Overcoming Theoretical Limitations of Soft Attention

Advanced Formal Language Theory Project Report, Spring 2022

Franz Nowak, Giacomo Camposampiero, Clemente Pasti, and **Bernhard Hilmarsson**
ETH Zürich - Swiss Federal Insitute of Technology
{fnowak, gcamposampie, cpasti, bhilmarsson, }@ethz.ch

1 Introduction

Transformers were initially proposed by Vaswani et al. (2017) and they rapidly arose in popularity in the NLP community as they outperformed previous state of the art techniques in a variety of tasks. While the ability of previous techniques to recognize different classes and types of formal languages has been extensively studied for more that 20 years (Gers and Schmidhuber, 2001; Weiss et al., 2018), relatively little work exists on the ability of transformers to recognize formal languages.

In 2020, Hahn showed that transformers cannot recognize the languages PARITY and 2DYCK for asymptotically large input sizes. The proof rests on the fact that a transformer's output depends on a single input symbol by a factor of $\mathcal{O}(\frac{1}{n})$. This makes it very hard to solve recognition tasks that depend on a single position. Bhattamishra et al. (2020a) proposed an architecture that can recognize 1DYCK, a simpler version of 2DYCK, with perfect accuracy for any finite n using hard attention (Xu et al., 2015).

Chiang and Cholak (2022) explored this further and showed that transformers with hand-crafted weights do exist that recognize FIRST and PARITY with accuracy 1 for any finite input length n, using soft attention.

$$\textsc{First} = \{w \in \Sigma^* \mid w_1 = 1\}$$
$$\textsc{Parity} = \{w \in \Sigma^* \mid w \text{ has odd number of 1s}\}$$

Moreover, layer normalization can be used to get the cross entropy arbitrarily close to 0.

In this work, we consider two other formal languages: The regular language ONE and the context free language PALINDROME.

$$\textsc{One} = \{w \in \Sigma^* \mid w \text{ contains exactly one } 1\}$$
$$\textsc{Palindrome} = \{w \in \Sigma^* \mid w \text{ is palindrome}\}$$

We also show how specifically designed transformer architectures can recognize them with perfect accuracy for any input size n.

The remainder of this manuscript is organized as follows: In Section 2 we introduce our notation and give details on the transformer architecture we used. In Section 3, we derive the custom transformer weights that can recognize ONE and PALINDROME with perfect accuracy. In practice, we find through experiment that the maximum length for which PALINDROME can be recognized perfectly is limited by a problem of floating-precision. In Section 4 we go over the experimental results for the exact and the learned solutions. To run our experiments, we created custom datasets for all the languages we study. For details on the generation of the datasets, see Appendix B. In Section 5 we discuss the limitations of our approach, and finally in Section 6 we draw our conclusions and suggest possible future routes of improvement for our work.

2 Background

2.1 Notation

For ease of comparison, we closely follow the notation of Chiang and Cholak (2022), using $\mathbb{I}[P]$ as the indicator function which is 1 if P is true and 0 otherwise, and $\mathbf{0}^{m\times n}$ for the $m \times n$ 0-matrix.

We define n as the length of the sequence plus any CLS or EOS tokens. The input is a string $w \in \Sigma^*$, with the i-th position of w being denoted by w_i for $i \in [0, ..., n-1]$.

2.2 Transformers

Like Chiang and Cholak (2022) and Hahn (2020), we use transformer encoders with sigmoid output layer (except where otherwise specified) and use the output at position 0 as the classification result.

The transformers are defined in the same way as in the original paper by Vaswani et al. (2017), except we define our own positional encodings as arbitrary functions on the position and length of input.

We use encoder-only transformers, where each encoder layer consists of a stack of L encoder layers, which in turn have a self-attention layer with H heads, followed by a feedforward neural network.

Each input vector $\mathbf{a}^{0,i}$ is the sum of a word embedding WE and positional embedding PE for character at position i:

$$\begin{aligned} \text{WE} &: \Sigma \to \mathbb{R}^d \\ \text{PE} &: \mathbb{N} \to \mathbb{R}^d \\ \mathbf{a}^{0,i} &= \text{WE}(w_i) + \text{PE}(i) \end{aligned}$$

The attention function is standard scaled dot-product attention:

$$\begin{aligned} \text{Att} &: \mathbb{R}^d \times \mathbb{R}^{n\times d} \times \mathbb{R}^{n\times d} \to \mathbb{R}^d \\ \text{Att}(\mathbf{q},\mathbf{K},\mathbf{V}) &= \mathbf{V}^\top \text{softmax}\frac{\mathbf{Kq}}{\sqrt{d}} \end{aligned}$$

Layer $l \in [1,...,L]$ with attention heads $h \in [1,...,H]$ is defined as follows, where lowercase and uppercase boldface letters are vectors and matrices with dimensions d and $d \times d$, respectively:

$$\begin{aligned} \mathbf{q}^{l,h,i} &= \mathbf{W}^{Q,l,h}\mathbf{a}^{l-1,i} \\ \mathbf{K}^{l,h} &= \left[\mathbf{W}^{K,l,h}\mathbf{a}^{l-1,0} \dots \mathbf{W}^{K,l,h}\mathbf{a}^{l-1,n-1}\right]^\top \\ \mathbf{V}^{l,h} &= \left[\mathbf{W}^{V,l,h}\mathbf{a}^{l-1,0} \dots \mathbf{W}^{V,l,h}\mathbf{a}^{l-1,n-1}\right]^\top \\ \mathbf{c}^{l,i} &= \sum_{h=1}^{H} \text{Att}(\mathbf{q}^{l,h,i},\mathbf{K}^{l,h},\mathbf{V}^{l,h}) + \mathbf{a}^{l-1,i} \\ \mathbf{h}^{l,i} &= \max(0, \mathbf{W}^{F,l,1}\mathbf{c}^{l,i} + \mathbf{b}^{F,l,1}) \\ \mathbf{a}^{l,i} &= \mathbf{W}^{F,l,2}\mathbf{h}^{l,i} + \mathbf{b}^{F,l,2} + \mathbf{c}^{l,i} \end{aligned}$$

