Chapter 6

Transformers

6.1 Model

In this section, we define transformers and relevant variants, and how transformers are used to describe formal languages. This section is adapted from Section 4 of the survey by Strobl, Merrill, et al. (2024).

Transformers are composed of an input layer (Section 6.1.1), one or more hidden layers (Section 6.1.4), and an output layer. The inputs and outputs of the layers are sequences of vectors, which we treat as members of $(\mathbb{R}^d)^*$.

6.1.1 Input layer

Strings are initially mapped to sequences of vectors by $emb\colon \Sigma^* \xrightarrow{\text{lp}} (\mathbb{R}^d)^*$, which is the sum of a *word embedding* $\mathrm{WE}\colon \Sigma \to \mathbb{R}^d$ and a *position(al) embedding* or *encoding* $\mathrm{PE}_n\colon [n] \to \mathbb{R}^d$ for $n \in \mathbb{N}_{>0}$:

$$emb(w_0 \cdots w_{n-1})[i] = \mathrm{WE}(w_i) + \mathrm{PE}_n(i). \tag{6.1}$$

In theoretical constructions, the word embedding can be any computable function.

The original transformer paper (Vaswani et al., 2017) introduced the following position embedding:

$$\mathrm{PE}_n(i)[j] = \begin{cases} 10000^{-j/d} \sin i & \text{if } j \text{ even} \\ 10000^{-(j-1)/d} \cos i & \text{if } j \text{ odd.} \end{cases} \tag{6.2}$$

Theoretical papers have explored other position embeddings, including i itself (Pérez et al., 2021), i/n (Yao et al., 2021; Chiang and Cholak, 2022), and $1/i$ or $1/i^2$ (Pérez et al., 2021).

6.1.2 Attention

Scaled dot-product self-attention with d input/output dimensions and d_{hid} key/value dimensions is a function

$$\begin{aligned} att\colon (\mathbb{R}^d)^* &\xrightarrow{\text{lp}} (\mathbb{R}^d)^* \\ \mathbf{X} &\mapsto \mathbf{Y} && (6.3)\\ \mathbf{s}[i,j] &= \frac{\mathbf{W}^{\mathrm{Q}}(\mathbf{X}[i]) \cdot \mathbf{W}^{\mathrm{K}}(\mathbf{X}[j])}{\sqrt{d_{\text{hid}}}} && (6.4)\\ \alpha[i,:] &= \operatorname{softmax}(\mathbf{s}[i,:]) && (6.5)\\ &= \frac{\exp \mathbf{s}[i,:]}{\sum_{j\in[d]} \exp \mathbf{s}[i,j]} && (6.6)\\ \mathbf{Y}[i] &= \sum_{j\in[n]} \alpha[i,j]\, \mathbf{W}^{\mathrm{V}}(\mathbf{X}[j]) && (6.7) \end{aligned}$$

with parameters

$$\begin{aligned} \mathbf{W}^{\mathrm{Q}}, \mathbf{W}^{\mathrm{K}} &\in \mathbb{R}^{d_{\text{hid}}\times d} \\ \mathbf{W}^{\mathrm{V}} &\in \mathbb{R}^{d\times d} \end{aligned}$$

We call the $\mathbf{s}[i,j]$ the *attention scores*, and we call the $\alpha[i,j]$ the *attention weights*.

Real transformers use *multi-head* self-attention, but we don't use it because it can be emulated using single-head self-attention.

Attention masking In *future-masked* (also known as *causally*-masked) self attention, a term $m(i,j)$ is added to Eq. (6.4) to force every position to attend only to preceding positions:

$$m(i,j) = \begin{cases} 0 & \text{if } j \le i \\ -\infty & \text{otherwise.} \end{cases} \tag{6.8}$$

(We define $\exp(-\infty) = 0$.) Some papers use *strict* future-masking, that is, $m(i,j) = 0$ iff $j < i$, and occasionally *past*-masking ($j \ge i$) and strict past-masking ($j > i$).

