Overcoming a Theoretical Limitation of Self-Attention

Anonymous ACL submission

Abstract

Although transformers are remarkably effective for many tasks, there are some surprisingly easy-looking regular languages that they struggle with. Hahn shows that for languages where acceptance depends on a single input symbol, a transformer's classification decisions get closer and closer to random guessing (that is, a cross-entropy of 1) as input strings get longer and longer. We examine this limitation using two languages: PARITY, the language of bit strings with an odd number of 1s, and FIRST, the language of bit strings starting with a 1. We demonstrate three ways of overcoming the limitation implied by Hahn's lemma. First, we settle an open question by constructing a transformer that recognizes PARITY with perfect accuracy, and similarly for FIRST. Second, we use layer normalization to bring the cross-entropy of both models arbitrarily close to zero. Third, when transformers need to focus on a single position, as for FIRST, we find that they can fail to generalize to longer strings; we offer a simple remedy to this problem that also improves length generalization in machine translation.

1 Introduction

Although transformers (Vaswani et al., 2017) are remarkably effective for many tasks, there are some surprisingly easy-looking formal languages that they struggle with. Hahn (2020) tries to explain some of these by showing (his Lemma 5) that changing a single input symbol only changes the output of a transformer encoder by $O(1/n)$, where n is the input string length. Thus, for a language where acceptance depends on a single input symbol, the loss difference between a correct and incorrect decision approaches zero as string length increases. So even a transformer with perfect accuracy might, in some sense, not be very good, and, in any case, it might be hard to learn.

Here, we examine this limitation using two simple regular languages:

$$\mathsf{PARITY} = \{w \in \Sigma^* \mid w \text{ has an odd number of 1s}\}$$
$$\mathsf{FIRST} = \{w \in \Sigma^* \mid w_1 = 1\}$$

where (here and throughout the paper) $\Sigma = \{0, 1\}$. Hahn's lemma applies to PARITY because the network must attend to all the symbols of the string, and a change in any one of them changes the correct answer. We have chosen FIRST as one of the simplest examples of a language that the lemma applies to. It only requires attention on the first symbol, but the lemma still applies because a change in this symbol changes the correct answer.

Although the lemma might be interpreted as limiting the ability of transformers to recognize these languages, we show three ways that this limitation can be overcome.

First, we show by explicit constructions that transformers do in fact exist that can perfectly recognize both languages. We have implemented these constructions and verified their accuracy experimentally (§2).

As predicted by Hahn's lemma, our constructed transformers have cross-entropy that approaches 1 bit (that is, just barely better than random guessing) as input length increases. But we show that by adding layer normalization, the cross-entropy can be made arbitrarily close to zero, independent of string length (§3).

In practice, we find, like Bhattamishra et al. (2020), that transformers cannot learn PARITY. Perhaps more surprisingly, when learning FIRST, transformers can have difficulty generalizing from shorter strings to longer strings. Although this is not a logical consequence of Hahn's lemma, it is a consequence of the behavior that Hahn's lemma predicts. Fortunately, this problem can be fixed with a simple modification, multiplying attention logits by $\log n$. This modification also improves length generalization in machine translation (§4).

2 Exact Solutions

The first way to overcome the limitation suggested by Hahn's lemma is to show by explicit construction that our two languages can in fact be recognized with perfect accuracy by transformers.

In this and later sections, we use the following notation frequently. If ϕ is a true-or-false statement, we write

$$\mathbb{I}[\phi] = \begin{cases} 1 & \text{if } \phi \text{ is true} \\ 0 & \text{otherwise.} \end{cases}$$

We define scaled dot-product attention as:

$$\text{Att}\colon \mathbb{R}^d \times \mathbb{R}^{n\times d} \times \mathbb{R}^{n\times d} \to \mathbb{R}^d$$

$$\text{Att}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = \left(\text{softmax}\,\frac{\mathbf{Kq}}{\sqrt{d}}\right)\mathbf{V}.$$

For any m and n, we write $\mathbf{0}^{m\times n}$ for the $m \times n$ zero matrix and $\mathbf{I}^{n\times n}$ for the $n \times n$ identity matrix.

2.1 FFNN for PARITY

Rumelhart et al. (1986) showed that for any n, there is a feedforward neural network (FFNN) that computes PARITY for strings of length exactly n. They also showed that a randomly initialized FFNN can learn to do this automatically.

Since our construction is partially based on theirs, it may be helpful to review their construction in detail. Let w be the input string, $|w| = n$, and k be the number of 1s in w. The input is a vector $\mathbf{x}$ such that $\mathbf{x}_i = \mathbb{I}[w_i = 1]$. The first layer computes k and compares it against $1, 2, \ldots, n$:

$$\mathbf{W}^1 = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} \qquad \mathbf{b}^1 = \begin{bmatrix} -0.5 \\ -1.5 \\ \vdots \\ -n + 0.5 \end{bmatrix}$$

so that

$$\mathbf{h}^1 = H(\mathbf{W}^1\mathbf{x} + \mathbf{b}^1) = \begin{bmatrix} \mathbb{I}[k \geq 1] \\ \mathbb{I}[k \geq 2] \\ \vdots \\ \mathbb{I}[k \geq n] \end{bmatrix}$$

where H is the step function ($H(x) = \mathbb{I}[x > 0]$), applied elementwise.