3 Exact Solutions

In this section, we give specific transformer architectures that recognize each of the introduced formal languages with accuracy 1. As they are designed to recognize a specific formal language, they do not need to be trained.

For readability, we write the vectors without padding, however they are in fact padded to length d where d is the model/embedding dimension.

3.1 Transformer for ONE

Our first exact transformer recognizes the language ONE as defined above. It will have just $L = 1$ encoder layer and $H = 1$ attention head. The embedding dimension is $d = 7$.

The idea for the ONE exact transformer is similar to that for the exact solution of FIRST and PARITY by Chiang and Cholak (2022). We also have prepended a CLS token in position $i = 0$.

3.1.1 Embeddings

The word and position embeddings are fixed (not learned) as follows:

$$\texttt{WE(0)} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \qquad \texttt{WE(1)} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$$

$$\texttt{WE(CLS)} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \qquad \texttt{PE(i)} = \begin{bmatrix}0\\0\\0\\\frac{i}{n}\end{bmatrix}$$

where the position embedding is just a standard relative position encoding that indicates the position as a fraction of the total length.

Thus, the full encoding of word w_i is:

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

3.1.2 Encoder

The only attention layer of the transformer does not attend to anything and is just used to obtain expressions for k and 1, averaged by n:

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

With the residual connection, we get:

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

Now we want to compute $\mathbb{I}[k = 1]$. We do this, in the same way as Chiang and Cholak (2022), by constructing a piecewise linear function that is 1 iff $k = 1$. For this, the FFNN uses two layers. The first layer has the following parameters:

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

Then with RELU activation, the output is:

$$\mathbf{h}^{1,i} = \frac{1}{n} \begin{bmatrix} \max(0, k-2) \\ \max(0, k-1) \\ \max(0, k) \\ 1 \end{bmatrix}$$

And the second layer combines them to get $\mathbb{I}[k = 1]$:

$$\mathbf{W}^{F,2} = \begin{bmatrix} & \mathbf{0}^{6\times 4} & & \\ 1 & -2 & 1 & -0.5 \end{bmatrix}$$
$$\mathbf{b}^{F,2} = \mathbf{0}$$

Note how the fourth dimension of $\mathbf{h}^{1,i}$ is used as a bias term by the second layer, since this way the bias is scaled by $\frac{1}{n}$. After the residual connection, we have the following at position 0 (at CLS):

$$\mathbf{a}^{1,0} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \frac{k}{n} \\ \frac{1}{n} \\ s \end{bmatrix}$$

where s is given by

$$s = \frac{\mathbb{I}[k = 1] - 0.5}{n}$$

which is positive if and only if $\mathbb{I}[k = 1]$ evaluates to 1.

3.1.3 Output Layer

Finally, the output layer just selects the entry in the last dimension and transforms it via a sigmoid activation function:

$$\mathbf{W}^O = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{b}^O = 0$$
$$y = \frac{1}{1 + \exp(-s)}$$

So the output is greater than $\frac{1}{2}$ for $k = 1$ and smaller otherwise.

3.2 Transformer for PALINDROME

The transformer that exactly solves PALINDROME is more non-standard than that for ONE in that it uses an indicator function as activation in the output layer instead of a sigmoid activation. We use $L = 2$ encoder layers with $H = 2$ attention heads in the second layer. The embedding dimension is $d = 11$.

3.2.1 Embeddings

The word embeddings are the same as for ONE, except for this problem the end of the input sequence is required to be marked by an EOS token, for which we add another word embedding:

$$\mathtt{WE(0)} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \mathtt{WE(1)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathtt{WE(CLS)} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \mathtt{WE(EOS)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The position embedding is rather non-standard, introducing markers for being in the left and right halves of the sequence (similar to how Chiang and Cholak's (2022) position encoding for FIRST has an indicator function to mark whether $i = 1$):

$$\mathtt{PE(i)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ i \\ n - i - 1 \\ \mathbb{I}[i \leq \frac{n-1}{2}] \\ \mathbb{I}[i \geq \frac{n-1}{2}] \end{bmatrix}$$

Note that for even sequence lengths n, the sets $\{i | i \leq \frac{n-1}{2}\}$ and $\{i | i \geq \frac{n-1}{2}\}$ are disjoint, while for odd n they both contain the center position $i = \frac{n-1}{2}$.