Hard attention Some theoretical analyses simplify attention by replacing the softmax with variants that focus attention only on the position(s) with the maximum value, breaking ties in various ways.

For any vector $\mathbf{x} \in \mathbb{R}^d$, define $M(\mathbf{x}) = \{i \in [n] \mid \forall j \in [n], \mathbf{x}[j] \le \mathbf{x}[i]\}$ to be the set of indices of the maximal elements of $\mathbf{x}$. In *leftmost*-hard attention, the leftmost maximal element is used, replacing Eq. (6.5) with:

$$\alpha[i,j] = \mathbb{I}[j = \min M(\mathbf{s}[i,:])] \tag{6.9}$$

whereas in *average*-hard attention, the maximal elements share weight equally:

$$\alpha[i,j] = \frac{\mathbb{I}[j \in M(\mathbf{s}[i,:])]}{|M(\mathbf{s}[i,:])|}. \tag{6.10}$$

Leftmost-hard attention was previously called *hard* attention by Hahn (2020) and *unique-hard* attention by Hao et al. (2022). One may also consider rightmost-hard attention, in which the rightmost maximal element is used. Average-hard attention was also called *hard* attention by Pérez et al. (2021) and *saturated* attention by Merrill, Sabharwal, and Smith (2022), and has been argued to be a realistic approximation to how trained transformers behave in practice (Merrill, Ramanujan, et al., 2021). Neither type of hard attention should be confused with the concept of hard attention used in computer vision (e.g., Xu et al., 2015).

6.1.3 Layer normalization

A d-dimensional *layer normalization* (Ba et al., 2016), or *layernorm* for short, is a function

$$\begin{aligned} \mathit{norm}\colon \mathbb{R}^d &\to \mathbb{R}^d \\ \mathbf{x} &\mapsto \gamma \odot \frac{\mathbf{x} - \bar{\mathbf{x}}}{\sqrt{\mathrm{Var}(\mathbf{x}) + \varepsilon}} + \beta \end{aligned} \tag{6.11}$$

where $\odot$ is component-wise multiplication,

$$\bar{\mathbf{x}} = \frac{1}{d} \sum_{i \in [d]} \mathbf{x}_i \tag{6.12}$$

$$\mathrm{Var}(\mathbf{x}) = \frac{1}{d} \sum_{i \in [d]} (\mathbf{x}_i - \bar{\mathbf{x}})^2 \tag{6.13}$$

and the parameters are

$$\begin{aligned} \gamma, \beta &\in \mathbb{R}^d \\ \varepsilon &\geq 0. \end{aligned}$$

The original definition of layernorm (Ba et al., 2016) sets $\varepsilon = 0$, but, for numerical stability, and to avoid division by zero, all implementations we are aware of set $\varepsilon > 0$. Observe that *norm* is Lipschitz-continuous iff $\varepsilon > 0$.

Some transformer analyses omit layernorm for simplicity (e.g. Pérez et al., 2021).

6.1.4 Hidden layers

A *transformer layer* (also known as a *block*) comes in two variants. The *post-norm* variant (Vaswani et al., 2017) is

$$\begin{aligned} \mathit{layer}\colon (\mathbb{R}^d)^* &\xrightarrow{\text{lp}} (\mathbb{R}^d)^* \\ \mathbf{X} &\mapsto \mathbf{Y} \text{ where} \\ \mathbf{H} &= \mathit{norm}_1(\mathbf{X} + \mathit{att}(\mathbf{X})) \\ \mathbf{Y} &= \mathit{norm}_2(\mathbf{H} + \mathit{ffn}(\mathbf{H})) \end{aligned} \tag{6.14}$$

and the *pre-norm* variant (Wang et al., 2019) has

$$\begin{aligned} \mathbf{H} &= \mathbf{X} + \mathit{att}(\mathit{norm}_1(\mathbf{X})) \\ \mathbf{Y} &= \mathbf{H} + \mathit{ffn}(\mathit{norm}_2(\mathbf{H})) \end{aligned} \tag{6.15}$$

where

- *att* is a self-attention with d input/output dimensions and d_{hid} key/value dimensions
- *ffn* is a feed-forward network with d input/output dimensions and two layers, one ReLU and one linear. In applications, the hidden layer usually has $4d$ dimensions, but theoretical constructions use as many or as few dimensions as needed.
- norm_1 and norm_2 are layernorms with d dimensions.

