nested brackets D:

$$\mathbf{a}^{0,i} = \begin{bmatrix} \mathbb{I}[w_i = 0] \\ \mathbb{I}[w_i = 1] \\ \mathbb{I}[w_i = CLS] \\ \mathbb{I}[i = n] \\ \frac{D}{i} \end{bmatrix}$$

We define the weights for the first self-attention sublayer as followed:

$$W^{Q,1,1} = 0$$
$$W^{K,1,1} = 0$$
$$W^{V,1,1} = \begin{bmatrix} & & 0^{5\times5} & & \\ 0 & 0 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Giving us:

$$c^{1,i} = \begin{bmatrix} \mathbb{I}[w_i = 0] \\ \mathbb{I}[w_i = 1] \\ \mathbb{I}[w_i = CLS] \\ \mathbb{I}[i = n] \\ \frac{D}{i} \\ \frac{1}{i} \\ \frac{1}{i}k(w[:i]) \end{bmatrix}$$

The weights for the first feedfoward sublayer are:

$$W^{F,1,1} = \begin{bmatrix} & 0 & 1 & 1 \\ & 0 & 0 & 1 \\ 0^{5\times4} & 0 & -1 & 1 \\ & -1 & 0 & -1 \\ & -1 & -1 & -1 \end{bmatrix}$$
$$b^{F,1,1} = 0^{5\times1}$$
$$W^{F,1,2} = \begin{bmatrix} & & 0^{7\times5} & & \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$
$$b^{F,1,2} = 0^{10\times1}$$

Giving us:

$$h^{1,i} = \frac{1}{i}\begin{bmatrix} \max(0, k(w[:i]) + 1) \\ \max(0, k(w[:i])) \\ \max(0, k(w[:i]) - 1) \\ \max(0, -k(w[:i]) - D) \\ \max(0, -k(w[:i]) - D - 1) \end{bmatrix}$$

$$a^{1,i} = \begin{bmatrix} \mathbb{I}[w_i = 0] \\ \mathbb{I}[w_i = 1] \\ \mathbb{I}[w_i = CLS] \\ \mathbb{I}[i = n] \\ \frac{D}{i} \\ \frac{1}{i} \\ \frac{1}{i}k(w[:i]) \\ \frac{1}{i}\mathbb{I}[k(w[:i]) = 0] \\ \frac{1}{i}\mathbb{I}[k(w[:i]) > 0] \\ \frac{1}{i}\mathbb{I}[k(w[:i]) < -D] \end{bmatrix}$$

In the second encoder layer, we again don't use the attention head:

$$W^{Q,2,1} = 0$$
$$W^{K,2,1} = 0$$
$$W^{V,2,1} = 0$$

We have

$$c^{2,i} = a^{1,i}$$

In the feedforward sublayer, we define the weights to be:

$$W^{F,2,1} = \begin{bmatrix} -1 & -1 & -1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$
$$b^{F,2,1} = 0$$
$$W^{F,2,2} = \begin{bmatrix} 0^{10\times1} \\ 1 \end{bmatrix}$$
$$b^{F,2,2} = 0^{11\times1}$$

The vector $a^{2,i}$ at a position i then is:

$$a^{2,i} = \begin{bmatrix} \mathbb{I}[w_i = 0] \\ \mathbb{I}[w_i = 1] \\ \mathbb{I}[w_i = CLS] \\ \mathbb{I}[i = n] \\ \frac{D}{i} \\ \frac{1}{i} \\ \frac{1}{i}k(w[:i]) \\ \frac{1}{i}\mathbb{I}[k(w[:i]) = 0] \\ \frac{1}{i}\mathbb{I}[k(w[:i]) > 0] \\ \frac{1}{i}\mathbb{I}[k(w[:i]) < -D] \\ \frac{1}{i}\mathbb{I}[k(w[:i]) = 0 \wedge i = n] \end{bmatrix}$$