The second layer adds up the odd elements and subtracts the even elements:

$$\mathbf{W}^2 = \begin{bmatrix} 1 & -1 & \cdots & (-1)^{n+1} \end{bmatrix} \qquad \mathbf{b}^2 = -0.5$$

Then $y = H(\mathbf{W}^2\mathbf{h}^1 + \mathbf{b}^2)$ is 1 if k is odd and 0 is k is even.

2.2 Transformer for PARITY

Following Hahn (2020), we consider transformer encoders with a sigmoid output layer on a single position. More precisely, given a string w, we assume that the input to the network is $\texttt{CLS} \cdot w$, where CLS (for "classification") is a special symbol commonly used for classification tasks. The network linearly projects the encoding of CLS to a scalar and applies a sigmoid function, and it accepts w iff the output probability is greater than $\frac{1}{2}$.

Proposition 1. *There is a transformer encoder with sigmoid output layer that recognizes (in the above sense) the language PARITY.*

Initially, we construct a transformer encoder without layer normalization, then will show how to add layer normalization (§3). Let $n = |w| + 1$, let $w_0 = \texttt{CLS}$, and let w_i be the i-th symbol of w. Let k be the number of occurrences of 1 in w. All vectors computed by the network have 9 dimensions; if we show fewer dimensions, assume the remaining dimensions to be zero.

The word and position embeddings are:

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

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

Since we are numbering positions starting from 0, dimension 4 ranges from 0 to $\frac{n-1}{n}$, and dimension 5 is +1 for even positions and −1 for odd positions.

We argue that dimension 5, being a cosine wave, is a fairly standard choice, although its period (2) is shorter than the shortest period in standard sinusoidal encodings (2π). Dimension 4 is admittedly not standard; however, we argue that it is a reasonable encoding, and extremely easy to compute.

Thus, the encoding of word w_i is:

$$\mathbf{x}^i = \text{WE}(w_i) + \text{PE}(i) = \begin{bmatrix} \mathbb{I}[w_i = \texttt{0}] \\ \mathbb{I}[w_i = 1] \\ \mathbb{I}[w_i = \texttt{CLS}] \\ \frac{i}{n} \\ \cos i\pi \end{bmatrix}.$$

The first self-attention layer has a single head, which finds k, the number of 1s. More precisely,

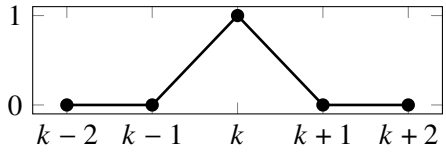

Figure 1: Piecewise linear function equivalent on the integers to $\mathbb{I}[i = k]$.

because attention always averages, it must compute the "average" number of 1s, that is, $\frac{k}{n}$, and stores it in dimension 6. It also stores $\frac{1}{n}$ in dimension 7, which we will need later.

$$\mathbf{W}^{1,1,\mathrm{Q}} = \mathbf{0}$$
$$\mathbf{W}^{1,1,\mathrm{K}} = \mathbf{0}$$
$$\mathbf{W}^{1,1,\mathrm{V}} = \begin{bmatrix} & \mathbf{0}^{5\times 5} & \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

The second head doesn't do anything ($\mathbf{W}^{1,2,\mathrm{V}} = \mathbf{0}$; the queries and keys can be anything). After the residual connection, we have:

$$\mathbf{h}^{1,i} = \mathrm{Att}\Big(\mathbf{W}^{1,1,\mathrm{Q}}\mathbf{x}^i, \begin{bmatrix}\mathbf{W}^{1,1,\mathrm{K}}\mathbf{x}^1 & \dots & \mathbf{W}^{1,1,\mathrm{K}}\mathbf{x}^n\end{bmatrix}^\top, \begin{bmatrix}\mathbf{W}^{1,1,\mathrm{V}}\mathbf{x}^1 & \dots & \mathbf{W}^{1,1,\mathrm{V}}\mathbf{x}^n\end{bmatrix}^\top\Big)$$
$$= \begin{bmatrix} \mathbb{I}[w_i = \mathtt{0}] \\ \mathbb{I}[w_i = \mathtt{1}] \\ \mathbb{I}[w_i = \mathtt{CLS}] \\ \frac{i}{n} \\ \cos i\pi \\ \frac{k}{n} \\ \frac{1}{n} \end{bmatrix}.$$

In the construction of Rumelhart et al. (1986), the next step is to compute $\mathbb{I}[i \le k]$ for each i, using step activation functions. There are two differences in our construction. First, we have ReLU activation functions, not step activation functions. Second, because attention must sum to one, if n is odd then the even and odd positions will get different attention weights, so the trick of subtracting even positions from odd positions will not work. Instead, we want to compute $\mathbb{I}[i = k]$ (Fig. 1).