In both variants, the $\mathbf{X}+$ and $\mathbf{H}+$ terms are called *residual connections* (He et al., 2015), also known as *skip connections*.

6.1.5 Transformer encoders

A *post-norm transformer encoder* is a length-preserving function

$$\begin{aligned} \mathit{tfr}\colon \Sigma^* &\xrightarrow{\text{lp}} (\mathbb{R}^d)^* \\ w &\mapsto \mathbf{H}^{(L)} \text{ where} \\ \mathbf{H}^{(0)} &= \mathit{emb}(w) \\ \mathbf{H}^{(1)} &= \mathit{layer}_1(\mathbf{H}^{(0)}) \\ &\vdots \\ \mathbf{H}^{(L)} &= \mathit{layer}_L(\mathbf{H}^{(L-1)}) \end{aligned} \tag{6.16}$$

where

- e is an input layer
- L is called the *depth*
- each $layer_\ell$ for $\ell \in [L]$ is a post-norm transformer layer (6.14).

A *pre-norm* transformer encoder is additionally parameterized by the weights of a final layernorm *norm* and is defined as:

$$\begin{aligned} tfr\colon \Sigma^* &\xrightarrow{\text{lp}} (\mathbb{R}^d)^* \\ w &\mapsto norm(\mathbf{H}^{(L)}) \text{ where} \\ \mathbf{H}^{(0)} &= emb(w) \\ \mathbf{H}^{(1)} &= layer_1(\mathbf{H}^{(0)}) \\ &\vdots \\ \mathbf{H}^{(L)} &= layer_L(\mathbf{H}^{(L-1)}) \end{aligned} \tag{6.17}$$

where

- e is an input layer
- L is called the *depth*
- each $layer_\ell$ for $\ell \in [L]$ is a pre-norm transformer layer (6.15)
- *norm* is a layernorm.

The encoder's output is a sequence of vectors in $(\mathbb{R}^d)^*$. To use it as a language recognizer, we add a linear output layer

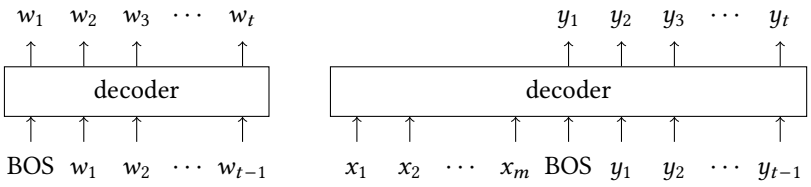
$$\begin{aligned} out\colon \mathbb{R}^d &\to \mathbb{R} \\ \mathbf{h} &\mapsto \mathbf{w} \cdot \mathbf{h} + b \qquad\qquad i \in [n] \end{aligned} \tag{6.18}$$

where parameters $\mathbf{w} \in \mathbb{R}^d$, and $b \in \mathbb{R}$. The encoder accepts iff $out(tfr(w)[n]) \geq 0$.

6.2 Expressivity

6.2.1 Decoders with intermediate steps

Transformer decoders A *transformer decoder* is a transformer encoder *tfr* with future masking in its attention, typically used to generate rather than recognize strings. GPT and its competitor LLMs are all transformer decoders.



(a) without prompt (b) with prompt

Figure 6.1: Generating strings from a transformer decoder.