Similar to before, we can use

$$\frac{1}{i}\mathbb{I}[k(w[:i]) = 0 \wedge i = n]$$
$$\frac{1}{i}\mathbb{I}[k(w[:i]) > 0]$$

to see if the sequence is in 1-DYCK. Unlike before, we now also require the depth to be at most D. Meaning that for any subsequence $w[:i]$, the difference in the number of opening and closing brackets might at most be $-D$. We see that this is equivalent to requiring

$$\forall i \leq n. \quad \mathbb{I}[k(w[:i]) < -D] \overset{!}{=} 0$$

Thus, the output layer computes the following sum:

$$\sum_{i=0}^{n}\Big(\frac{1}{i}\mathbb{I}[k(w[:i]) = 0 \wedge i = n] \\ -\frac{1}{i}\mathbb{I}[k(w[:i]) > 0] \\ -\frac{1}{i}\mathbb{I}[k(w[:i]) < -D]\Big)$$

If all three conditions are met, this sum is equivalent to $1/n$. Otherwise, it is either less or equal to zero. A sigmoid function therefore correctly classifies any input sequence.

5 Experiment

For the soft attention transformer which was built for reproducing results of Hahn we used PyTorch's built-in implementation of transformers (Paszke et al., 2019). For the hard attention transformer and for the constructions specified we modified PyTorch's implementation accordingly. We achieved this with the help of two open repositories: (Chiang, 2022; Bhattamishra, 2020). We always used $L = 2$ layers and $H = 2$ heads and always ran training for 150 epochs. The following and more results can also be found in our shared GitLab repository.

Reproducing Limitations Trying to reproduce Hahn (2020)'s theoretical results, we ran 2-Dyck on a transformer with soft and with hard attention respectively. We additionaly ran 1-Dyck and Dyck-(1,2) on a transformer with soft attention. When the transformer isn't able to generalize on soft attention, then the same follows for hard attention. We observe that shorter string lengths can indeed be learned, when the language is easier. The results showing this can be found in the Appendix A. In Figure 1 one can see that indeed none of the languages could be learned for a string length of 100 in 150 epochs with the standard transformer.

Application of Improvements We assured ourselves with several string lengths that the constructions described in section 4 indeed result in perfect accuracy for 1-Dyck and Dyck-(1,d). For Dyck-(1,d) we further experimented on different depths. The results can be found in the shared GitLab repository. But we also came across the same problem as Chiang and Cholak (2022) that these constructions approach their worst possible cross-entropy value of 1 bit per string with growing string lengths. We therefore also tried training transformers on both 1-Dyck and Dyck-(1,d) for $D = 3$, once without layer normalization and once with layer normalization, where we used the same value as Chiang and Cholak (2022), $\varepsilon = 10^{-5}$.

In figure 2 and figure 3 we can see that our constructed transformer already learns much better than the standard transformer, but adding layer normalization helps the learnability of the transformers even more.

6 Conlusions

In this paper we experimentally investigated the limits of self-attention in hierarchical structure and adapted existing solutions to overcome these limits. The experiments show us that transformers also have problems in learning and recognizing any Dyck -language. This could follow from the fact, that a word from any Dyck -language, is not in the language anymore when changing one single character in the word. This would mean that Hahn (2020)'s Lemma is applicable for every Dyck -language. Our constructions to overcome these limitations show us that the suggestions by Chiang and Cholak (2022) also can be applied to hierarchical languages. We have a construction that can achieve perfect accuracy. We did this by adapting the ideas of Chiang and Cholak (2022) by changing the weights according to the languages of concern 1-Dyck and Dyck-(1,d) and by adding positional masking to the transformer. We were also able to profit from layer normalization in the same way presented in the same paper. The learnability improved and cross entropy decreased by further adding layer normalization to our constructions.

However, we see a lot of future work that could follow from this project. Our experiments clearly indicate that Hahn (2020)'s results also could be formally proven for languages like 1-Dyck and Dyck-(1,d) . Further, one can try to come up with a construction to achieve perfect accuracy for 2-Dyck or any n-Dyck with $n \geq 1$ or prove why this is not possible.