The first FFNN has two layers. The first is:

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

This gives:

$$\mathbf{h}^{1½,i} = \max\left(0, \mathbf{W}^{2,1}\mathbf{h}^{1,i} + \mathbf{b}^{2,1}\right)$$
$$= \frac{1}{n}\begin{bmatrix} \max(0, k-i-1) \\ \max(0, k-i) \\ \max(0, k-i+1) \end{bmatrix}.$$

The second layer linearly combines these three values to get $\mathbb{I}[i = k]$ as desired.

$$\mathbf{W}^{2,2} = \begin{bmatrix} & \mathbf{0}^{7\times 3} & \\ 1 & -2 & 1 \end{bmatrix} \qquad \mathbf{b}^{2,2} = \mathbf{0}.$$

After the residual connection, we have:

$$\mathbf{h}^{2,i} = \mathbf{W}^{2,2}\mathbf{h}^{1½,i} + \mathbf{b}^{2,2} + \mathbf{h}^{1,i}$$
$$= \begin{bmatrix} \mathbb{I}[w_i = \mathtt{0}] \\ \mathbb{I}[w_i = \mathtt{1}] \\ \mathbb{I}[w_i = \mathtt{CLS}] \\ \frac{i}{n} \\ \cos i\pi \\ \frac{k}{n} \\ \frac{1}{n} \\ \frac{\mathbb{I}[i=k]}{n} \end{bmatrix}.$$

The second self-attention layer tests whether position k is even or odd. It does this using two heads, one which attends more strongly to the odd positions, and one which attends more strongly to the even positions; both average dimension 8:

$$\mathbf{W}^{3,1,\mathrm{Q}} = \begin{bmatrix} 0 & 0 & c\sqrt{d} & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$
$$\mathbf{W}^{3,1,\mathrm{K}} = \begin{bmatrix} 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{bmatrix}$$
$$\mathbf{W}^{3,1,\mathrm{V}} = \begin{bmatrix} & & & \mathbf{0}^{8\times 8} & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\mathbf{W}^{3,2,\mathrm{Q}} = \begin{bmatrix} 0 & 0 & c\sqrt{d} & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$
$$\mathbf{W}^{3,2,\mathrm{K}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$
$$\mathbf{W}^{3,2,\mathrm{V}} = \begin{bmatrix} & & & \mathbf{0}^{8\times 8} & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

where $c > 0$ can be any constant. Then

$$\mathbf{h}^{3,h,i} = \mathrm{Att}\Big(\mathbf{W}^{3,h,\mathrm{Q}}\mathbf{h}^{2,i}, \begin{bmatrix}\mathbf{W}^{3,h,\mathrm{K}}\mathbf{h}^{2,1} & \dots & \mathbf{W}^{3,h,\mathrm{K}}\mathbf{h}^{2,n}\end{bmatrix}^\top, \begin{bmatrix}\mathbf{W}^{3,h,\mathrm{V}}\mathbf{h}^{2,1} & \dots & \mathbf{W}^{3,h,\mathrm{V}}\mathbf{h}^{2,n}\end{bmatrix}^\top\Big)$$
$$\mathbf{h}^{3,i} = \mathbf{h}^{3,1,i} + \mathbf{h}^{3,2,i} + \mathbf{h}^{2,i}.$$

The second FFNN doesn't do anything ($\mathbf{W}^{4,1} = \mathbf{b}^{4,1} = \mathbf{W}^{4,2} = \mathbf{b}^{4,2} = \mathbf{0}$). The vector at position 0

is then

$$\mathbf{h}^{4,n} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ \frac{k}{n} \\ \frac{1}{n} \\ \frac{\mathbb{I}[k=0]}{n} \\ s \end{bmatrix}.$$

If n is even,

$$s = (-1)^{k+1} \cdot \frac{2}{n} \cdot \frac{\exp c - \exp(-c)}{\exp c + \exp(-c)}$$

which is positive if k is odd and negative if k is even. As predicted by Hahn, it is in $O(1/n)$. If n is odd, the expression for s is somewhat more complicated, but it is still positive iff k is odd, and it is still in $O(1/n)$.

Finally, the output layer is a sigmoid layer that just looks at dimension 9:

$$\mathbf{W}^5 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{b}^5 = \mathbf{0}$$

$$y = \sigma(s) = \frac{1}{1 + \exp(-s)}.$$

So the output is greater than $\frac{1}{2}$ iff k is odd.