We assume that Σ contains a special symbol BOS that does not occur anywhere else; later, we will add several other special symbols. The input string is the prefix of previously-generated symbols, $w_{<t} = w_0 \cdots w_{t-1}$, where $w_0 = \text{BOS}$. The output is a probability distribution $\hat{p}(w_t \mid w_{<t})$ over the next symbol,

$$\begin{aligned} out\colon \mathbb{R}^d &\to \mathbb{R} \\ \mathbf{x} &\mapsto \mathbf{W}\mathbf{x} + \mathbf{b} && (6.19) \\ \hat{p}(\cdot \mid w_{<t}) &= \text{softmax}(out(tfr(w_{<t})[t-1])) && (6.20) \end{aligned}$$

where parameters $\mathbf{W} \in \mathbb{R}^{|\Sigma| \times d}$ and $\mathbf{b} \in \mathbb{R}^{|\Sigma|}$.

To sample a string, we first sample w_1 from $\hat{p}(w_1 \mid \text{BOS})$, then, for each time step $t > 1$, sample w_t from $\hat{p}(w_t \mid w_{<t})$. The process stops when $w_t = \text{EOS}$. Because each sampled output symbol becomes part of the input at the next time step, this kind of model is called *autoregressive*. See Fig. 6.1a.

In most (not all) theoretical papers about transformer decoders, we want the decoder to output a single next symbol instead of a probability distribution over next symbols. To do this, we can select the argmax of $out(tfr(w_{<t})[t-1])$ instead. Warning: In general, selecting the argmax at each step does *not* give you the highest-probability string.

We can also provide a *prompt* x to the decoder, which the decoder can see as part of its input but doesn't have to output. In that case, for $t \geq 1$, the input string is $x \cdot \text{BOS} \cdot y_{<t}$, and the output is y_t. See Fig. 6.1b.

Intermediate steps In many applications of transformer decoders, the prompt x is some kind of question, and the desired output y is the answer. For example, $x = \texttt{101*101}$ and $y = \texttt{10201}$. Researchers have found in practice that sometimes a transformer decoder isn't very good at answering certain kinds of questions, but if one allows the decoder to insert a number of *intermediate* time steps between the prompt and the final output, it sometimes performs much better. This is known as a *scratchpad* (Nye et al., 2022) or *chain of thought* (Wei et al., 2022).

Here, we're only interested in the case where the final output is a single symbol, so we have the following definition.

Definition 6.1. Let f be a transformer decoder. For any string $x \in \Sigma^*$, we say that f, on prompt x, outputs y_T after T intermediate steps if there is a string $y = y_1 \cdots y_T$ such that for all $t = 1, \ldots, T$, we have $f(x \cdot \text{BOS} \cdot y_1 \cdots y_{t-1}) = y_t$.

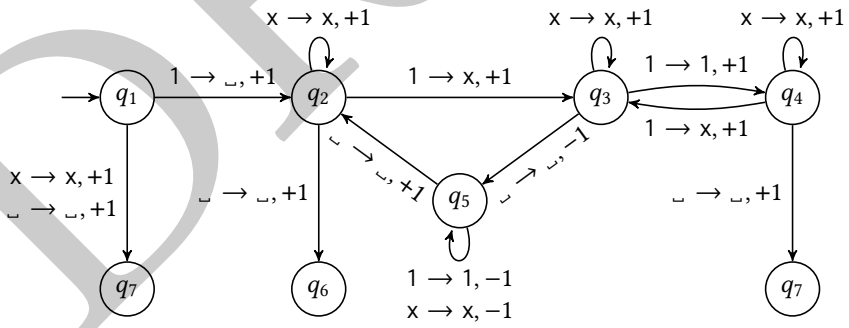
Turing machines We assume that you are familiar with Turing machines. We use the definition of Turing machine in the textbook by Sipser (2013), with one small modification. Just to make sure we're on the same page, we give the barest of definitions here.

Definition 6.2. A Turing machine is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$, where

- Q is a finite set of states
- Σ is a finite input alphabet, where $\sqcup \notin \Sigma$
- Γ is a finite tape alphabet, where $\Sigma \cup \{\sqcup\} \subseteq \Gamma$
- $\delta \colon Q \times \Gamma \to Q \times \Gamma \times \{-1, +1\}$ is the transition function.