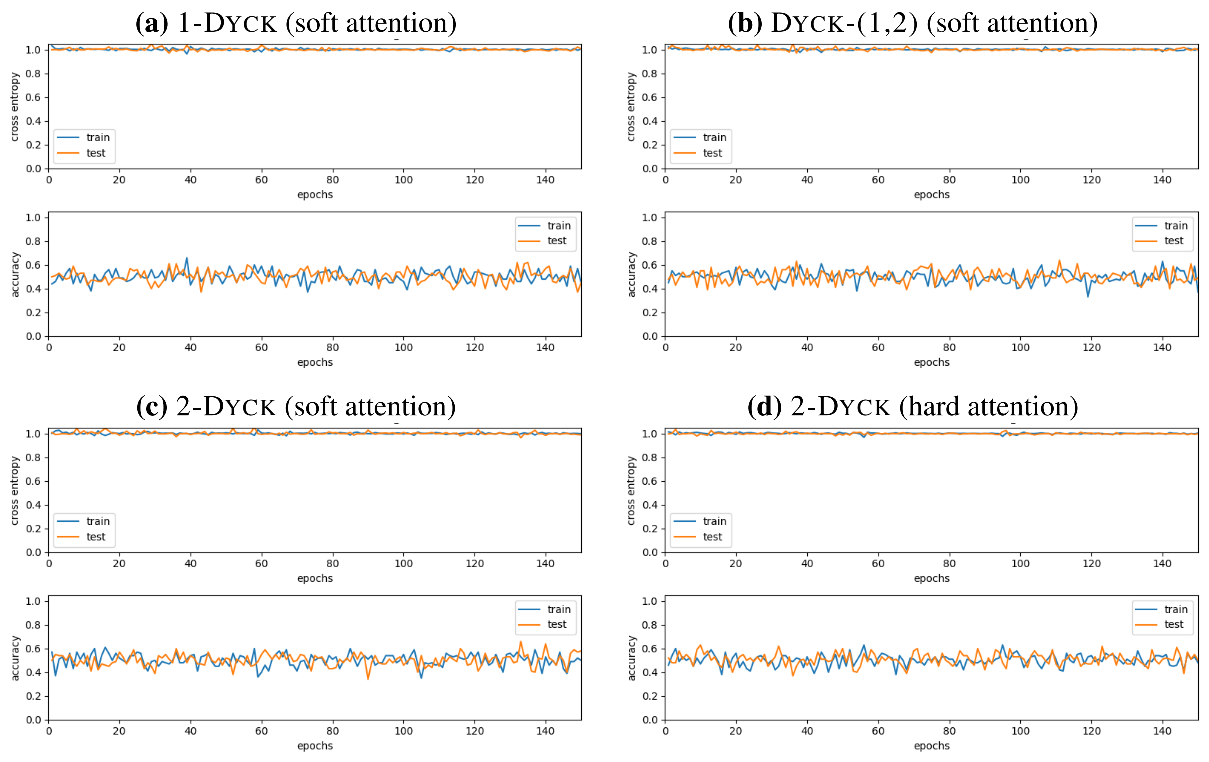


Figure 1: Training a standard transformer on (a) 1-DYCK , (b) DYCK-(1,2) , (c) 2-DYCK with soft attention, and (d) 2-DYCK with hard attention for 150 epochs. Each epoch has 100 training strings and 100 test string of length 100.