2.3 Transformer for FIRST

Next, we construct a transformer for FIRST. In line with the common practice of learning per-position word embeddings (Gehring et al., 2017), we use position embeddings that test whether a word is at position 1:

$$\mathbf{x}^i = \begin{bmatrix} \mathbb{I}[w_i = \texttt{0}] \\ \mathbb{I}[w_i = 1] \\ \mathbb{I}[w_i = \texttt{CLS}] \\ \mathbb{I}[i = 1] \end{bmatrix}.$$

The first self-attention layer does nothing ($\mathbf{W}^{1,1,\mathrm{V}} = \mathbf{0}$), so after the residual connection, $\mathbf{h}^{1,i} = \mathbf{x}^i$.

The first FFNN computes a new component (5) that tests whether $i = 1$ and $w_1 = \texttt{1}$:

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

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

so that

$$\mathbf{h}^{2,i} = \begin{bmatrix} \mathbb{I}[w_i = \texttt{0}] \\ \mathbb{I}[w_i = 1] \\ \mathbb{I}[w_i = \texttt{CLS}] \\ \mathbb{I}[i = 1] \\ \mathbb{I}[w_i = 1 \wedge i = 1] \end{bmatrix}.$$

(We have chosen $\mathbf{W}^{2,1}$ in a slightly unusual way in order to avoid using the bias term, in anticipation of §3 when we will add layer normalization.)

The second self-attention layer has a single head, which makes CLS focus on position 1.

$$\mathbf{W}^{3,1,\mathrm{Q}} = \begin{bmatrix} 0 & 0 & c\sqrt{d} & 0 & 0 \end{bmatrix}$$

$$\mathbf{W}^{3,1,\mathrm{K}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{W}^{3,1,\mathrm{V}} = \begin{bmatrix} & & \mathbf{0}^{5\times 5} & & \\ 0 & 0 & 0 & -\frac{1}{2} & 1 \end{bmatrix}$$

where $c > 0$ is a constant. The second FFNN doesn't do anything ($\mathbf{W}^{4,1} = \mathbf{b}^{4,1} = \mathbf{W}^{4,2} = \mathbf{b}^{4,2} = \mathbf{0}$). So at position 0,

$$\mathbf{h}^{4,n} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ \frac{\exp c}{\exp c + n - 1} \left(\mathbb{I}[w_1 = 1] - \frac{1}{2} \right) \end{bmatrix}.$$

The final output layer just selects component 6:

$$\mathbf{W}^5 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{b}^5 = 0.$$

So the output probability is greater than $\frac{1}{2}$ iff $w_1 = 1$. However, it will get closer to $\frac{1}{2}$ as n increases.

2.4 Experiments

We implemented both of the above constructions using modified versions of PyTorch's built-in implementation of transformers (Paszke et al., 2017). These constructions achieve perfect accuracy for strings with lengths sampled from $[1, 1000]$.

However, in Figure 2, the red curves ("no layer norm") show that, as strings grow longer, the cross-entropy approaches 1 bit (that is, barely better than random guessing). We discuss this problem in the next section.

3 Layer Normalization

The second way to mitigate or eliminate the limitation of Hahn's lemma is layer normalization (Ba

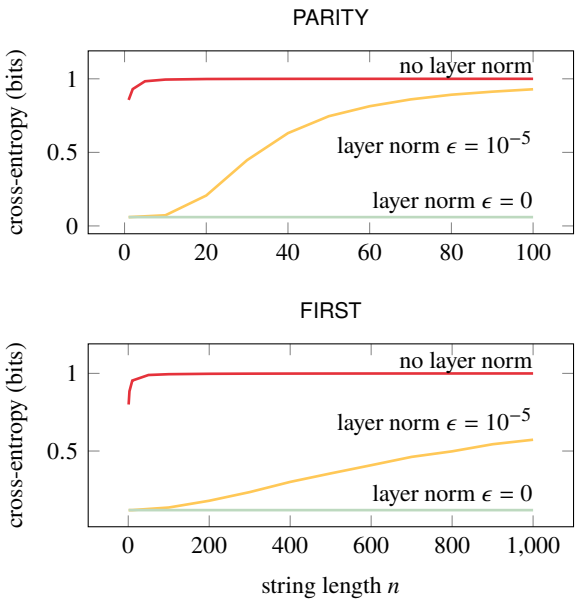

Figure 2: Cross-entropy of exact solutions for PARITY and FIRST computed over 1000 random strings of length n. Without layer norm, the cross-entropy quickly approaches its upper bound of one. With layer norm and $\epsilon > 0$, the cross-entropy is better but still grows with n. With $\epsilon = 0$, cross-entropy is independent of n.

et al., 2016), which is defined, for any vector $\mathbf{x}$, as

$$\mathrm{LN}(\mathbf{x}; \gamma, \beta) = \frac{\mathbf{x} - \mathrm{mean}(\mathbf{x})}{\sqrt{\mathrm{var}(\mathbf{x}) + \epsilon}} \circ \gamma + \beta$$

where the functions mean and var compute the mean and variance, respectively, of the elements of $\mathbf{x}$, and $\circ$ is the elementwise (Hadamard) product. We fix $\beta = 0$ and $\gamma = 1$, so that the result has approximately zero mean and unit variance. The constant ϵ was not present in the original definition (Ba et al., 2016) but is added in all implementations that we are aware of, for numerical stability.