The encoding of word w_i is:

$$\mathbf{a}^{0,i} = \begin{bmatrix} \mathbb{I}[w_i = 0] \\ \mathbb{I}[w_i = 1] \\ \mathbb{I}[w_i = CLS] \\ \mathbb{I}[w_i = EOS] \\ i \\ n - i - 1 \\ \mathbb{I}[i \leq \frac{n-1}{2}] \\ \mathbb{I}[i \geq \frac{n-1}{2}] \end{bmatrix}$$

3.2.2 First Encoder Layer

The first attention layer does nothing, i.e. $\mathbf{W}^{Q,1} = \mathbf{W}^{K,1} = \mathbf{W}^{V,1} = \mathbf{0}$, and the first FFNN computes two new components, one that checks if at position i we have a 1 and the position is in the first half, and another that checks if it is a 1 in the second half:

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

The second layer just projects the result to the currently empty dimensions after the positional encoding:

$$\mathbf{W}^{F,1,2} = \begin{bmatrix} \multicolumn{2}{c}{\mathbf{0}^{8\times 2}} \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$\mathbf{b}^{F,1,2} = \mathbf{0}$$

$$\mathbf{a}^{1,i} = \begin{bmatrix} \mathbb{I}[w_i = 0] \\ \mathbb{I}[w_i = 1] \\ \mathbb{I}[w_i = CLS] \\ \mathbb{I}[w_i = EOS] \\ i \\ n - i - 1 \\ \mathbb{I}[i \leq \frac{n-1}{2}] \\ \mathbb{I}[i \geq \frac{n-1}{2}] \\ \mathbb{I}[i \leq \frac{n-1}{2} \wedge w_i = 1] \\ \mathbb{I}[i \geq \frac{n-1}{2} \wedge w_i = 1] \end{bmatrix}$$

3.2.3 Second Encoder Layer

The second attention layer has 2 heads. The first head only looks at positions in the first half of the sequence that have a 1, and the second does the same for the second half. Both heads attend more to positions closer to the center of the sequence:

$$\mathbf{W}^{Q,2,1} = \begin{bmatrix} 0 & 0 & c\sqrt{d} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$
$$\mathbf{W}^{K,2,1} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$
$$\mathbf{W}^{V,2,1} = \begin{bmatrix} \multicolumn{4}{c}{\mathbf{0}^{10\times 10}} \\ 0 & \dots & 0 & 1 \end{bmatrix}$$
$$\mathbf{W}^{Q,2,2} = \begin{bmatrix} 0 & 0 & c\sqrt{d} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$
$$\mathbf{W}^{K,2,2} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$
$$\mathbf{W}^{V,2,2} = \begin{bmatrix} \multicolumn{4}{c}{\mathbf{0}^{10\times 10}} \\ 0 & \dots & 0 & -1 \end{bmatrix}$$

The results of both heads are combined and projected into dimension 11 (see Appendix A for details). The second FFNN does nothing ($\mathbf{W}^{F,2,1} = \mathbf{b}^{F,2,1} = \mathbf{W}^{F,2,2} = \mathbf{b}^{F,2,2} = \mathbf{0}$).

The vector at the CLS token after the residual connection is:

$$\mathbf{a}^{2,0} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ n-1 \\ 1 \\ 0 \\ 0 \\ 0 \\ s \end{bmatrix}$$

For $c = \ln 2$, s turns out to be the following (see Appendix A for the derivation):

$$s = \frac{1}{C} \sum_{\text{error at } w_i} (-1)^{\mathbb{I}[w_i=0]} 2^i$$

where $C = 2^n - 1$ and an error at symbol w_i is defined as those symbols in the first half of the sequence where $w_i \neq w_{n-i-1}$. So clearly, $s = 0$ iff the input is a palindrome.

3.2.4 Output Layer

The output layer selects s and uses an indicator function as activation to determine if $s = 0$:

$$\mathbf{W}^O = \begin{bmatrix} 0 & \dots & 0 & 1 \end{bmatrix}$$
$$\mathbf{b}^O = 0$$
$$y = \mathbb{I}[s = 0]$$

Note that in the worst case, the above expression for s has a magnitude of $\mathcal{O}(\frac{1}{2^n})$, meaning that for large n and fixed precision floating point numbers, s may be so small that floating point errors cause the result to be wrong.

3.2.5 Intuition

Figure 1 outlines the intuition behind the exact solution for PALINDROME.

First, we split the sequence into two halves. Then we flip the order of the right half, which is done by the positional encoding entry $n - i - 1$. We then interpret the left half and the reversed right half as binary numbers. This is done implicitly through the softmax in the attention layer, where setting $c = ln2$ converts the softmax into base 2, and the query with the position adds the exponents according to the character position[1]. Finally, subtracting one from the other yields 0 iff they are the same. For a formal derivation, see Appendix A.

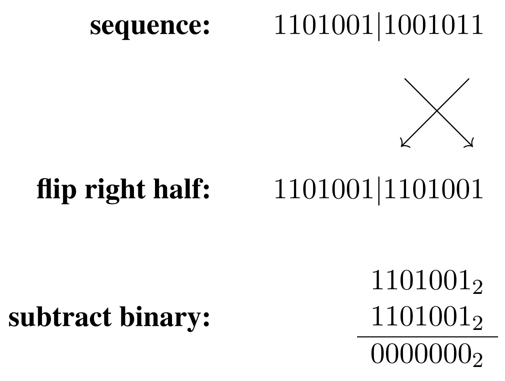


Figure 1: Intuition behind PALINDROME exact solution.

4 Experiments

4.1 Exact solutions

For the languages FIRST, PARITY and ONE we ran the same experiments using the exact solutions. Starting at $n = 10$ and going up to $n = 10000$, in intervals of near multiples of 2. In all cases the accuracy for the exact solutions was 1, i.e. the custom transformers were always able to recognize the language perfectly without training.