The tape has a left end and extends infinitely to the right. On input $w \in \Sigma^*$, the tape is initialized to $w \sqcup \sqcup \cdots$. If the current state is q, the current tape symbol is a, and $\delta(q, a) = (r, b, m)$, then the machine enters state r, writes a b, and moves left if $m = -1$, right if $m = +1$. If the machine enters state q_{accept}, it halts and accepts w; if it enters state q_{reject}, it halts and rejects w.

Example 6.3. Here's an example Turing machine (Sipser, 2013), with $q_{\text{start}} = q_1$, $q_{\text{accept}} = q_6$, $q_{\text{reject}} = q_7$. It decides the language $\{1^{2^n} \mid n \geq 0\}$.



The reject state q_7 appears twice to reduce clutter.

The (not very exciting) run of this machine on string 11 is:

$$
\begin{array}{ll}
q_1 & \hat{1}1\sqcup\cdots \\
q_2 & \sqcup\hat{1}\sqcup\cdots \\
q_3 & \sqcup\hat{\mathsf{x}}\sqcup\cdots \\
q_5 & \sqcup\hat{\mathsf{x}}\sqcup\cdots \\
q_5 & \hat{\sqcup}\mathsf{x}\sqcup\cdots \\
q_2 & \sqcup\hat{\mathsf{x}}\sqcup\cdots \\
q_6 & \text{accept}
\end{array}
$$

Simulating Turing machines

Theorem 6.4. *For any Turing machine M with input alphabet Σ, there is a transformer decoder f with average-hard attention that is equivalent to M in the following sense. For any string $w \in \Sigma^*$:*

- *If M halts and accepts on input w, then there is a T such that f, on prompt w, outputs* ACC *after T intermediate steps.*
- *If M halts and rejects on input w, then there is a T such that f, on prompt w, outputs* REJ *after T intermediate steps.*
- *If M does not halt on input w, then there does not exist a T such that f, on prompt w, outputs either* ACC *or* REJ.

The rest of this section proves the above theorem. There are several related proofs in the literature (Pérez et al., 2021; Bhattamishra et al., 2020; Merrill and Sabharwal, 2024); ours is an amalgam of these.

Let $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$. The alphabet of $f.tfr$ is $\Sigma \cup \{\text{BOS}, \text{ACC}, \text{REJ}\} \cup (Q \times \Gamma \times \{-1, +1\})$. At each time step starting with BOS, the network outputs a triple $(r, b, m) \in (Q \times \Gamma \times \{-1, +1\})$ indicating what the next simulated action of M is.

Each time step $i = 1, 2, \ldots$ of the transformer proceeds as follows.

1. Unpack the current input symbol, $x^{(i)}$:
   - If $x^{(i)} \in \Sigma \cup \text{BOS}$, let $q^{(i)} = \bot$ and $m^{(i-1)} = 0$.
   - Else, let $(q^{(i)}, b^{(i-1)}, m^{(i-1)}) = x^{(i)}$.
2. Compute the head position: $h^{(i)} = \sum_{j=0}^{i-1} m^{(j)}$.
3. Compute the symbol under the head, $a^{(i)}$:

- Find j^*, the rightmost position $j < i$ such that $h^{(j)} = h^{(i)}$.
- If j^* exists and $b^{(j)} \neq \perp$, let $a^{(i)} = b^{(j^*)}$.
- Else, if $x^{(h^{(i)})} \in \Sigma$, let $a^{(i)} = x^{(h^{(i)})}$.
- Else, let $a^{(i)} = \text{␣}$.

4. Compute the next transition:

- If $x^{(i)} \in \Sigma$, just output $y^{(i)} = x^{(i)}$. (It will be ignored anyway.)
- Else, if $x^{(i)} = \text{BOS}$, let $q^{(i+1)} = q_{\text{start}}$, $b^{(i)} = a^{(i)}$, and $m^{(i)} = 0$.
- Else, let $(q^{(i+1)}, b^{(i)}, m^{(i)}) = \delta(q^{(i)}, a^{(i)})$.
- If $q^{(i+1)} = q_{\text{accept}}$ or q_{reject}, output $y^{(i)} = \text{ACC}$ or $y^{(i)} = \text{REJ}$, respectively.
- Else, output $y^{(i)} = (q^{(i+1)}, b^{(i)}, m^{(i)})$.