The original transformer definition performs layer normalization immediately after every residual connection.[1] In this section, we modify our two constructions above to use layer normalization. This modification has two steps.

3.1 Removing recentering

The first is to nullify the recentering effect of layer normalization by making the network compute each value x as well as its negation $-x$. The new word encodings are defined in terms of those in the original construction:

$$\bar{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ \mathbf{0} \end{bmatrix}.$$

Likewise for the self-attention parameters:

$$\bar{\mathbf{W}}^{\ell,h,\mathrm{Q}} = \begin{bmatrix} \mathbf{W}^{\ell,h,\mathrm{Q}} & \mathbf{0} \end{bmatrix}$$
$$\bar{\mathbf{W}}^{\ell,h,\mathrm{K}} = \begin{bmatrix} \mathbf{W}^{\ell,h,\mathrm{K}} & \mathbf{0} \end{bmatrix}$$
$$\bar{\mathbf{W}}^{\ell,h,\mathrm{V}} = \begin{bmatrix} \mathbf{W}^{\ell,h,\mathrm{V}} & \mathbf{0} \\ -\mathbf{W}^{\ell,h,\mathrm{V}} & \mathbf{0} \end{bmatrix}.$$

Likewise for the position-wise FFNN parameters:

$$\bar{\mathbf{W}}^{\ell,1} = \begin{bmatrix} \mathbf{W}^{\ell,1} & \mathbf{0} \end{bmatrix} \qquad \bar{\mathbf{b}}^{\ell,1} = \mathbf{b}^{\ell,1}$$
$$\bar{\mathbf{W}}^{\ell,2} = \begin{bmatrix} \mathbf{W}^{\ell,2} \\ -\mathbf{W}^{\ell,2} \end{bmatrix} \qquad \bar{\mathbf{b}}^{\ell,2} = \begin{bmatrix} \mathbf{b}^{\ell,2} \\ -\mathbf{b}^{\ell,2} \end{bmatrix}.$$

Then each layer of activations is (before layer normalization)

$$\bar{\mathbf{h}}^{\ell,i} = \begin{bmatrix} \mathbf{h}^{\ell,i} \\ -\mathbf{h}^{\ell,i} \end{bmatrix}.$$

which always has zero mean, so that layer normalization does not add or subtract anything. It does scale the activations, but the two transformers constructed above are scale-invariant in the sense that any activation layer can be scaled by any positive number without changing the final decisions.

3.2 Reducing cross-entropy

Furthermore, in any transformer, we can use layer normalization to shrink the cross-entropy as small as we like, contrary to Hahn's Lemma 5. In Hahn's formulation, position-wise functions like layer normalization can be subsumed into his f^{act}, but the lemma assumes that f^{act} is Lipschitz-continuous, and layer normalization with $\epsilon = 0$ is not.

Proposition 2. *For any transformer T with layer normalization ($\epsilon = 0$) that recognizes a language $\mathcal{L}$, and for any $\eta > 0$, there is a transformer with layer normalization that recognizes $\mathcal{L}$ with cross-entropy at most η.*

Proof. Let d be the number of dimensions in the original vectors of activations, and let L be the index of the output layer (above, $L = 5$). Then we add a new layer whose self-attention doesn't do

[1] It is also common to place layer normalization before residual connections (Wang et al., 2019; Nguyen and Salazar, 2019), but we follow the original transformer definition here.

anything ($\mathbf{W}^{L,h,\mathrm{V}} = \mathbf{0}$) and whose FFNN is:

$$\bar{\mathbf{W}}^{L,1} = \begin{bmatrix} \mathbf{I}^d \\ -\mathbf{I}^d \end{bmatrix} \qquad \bar{\mathbf{b}}^{L,1} = \begin{bmatrix} \mathbf{0}^d \\ \mathbf{0}^d \end{bmatrix}$$

$$\bar{\mathbf{W}}^{L,2} = \begin{bmatrix} -\mathbf{I}^d & \mathbf{I}^d \end{bmatrix} + \begin{bmatrix} \mathbf{W}^L & -\mathbf{W}^L \\ -\mathbf{W}^L & \mathbf{W}^L \\ \mathbf{0}^{(d-2)\times d} & \mathbf{0}^{(d-2)\times d} \end{bmatrix}$$

$$\bar{\mathbf{b}}^{L,2} = \begin{bmatrix} \mathbf{b}^L \\ -\mathbf{b}^L \\ \mathbf{0}^{d-2} \end{bmatrix}.$$

This causes the residual connection to zero out all dimensions except two, so that if s was the original output logit, the output of this new layer (before layer normalization) is

$$\bar{\mathbf{h}}^{L,i} = \begin{bmatrix} s \\ -s \\ \mathbf{0}^{d-2} \end{bmatrix}.$$