For PALINDROME we test all $n \in [1, 200]$. As Figure 2 shows, the exact implementation achieves perfect accuracy in the interval $[1, 37]$. However, the performance deteriorates starting from string length $n = 38$ because of floating point errors, going down to 0.5 accuracy at around $n = 70$.

[1]Note that in fact we are obtaining reverse binary numbers where the most significant bit is the right-most digit.

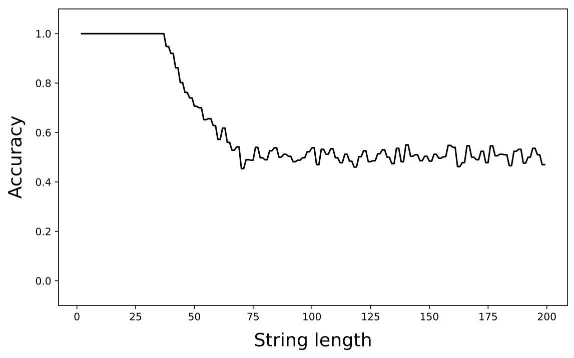


Figure 2: Validation accuracy of Palindrome exact transformer on different string lengths.

From there on our transformer performs no better than random guessing.

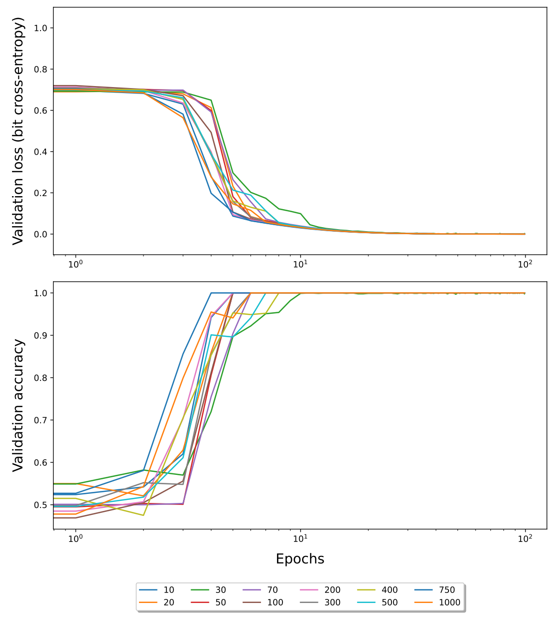
4.2 Learned solutions

As in (Chiang and Cholak, 2022), we investigate the learnability of the four studied languages, FIRST, PARITY, ONE, and PALINDROME. In our work, we define and test two main properties for each language:

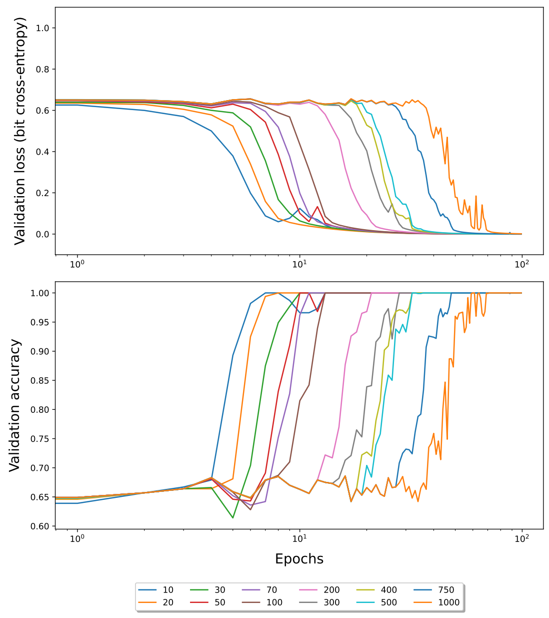
- Learnability, that is the ability of a transformer to learn a solution for strings of fixed length.
- Generalizability, that is the ability of a transformer to learn a solution that allows to generalize to unknown string lengths.

This distinction is proposed after observing that for some languages, as for example ONE, transformers could easily learn solutions for strings of the same length as the training examples, but struggled to find more general solutions that could adapt to arbitrary input lengths.

The two properties were tested using a standard encoder-only implementation of the original transformer, as we did for the exact solutions. All the experimental results were obtained by calculating an average over 20 different runs. As in (Chiang and Cholak, 2022), we used $d_{\text{model}} = 16$ for learned word encodings, self-attention, and FFNN outputs with $d_{\text{FFNN}} = 64$. Layer normalization with $\epsilon = 10^{-1}$ was used after the residual connections. We used PyTorch default initialization for the learned parameters and the Adam optimizer (Kingma and Ba, 2015) with learning rate $\eta = 3 \times 10^{-4}$. We also do not use dropout, as (Chiang and Cholak, 2022) observed that did not seem to help. Moreover, we experimented with



(a) Learning of FIRST.



(b) Learning of ONE.

Figure 3: Learning curves for the languages FIRST and ONE, trained on strings of different lengths for 100 epochs.

different types of positional encoding, namely positional encodings from exact solutions, FIRST and PARITY positional encodings from (Chiang and Cholak, 2022), and standard sinusoidal positional encodings. Finally, we also investigated the use of variable string length for the training examples.