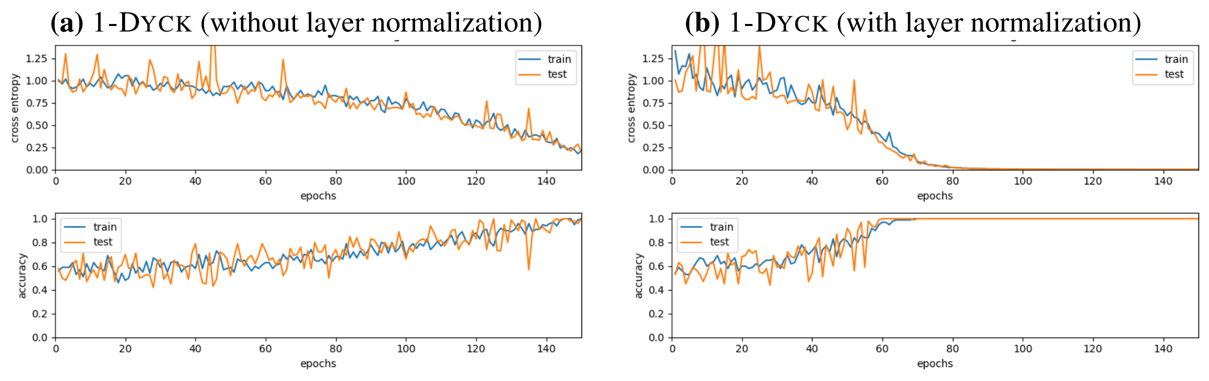


Figure 2: Training the self-constructed transformer on 1-DYCK (a) without and (b) with layer normalization for 150 epochs. Each epoch has 100 training strings and 100 test string of length 100.

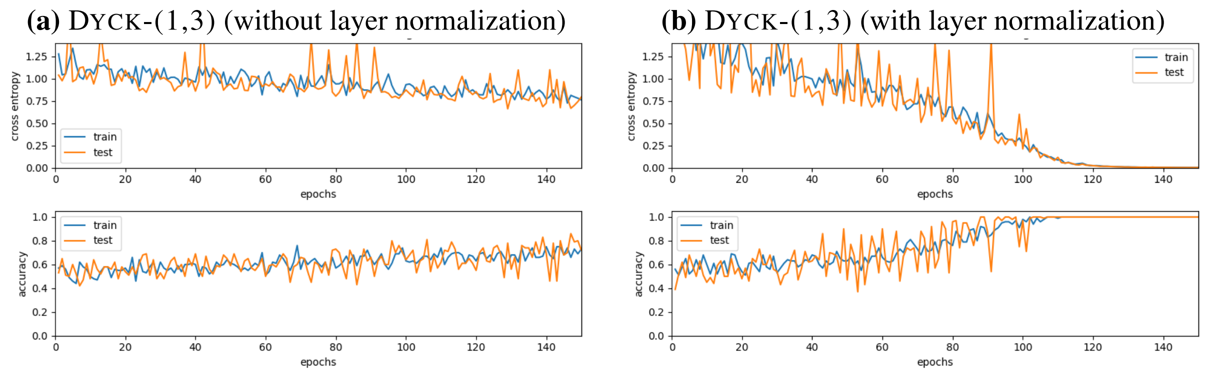


Figure 3: Training the self-constructed transformer on DYCK-(1,3) (a) without and (b) with layer normalization for 150 epochs. Each epoch has 100 training strings and 100 test string of length 100.

References

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

Satwik Bhattamishra. 2020. Satwik77/transformer-formal-languages: Emnlp 2020: On the ability and limitations of transformers to recognize formal languages.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. 2020. On the ability and limitations of transformers to recognize formal languages.

David Chiang. 2022. Ndnlp/parity: Code for "overcoming a theoretical limitation of self-attention".

David Chiang and Peter Cholak. 2022. Overcoming a theoretical limitation of self-attention. *arXiv preprint arXiv:2202.12172*.

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.

Brian DuSell and David Chiang. 2020. Learning context-free languages with nondeterministic stack rnns. *arXiv preprint arXiv:2010.04674*.

Brian DuSell and David Chiang. 2021. Learning hierarchical structures with differentiable nondeterministic stacks. *arXiv preprint arXiv:2109.01982*.