Now, if $\epsilon = 0$, layer normalization scales this vector to have unit variance exactly, so it becomes

$$\mathrm{LN}(\bar{\mathbf{h}}^{L,i}) = \begin{bmatrix} \pm\sqrt{d/2} \\ \mp\sqrt{d/2} \\ \mathbf{0}^{d-2} \end{bmatrix}.$$

The new output layer simply selects the first dimension, scaling it by c:

$$\bar{\mathbf{W}}^{L+1} = \begin{bmatrix} c & 0 & \mathbf{0}^{d-2} \end{bmatrix} \qquad \bar{\mathbf{b}}^{L} = 0.$$

Finally, set $c = -\frac{1}{\sqrt{d/2}} \log(\exp \eta - 1)$. If the input string is in $\mathcal{L}$, then the cross-entropy is $\log \sigma(c\sqrt{d/2}) = \eta$. Similarly, if the input string is not in $\mathcal{L}$, then the cross-entropy is $\log(1 - \sigma(-c\sqrt{d/2})) = \eta$. □

However, in practice, ϵ is always set to a nonzero value, which makes layer normalization Lipschitz-continuous, so Hahn's Lemma 5 still applies.

3.3 Experiments

We tested our exact solutions, modified as described above to use layer normalization. Figure 2 shows that layer normalization with $\epsilon > 0$ improves the cross-entropy, but it still grows with n and approaches 1. With $\epsilon = 0$, the cross-entropy is independent of n and, as argued above (Proposition 2), can be made as low as desired.

4 Learnability

In this section, we turn to the question of learnability, which will lead to a third way of overcoming the limitation suggested by Hahn's lemma.

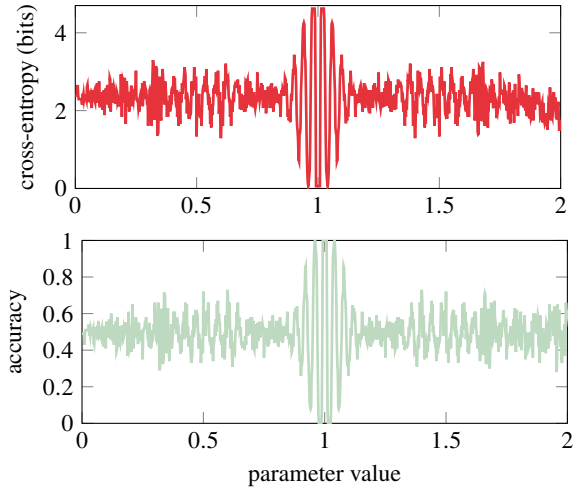

Figure 3: The cross-entropy and accuracy of our solution to PARITY are both extremely sensitive to the parameter $[\bar{\mathbf{W}}^{1,1,\mathrm{V}}]_{6,2}$, which is responsible for computing $\frac{k}{n}$.

4.1 Experiments: standard transformers

We tried training transformers on both PARITY and FIRST. We used transformers with the same number of layers and heads as the corresponding exact solution. We used $d_{\text{model}} = 16$ dimensions for word encodings and for self-attention and FFNN outputs and $d_{\text{FFNN}} = 64$ dimensions for FFNN hidden layers. We used layer normalization after residual connections. We used PyTorch's default initialization and trained using Adam (Kingma and Ba, 2015) with learning rate 3×10^{-4} (Karpathy, 2016). We did not use dropout, as it did not seem to help.

We found, like Bhattamishra et al. (2020), that we were unable to get a transformer to learn to recognize PARITY. Figure 3 shows the cross-entropy and accuracy of the model if we start with the solution constructed above (with layer normalization, $\epsilon = 0$) and vary the parameter $[\bar{\mathbf{W}}^{1,1,\mathrm{V}}]_{6,2}$, which is responsible for computing $\frac{k}{n}$. It's not surprising that, although we can perturb the parameters a little bit and the model can recover, it is incapable of learning from scratch.

FIRST is much easier to learn, but the bad news is that the learned transformers do not generalize well to longer sentences. Figure 4, left, shows that when a transformer is trained on shorter strings ($n = 10, 30, 100, 300$) and tested on longer strings ($n = 1000$), the accuracy is not perfect. Indeed, for training $n = 10$, the accuracy is hardly better than random guessing.

4.2 Flawed transformer for FIRST

To explain why, we describe a simpler but worse transformer for FIRST. In our solution above (§2.3), the second self-attention layer attended mostly to the first position, but not totally. It relied on the fact that in the second self-attention layer, the values of the non-first positions ($\mathbf{W}^{3,1,\mathrm{V}}\mathbf{h}^{2,i}$ for $i > 1$) are exactly zero and therefore do not contribute to the output.