The first property we tested on the four languages was learnability. Figure 3 shows the results of these experiment for FIRST and ONE. In this experiment, we train the transformer with different string lengths s, $10 \leq s \leq 1000$ and report the validation loss and accuracy of the model tested on strings of the same length. It is clear from Figure 3 that the transformer successfully learns both FIRST and ONE, achieving both high accuracy and low per-string cross-entropy for the two languages. We also report the learning curves obtained for PARITY and PALINDROME in Appendix D. Like Chiang and Cholak (2022) and Bhattamishra et al. (2020a), we did not manage to find a learned solution to PARITY and, as shown by Figure 8, PALINDROME was not learnable in our experiments either.

Since generalizability is intrinsically more difficult to achieve than learnability, as the former requires as a necessary condition the latter, we only tested it for languages that we proved to be learnable in our learnability experiments, that is FIRST and ONE. Figure 4 shows the empirical results obtained for FIRST. We observed that the naive implementation of FIRST does not generalize well to unknown string lengths (left column of Figure 4), but this issue can be solved by scaling the attention logits by $\log n$, where n is the length of the training strings (right column of Figure 4). Hence, our results consistently confirm the findings of (Chiang and Cholak, 2022) on the same languages, proving that log-length is effectively able to overcome the limitation suggested by Hahn's lemma.

Figure 5 shows the experimental results for ONE generalizability. The left column reports the results that were obtained training a transformer with positional encodings from our ONE exact solution, on training examples of fixed length. It can be observed that the transformer could not generalize to longer string lengths, despite being able to effectively learn solutions for strings of the same length as the training samples. However, we discovered that a better solution for ONE (which has perfect accuracy when the difference between training and test lengths is not too pronounced) can be learned by using the exact combination of FIRST positional encodings and variable length training samples, with both these parameters being strictly necessary for generalizability (right column of Figure 5).

A more in-depth analysis of the solution weights that this particular transformer was able to recover was attempted, but due to the high dimensionality of the weight matrices, their dense nature, and the complexity of the model, we could not leverage them to obtain additional insights.

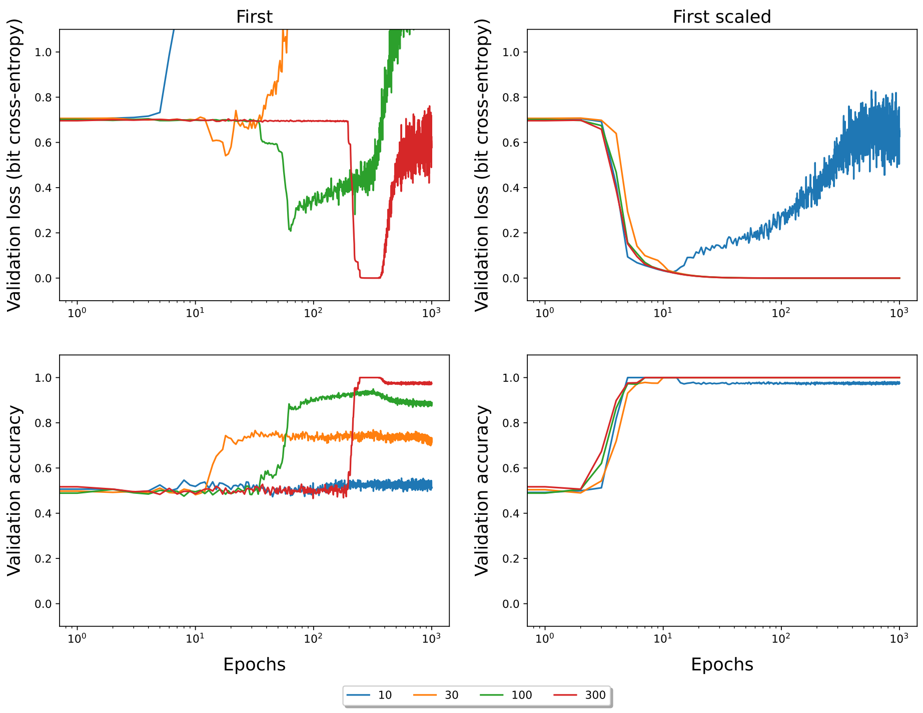


Figure 4: Generalizability learning curves for FIRST, trained and tested over different string lengths for 1000 epochs. Left: Standard FIRST transformer. Right: Scaled FIRST transformer.

5 Discussion

While Bhattamishra et al. (2020b) found an exact solution for specific context free languages using hard attention, we are instead trying to use soft attention to model the context free language PALINDROME. We do in fact find a theoretical solution that works for recognising PALINDROME using only soft attention. However, the practical implementation has an obvious shortcoming, which is that with fixed precision floating point numbers, it can only recognize palindromes reliably up to a certain length ($n \leq 37$). The reason for this is that the solution works by effectively interpreting the left and right side into binary/ reverse binary representation, respectively, whose value grows exponentially with n, meaning the normalisation constant of the softmax in the attention layer grows exponentially too, resulting in numbers of magnitude $\mathcal{O}(\frac{1}{2^n})$ (see Appendix C for more details). This is necessary when using soft attention, since all positions by definition contribute to the attention output, just with different weights. So the symbols at each position need to be "tagged" by the position, which requires a type of unique weighting which the binary coding provides.