Javid Ebrahimi, Dhruv Gelda, and Wei Zhang. 2020. How can self-attention networks recognize dyck-n languages? *arXiv preprint arXiv:2010.04303*.

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

Yiding Hao, Dana Angluin, and Robert Frank. 2022. Formal language recognition by hard attention transformers: Perspectives from circuit complexity.

John Hewitt, Michael Hahn, Surya Ganguli, Percy Liang, and Christopher D Manning. 2020. Rnns can generate bounded hierarchical languages with optimal memory. *arXiv preprint arXiv:2010.07515*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Jorge Pérez, Javier Marinković, and Pablo Barceló. 2019. On the turing completeness of modern neural network architectures. *arXiv preprint arXiv:1901.03429*.

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*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2021. Thinking like transformers.

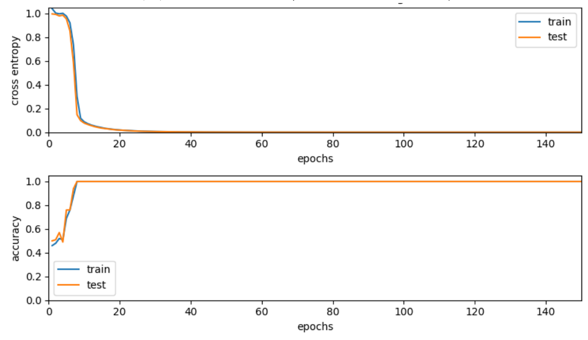
Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. 2021. Self-attention networks can process bounded hierarchical languages. *arXiv preprint arXiv:2105.11115*.

A Further Results

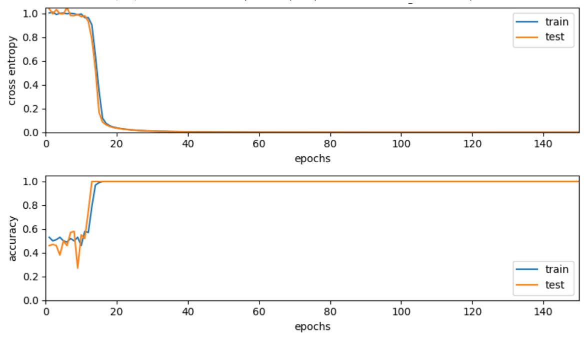
The experiments as described in section 5 were run for different lengths. In this section we will present some further interesting results, but all of them can be also find in the shared GitLab repository. In figure 4 one can see how the standard transformer learns the languages 1-DYCK , DYCK-(1,D) and 2-DYCK when the string length is rather short with 10 tokens.

We also see in figure 5 and figure 6 what happens to the transformers we implemented when they have to learn words of length 10.

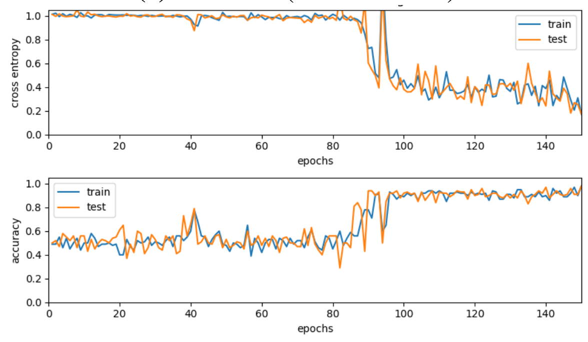
(a) 1-DYCK (soft attention)



(b) DYCK-(1,2) (soft attention)



(c) 2-DYCK (soft attention)



(d) 2-DYCK (hard attention)