But consider the following transformer, which uses only a single layer and does not zero out the values of the non-first positions:

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

The FFNN doesn't do anything ($\mathbf{W}^{2,1} = \mathbf{b}^{2,1} = \mathbf{W}^{2,2} = \mathbf{b}^{2,2} = \mathbf{0}$), and the final output layer just selects component 5. So if k is the total number of 1s, the final logit at position 0 would be

$$s = \frac{\exp c - 1}{\exp c + n - 1}\left(\mathbb{I}[w_1 = 1] - \frac{1}{2}\right) + \frac{1}{\exp c + n - 1}\left(k - \frac{n}{2}\right).$$

If $c > \log(n-1)$, then this is positive iff $w_1 = 1$. But if $c \le \log(n-1)$, the new second term can be big enough to make the model output an incorrect answer. This suggests that if we train a transformer on strings with length up to N, then the learned parameters will be large enough to classify strings of length up to N correctly, but may misclassify strings longer than N.

This explanation is corroborated by the bottom-left graph in Figure 4 shows the combined attention weight of both layers on the first position. Initially, attention weight increases on position 1, and test cross-entropy decreases while test accuracy increases. But the lower the training length n is, the earlier and lower the attention weight plateaus.

4.3 Log-length scaled attention

Fortunately, this problem is easy to fix by scaling the logits of each attention layer by $\log n$, that is, redefining attention as

$$\mathrm{Att}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = \left(\mathrm{softmax}\,\frac{\log n}{\sqrt{d}}\mathbf{K}\mathbf{q}\right)\mathbf{V}. \quad (1)$$

	train all test all	train short test long
baseline	32.6	11.4
scaled	32.5	12.4

Table 1: When training and testing on data with the same length distribution, scaling attention logits has no significant effect on BLEU, but when training on short sentences (≤ 20 tokens) and testing on long sentences (> 20 tokens), scaling helps significantly ($p < 0.01$).

Then taking the model in §4.2 with $c = 1$ gives

$$s = \frac{n-1}{2n-1}\left(\mathbb{I}[w_1 = 1] - \frac{1}{2}\right) + \frac{1}{2n-1}\left(k - \frac{n}{2}\right)$$

which is positive iff $w_1 = 1$. Moreover, scaling is another way to make the cross-entropy low:

Proposition 3. *For any $\eta > 0$ there is a transformer with attention defined as in Eq. (1), and with or without layer normalization, that recognizes FIRST with cross-entropy at most η.*

Proof. Without layer normalization, we can take the model in §2.3 with $c = 1$, which gives

$$s = \frac{n}{2n-1}\left(\mathbb{I}[w_1 = 1] - \frac{1}{2}\right)$$
$$\frac{1}{4} < |s| \le \frac{1}{2}.$$

With layer normalization, we can apply the modification of §3 to nullify the recentering effect of layer normalization. The final logit is

$$\bar{s} = s\left(\frac{1}{6}(1+s^2) + \epsilon\right)^{-\frac{1}{2}} > \frac{1}{4}\left(\frac{5}{24} + \epsilon\right)^{-\frac{1}{2}}.$$

In either case, since the final logit has a lower bound not dependent on n, the output layer weights can be scaled as in the proof of Proposition 2 to make the cross-entropy at most η. □

4.4 Experiments: scaled attention

Figure 4, right column, shows the training of transformers with scaling of attention logits by $\log n$. For all training lengths n, the model is able to learn with perfect test cross-entropy and accuracy.

We see a similar effect on low-resource English-to-Vietnamese machine translation (Table 1), using Witwicky, an open-source implementation of transformers, with all default settings.[2] When train