6 Conclusion

In response to the limitations postulated by Hahn (2020), Chiang and Cholak (2022) showed that there exist transformers that can recognize PARITY with perfect accuracy. In this work, we have verified their results experimentally, finding that the exact solutions do indeed work as described, but that PARITY cannot be learned. We then extended their results by deriving custom transformer weights that, at least in in theory, can recognize instances of the regular language ONE and the context free language PALINDROME for arbitrary input sizes n. In practice, however, the solution for PALINDROME does not generalise to longer sequences due to floating point precision errors. Chiang and Cholak (2022) also showed that for FIRST and PARITY it is possible to achieve cross-entropy arbitrarily close to zero through layer normalization. An interesting avenue for future work would be to investigate whether the same result can be achieved for PALINDROME and ONE. Our experiment code can be found on Github[2].

[2]https://github.com/giacomocamposampiero/palindrome-transformer

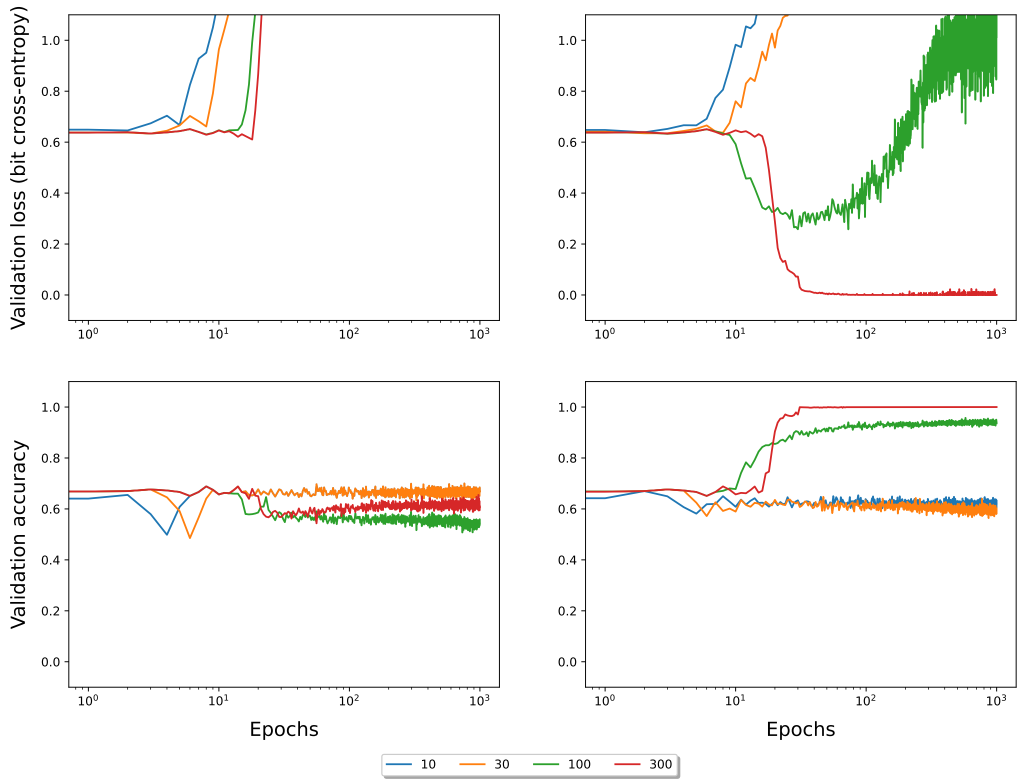


Figure 5: Generalizability learning curves for ONE, trained and tested over different string lengths for 1000 epochs. Left: Standard ONE transformer. Right: ONE transformer with First positional embeddings.

References

S. Bhattamishra, Kabir Ahuja, and Navin Goyal. 2020a. On the ability and limitations of transformers to recognize formal languages. In *EMNLP*.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. 2020b. On the Ability and Limitations of Transformers to Recognize Formal Languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7096–7116, Online. Association for Computational Linguistics.

David Chiang and Peter Cholak. 2022. Overcoming a theoretical limitation of self-attention. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7654–7664, Dublin, Ireland. Association for Computational Linguistics.

F.A. Gers and E. Schmidhuber. 2001. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340.

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

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. On the practical computational power of finite precision rnns for language recognition. In *ACL*.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention.

A Correctness of PALINDROME

In the following we will derive an expression for s as referenced in Section 3.2.3.

As a reminder, the scaled dot-product attention is given by

$$\mathrm{Att}(\mathbf{q},\mathbf{K},\mathbf{V}) = \mathbf{V}^\top \mathrm{softmax}\frac{\mathbf{Kq}}{\sqrt{d}}$$

where for position 0 (at the CLS token) we have:

$$\mathbf{q}^{2,1,0} = \mathbf{q}^{2,2,0} = c\sqrt{d} \times \mathbb{I}[w_i = CLS] = c\sqrt{d}$$

and the keys and values at position i are:

$$\begin{aligned}
\mathbf{K}^{2,1} &= [0,\dots,i,\dots,n-1]^\top \\
\mathbf{V}^{2,1} &= \begin{bmatrix} \mathbf{0}^{10\times n} \\ \mathbb{I}[w_i=1],\dots,\mathbb{I}[w_i=1],0,\dots,0 \end{bmatrix}^\top \\
\mathbf{K}^{2,2} &= [n-1,\dots,n-i-1,\dots,0] \\
\mathbf{V}^{2,2} &= \begin{bmatrix} \mathbf{0}^{10\times n} \\ 0,\dots,0,\mathbb{I}[w_i=1],\dots,\mathbb{I}[w_i=1] \end{bmatrix}
\end{aligned}$$