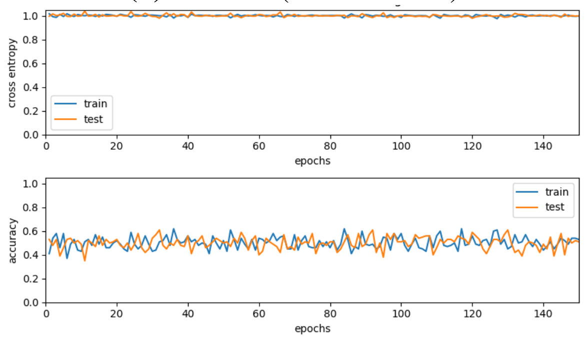
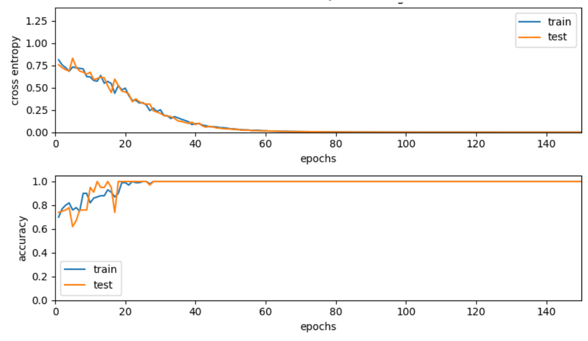


Figure 4: Training a standard transformer on (a) 1-DYCK , (b) DYCK-(1,2) , (c) 2-DYCK with soft attention, and (d) 2-DYCK with hard attention for 150 epochs. Each epoch has 100 training strings and 100 test string of length 10.

(a) 1-DYCK (without layer normalization)



(b) 1-DYCK (with layer normalization)

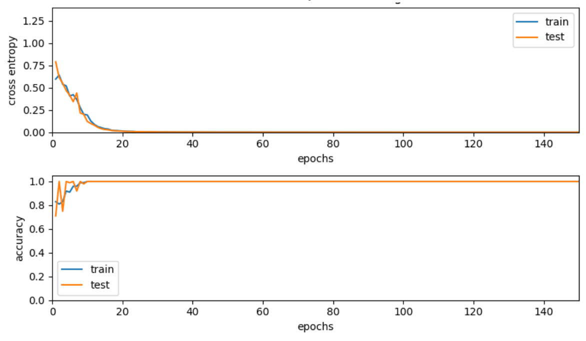
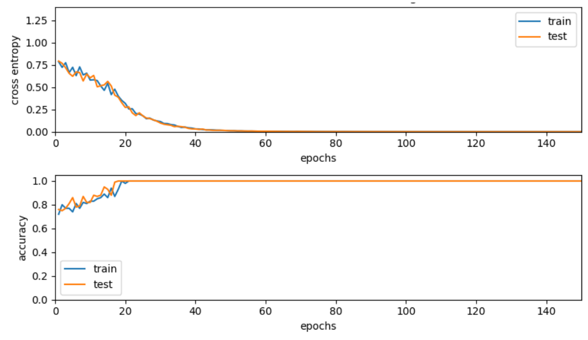


Figure 5: Training the self-constructed transformer on 1-DYCK (a) without and (b) with layer normalization for 150 epochs. Each epoch has 100 training strings and 100 test string of length 10.

(a) DYCK-(1,3) (without layer normalization)



(b) DYCK-(1,3) (with layer normalization)

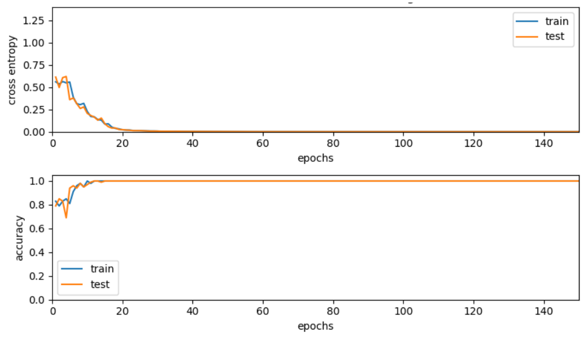


Figure 6: Training the self-constructed transformer on DYCK-(1,3) (a) without and (b) with layer normalization for 150 epochs. Each epoch has 100 training strings and 100 test string of length 100.