[2] https://github.com/tnq177/witwicky

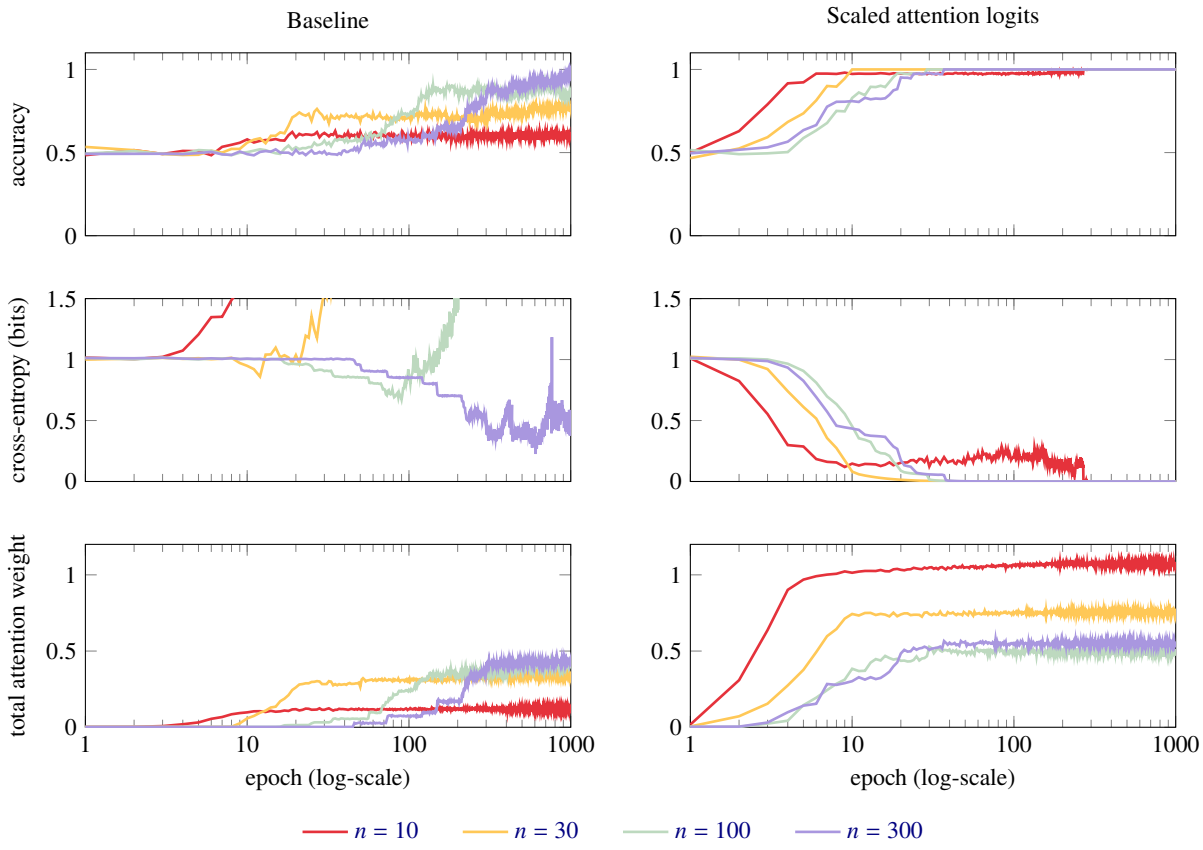

Figure 4: Training a transformer on FIRST. Each epoch has 100 training strings of varying length (see legend) and 100 test strings of length 1000. All curves are averaged over 20 runs. Left: Standard transformer with layer normalization. Right: Attention logits scaled by $\log n$.

and test length distributions are the same, scaling attention logits has no significant effect. But if we train only on sentences with the median length or shorter ($\leq$ 20 tokens) and test only on sentences longer than median length ($>$ 20 tokens), scaling attention logits by $\log n$ helps significantly.

5 Related Work

RASP (Weiss et al., 2021) is a simple programming language in which programs can be compiled into transformers. While the two languages studied here can easily be written in RASP, this does not imply in itself the existence of transformers that can recognize these languages. First, RASP's aggregate operation (which corresponds to attention) always attends uniformly to a subset of positions, unlike softmax attention. Second, RASP's elementwise operations are embedded directly in the output transformer, not translated into FFNNs.

Bhattamishra et al. (2020) carry out theoretical and experimental studies of transformers for various formal languages. The theoretical results are for a different variant of transformers than ours (transformer encoders with self-attention masked so that each position attends only to previous positions), and focus on such transformers' ability to maintain counters that are constrained to be non-negative. Their experimental results suggest that transformers have difficulty learning some regular languages, including PARITY.

6 Conclusion

We have shown that the questions of (a) whether a neural network can recognize a language, (b) whether it can achieve low cross-entropy on a language, and (c) whether it can learn a language are three separate questions – since we have given examples of (a) without (b) and (b) without (c). In particular, since the answer to (b) can hinge on small details of the model, we conclude that it is not very useful as a way of measuring expressivity. Furthermore, we found that although the limited influence of a single input symbol implied by Hahn's lemma can lead to failure to generalize to longer lengths. Our proposed fix, scaling attention logits by $\log n$, is easy to implement and very effective.

References

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

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. 2020. On the ability and limitations of Transformers to recognize formal languages. In *Proc. EMNLP*, pages 7096–7116.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional sequence to sequence learning. In *Proc. ICML*, pages 1243–1252.

Michael Hahn. 2020. Theoretical limitations of self-attention in neural sequence models. *Trans. ACL*, 8:156–171.

Andrej Karpathy. 2016. 3e-4 is the best learning rate for Adam, hands down. Twitter.

Diederik P. Kingma and Jimmy Lei Ba. 2015. Adam: A method for stochastic optimization. In *Proc. ICLR*.

Toan Q. Nguyen and Julian Salazar. 2019. Transformers without tears: Improving the normalization of self-attention. In *Proc. International Workshop on Spoken Language Translation*.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS Workshop on the Future of Gradient-Based Machine Learning Software & Techniques*.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. *Learning Internal Representations by Error Propagation*, pages 318–362. MIT Press, Cambridge, MA, USA.

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 *Proc. NeurIPS*, pages 5998–6008.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019. Learning deep transformer models for machine translation. In *Proc. ACL*, pages 1810–1822.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2021. Thinking like Transformers. In *Proc. ICML*.