To simplify the notation, let $\bar{i} := n-i-1$. Then s, which is at position 11 of the output of the last attention layer at position 0 can be written as:

$$\begin{aligned}
s = a^{2,0}_{11} &= \left(\sum_{h\in 1,2} Att\left(\mathbf{q}^{2,h,0},\mathbf{K}^{2,h},\mathbf{V}^{2,h}\right) + \mathbf{a}^{1,0}\right)_{11} \\
&= (\mathbf{V}^{2,1})^\top \mathrm{softmax}\left(\frac{\mathbf{K}^{2,1}\mathbf{q}^{2,1,0}}{\sqrt{d}}\right)_{11} \\
&+ (\mathbf{V}^{2,2})^\top \mathrm{softmax}\left(\frac{\mathbf{K}^{2,2}\mathbf{q}^{2,2,0}}{\sqrt{d}}\right)_{11} + \mathbf{a}^{1,0}_{11}
\end{aligned}$$

Plugging in the values from above and only regarding the 11-th dimension where $a^{1,0}_{11} = 0$, this becomes:

$$\begin{aligned}
s &= \sum_{i=0}^{n-1} \mathbb{I}[w_i = 1 \wedge i \le \tfrac{n-1}{2}] \frac{\exp(ci)}{\sum_{j=0}^{n-1}\exp(cj)} \\
&- \sum_{i=0}^{n-1} \mathbb{I}[w_i = 1 \wedge i \ge \tfrac{n-1}{2}] \frac{\exp[c\bar{i}]}{\sum_{j=0}^{n-1}\exp[c\bar{j}]} \\
&= \sum_{i=0}^{\lfloor\frac{n-1}{2}\rfloor} \mathbb{I}[w_i = 1] \frac{\exp(ci)}{\sum_{j=0}^{n-1}\exp(cj)} \\
&- \sum_{i=\lceil\frac{n-1}{2}\rceil}^{n-1} \mathbb{I}[w_i = 1] \frac{\exp[c\bar{i}]}{\sum_{j=0}^{n-1}\exp[c\bar{j}]} \\
&= \sum_{i=0}^{\lfloor\frac{n-1}{2}\rfloor} \mathbb{I}[w_i = 1] \frac{\exp(ci)}{\sum_{j=0}^{n-1}\exp(cj)} \\
&- \sum_{i=0}^{\lfloor\frac{n-1}{2}\rfloor} \mathbb{I}[w_{\bar{i}} = 1] \frac{\exp(ci)}{\sum_{j=0}^{n-1}\exp(cj)} \\
&= \frac{1}{\sum_{j=0}^{n-1}\exp(cj)} \sum_{i=0}^{\lfloor\frac{n-1}{2}\rfloor} \left(\mathbb{I}[w_i = 1] - \mathbb{I}[w_{\bar{i}} = 1]\right) e^{ci} \\
&= \frac{1}{2^n - 1} \sum_{w_i \ne w_{\bar{i}}} (-1)^{\mathbb{I}[w_i=0]} 2^i
\end{aligned}$$

Where in the last step, we use that $c = \ln 2$.

Trivially, if w is a palindrome, the above is 0 because the sum vanishes. Conversely, since 2 is the basis of the binary numeral system, it is easy to see that if w is not a palindrome, s cannot be 0 as there is no way to sum powers of 2 with coefficients ≤ 1 to get a larger power of 2.

Note that, for even lengths n, $\{i|i \le \frac{n-1}{2}\}$ and $\{i|i \ge \frac{n-1}{2}\}$ are disjoint, while for odd n they both contain the center index $i = \frac{n-1}{2}$. However, when subtracting both sides, the term arising from the center position conveniently vanishes.

Note also that for the above calculations we require the sequence to be symmetric including tags such as CLS and EOS. This is because otherwise the sums from the two halves of the sequence would have different softmax normalisation constants in the derivation above. This is the reason an EOS tag was added to sequences for the PALINDROME exact solution.

B Dataset Details

B.1 FIRST & PARITY

$$\textsc{First} = \{w \in \Sigma^* \mid w_1 = 1\}$$
$$\textsc{Parity} = \{w \in \Sigma^* \mid w \text{ has an odd number of 1s}\}$$

The data generation for FIRST and PARITY is implemented in the same way as in Chiang and Cholak (2022). For a given n we create a sequence of length n containing $\{0, 1\}$ uniformly at random and prepend a CLS token. Then the label is assigned depending on whether the resulting sequence is in the language or not. This is a reasonable method of generating data for these regular languages as a uniform sequence of $\{0, 1\}$ has a 50% chance of being in either language and contains all inputs.

B.2 ONE

$$\textsc{One} = \{w \in \Sigma^* \mid w \text{ contains exactly one } 1\}$$

As it turns out, the method of generating sequences uniformly at random as above is ill-suited to our newly added languages ONE and PALINDROME. This is because generating a sequence uniformly at random for ONE would result in very few examples that would belong to the language. Therefore we decided on fixing the number of ones in the sequence according to a Poission distribution with $\lambda = 1.5$ (see Figure 6) and then selecting their *positions* uniformly at random. This results in a good percentage of examples which are in the language but also includes examples with multiple ones, where the likelihood of larger numbers of ones is ever decreasing.

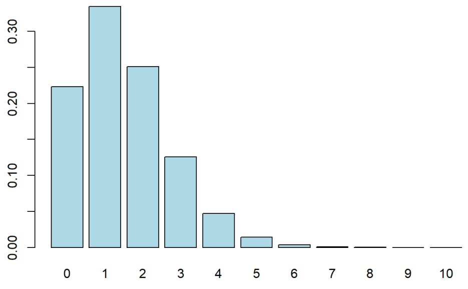


Figure 6: Poisson Distribution with $\lambda = 1.5$ (https://www.est.uc3m.es)

B.3 PALINDROME

$$\textsc{Palindrome} = \{w \in \Sigma^* \mid w \text{ is palindrome}\}$$

The same issue arises for PALINDROME with generating sequences uniformly at random, since it is extremely unlikely for a random sequence of ones and zeros to be a palindrome. Therefore, we use the following strategy:

For a specific n we generate a uniform random sequence of length $\lfloor \frac{n}{2} \rfloor$. If n is even, we just concatenate the reverse of the sequence to itself. If n is odd, we additionally insert another random symbol of the language between the sequence and its reversal. This results in sequences which are in the language PALINDROME. As palindromes are so rare, arbitrary sequences for non-palindromes would make up almost all of the training data, making it even harder for a transformer to learn the language. Hence we instead decided that for counter examples we create palindromic sequences, and then just flip any one of the characters (except the middle one for odd n). This results in negative examples that are very similar to the positive ones, with the fractional difference decreasing linearly with the sequence length.

C Float Precision and PALINDROME

Here we give a short demonstration of why our exact solution for PALINDROME does not work for arbitrarily large sequence lengths n. Floating point precision errors happen when fixed precision floating point numbers are used for calculations, as is the case for *PyTorch*. This is because floating point numbers are represented in scientific notation, with a number between (1 and 10) with a fixed number of decimal places, and a factor of the order of magnitude. When summing numbers of different orders of magnitude, the resulting number will include all the decimal places of the larger number, but fewer (or even none) of the decimal places of the smaller number. For illustration, assume we have 5 significant bits for the first part. Then the following sum will neglect the second summand:

$$\begin{aligned} 1.2345 \times 10^8 + 6.7890 \times 10^3 &= 123456789 \\ &\approx 1.2345 \times 10^8 \end{aligned}$$

This type of issue also causes problems for our exact PALINDROME implementation as we demonstrate with a small piece of sample code (`floating_point_error.py`). The sample creates a tensor with a singular floating point number and then sequentially calculates

$$\sum_{i=0}^{n} -\frac{2^i}{2^{n+1}-1} + \sum_{i=n}^{0} \frac{2^i}{2^{n+1}-1}$$

Similarly to how our exact solution calculates the value **s**. Here n represents the length of the sequence, and the summation should clearly be 0. However, because floating point numbers in *PyTorch* have fixed number of significant bits, we get inaccurate results for large n where the summands with smaller powers of i will not contribute to the sum. Additionally, the order of summation also gives different results. Both contribute to the fact that for large n we get both type 1 and type 2 errors for our exact solution.

D Learning curves for ONE, PALINDROME and PARITY

Figures 7 and 8 show the learning curves for the PARITY and PALINDROME languages, respectively.

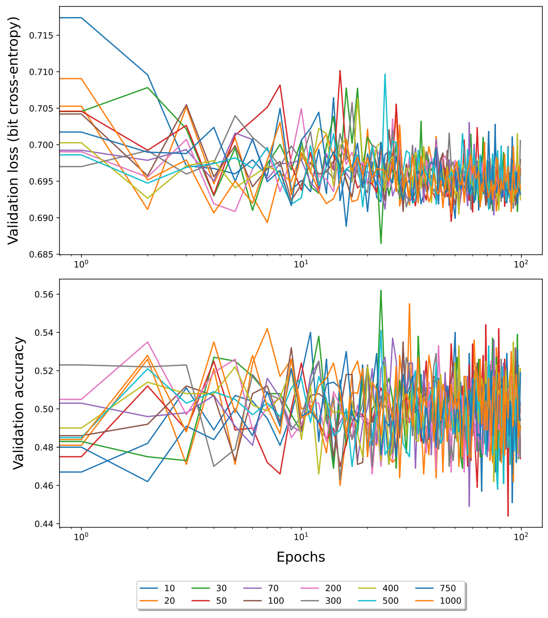


Figure 7: Learning of PARITY, trained on different string lengths for 100 epochs..

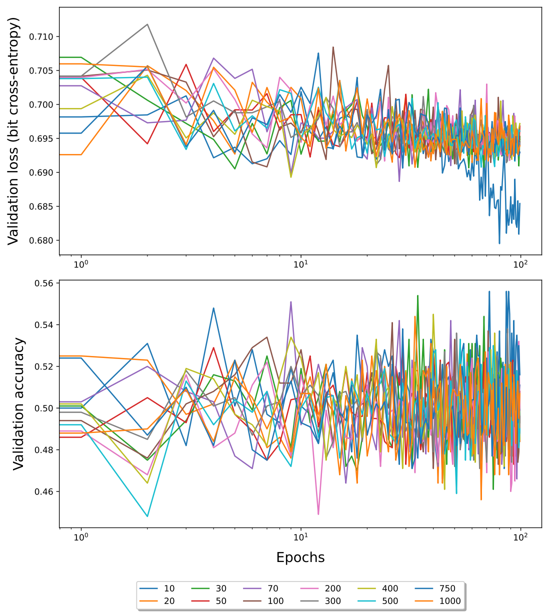


Figure 8: Learning of PALINDROME, trained on different string lengths for 100 epochs..