For example, the run from Example 6.3 is simulated by:

i	tape	$x^{(i)}$	$q^{(i)}$	$b^{(i-1)}$	$m^{(i-1)}$	$h^{(i)}$	$a^{(i)}$	$y^{(i)}$
1		1	$\perp$	$\perp$	0	0	1	1
2		1	$\perp$	$\perp$	0	0	1	1
3		BOS	$\perp$	$\perp$	0	0	1	$(q_1, 1, 0)$
4	$\hat{1}1$␣ $\cdots$	$(q_1, 1, 0)$	q_1	1	0	0	1	$(q_2, \text{␣}, +1)$
5	␣$\hat{1}$␣ $\cdots$	$(q_2, \text{␣}, +1)$	q_2	␣	+1	1	1	$(q_3, \text{x}, +1)$
6	␣x$\hat{\text{␣}}$ $\cdots$	$(q_3, \text{x}, +1)$	q_3	x	+1	2	␣	$(q_5, \text{␣}, -1)$
7	␣$\hat{\text{x}}$␣ $\cdots$	$(q_5, \text{␣}, -1)$	q_5	␣	−1	1	x	$(q_5, \text{x}, -1)$
8	␣$\hat{\text{x}}$␣ $\cdots$	$(q_5, \text{x}, -1)$	q_5	x	−1	0	␣	$(q_2, \text{␣}, +1)$
9	␣$\hat{\text{x}}$␣ $\cdots$	$(q_2, \text{␣}, +1)$	q_2	␣	+1	1	x	ACC

Now we have to construct transformer layers to perform the above steps. Each input vector contains a word embedding, $\mathbf{e}_{x^{(i)}}$, and a position embedding with 4 components:

$$\mathbf{H}^{(0)}[i] = \begin{bmatrix} \mathbf{e}_{x^{(i)}} \\ 1/i \\ 1 \\ i \\ i^2 \end{bmatrix}. \tag{6.21}$$

Step 1 is piecewise linear, so it can be computed by a FFNN (Theorem 3.6). Afterwards, the activation vector at position i is:

$$\mathbf{H}^{(1)}[i] = \begin{bmatrix} \mathbf{e}_{x^{(i)}} \\ 1/i \\ 1 \\ i \\ i^2 \\ \mathbf{e}_{q^{(i)}} \\ \mathbf{e}_{b^{(i-1)}} \\ m^{(i-1)} \end{bmatrix}. \tag{6.22}$$

Step 2 can be computed by a uniform self-attention layer:

$$\mathbf{q}^{(i)} = 0 \qquad \mathbf{k}^{(j)} = 0 \qquad \mathbf{v}^{(j)} = \begin{bmatrix} \mathbf{0} \\ m^{(i-1)} \end{bmatrix}. \tag{6.23}$$

Uniform self-attention computes an average, not a sum. Since at time step i, there are i positions to average over, the result is $h^{(i)}/i$, not $h^{(i)}$. We'll correct this in the next step.

$$\mathbf{H}^{(2)}[i] = \begin{bmatrix} \mathbf{e}_{x^{(i)}} \\ 1/i \\ 1 \\ i \\ i^2 \\ \mathbf{e}_{q^{(i)}} \\ \mathbf{e}_{b^{(i-1)}} \\ m^{(i-1)} \\ h^{(i)}/i \end{bmatrix} \tag{6.24}$$

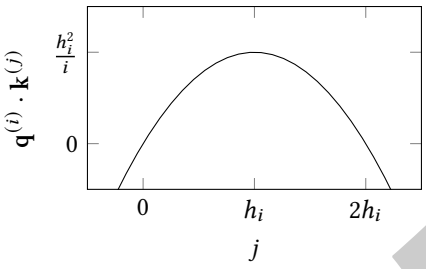
Step 3 is the most difficult step. There are several schemes that have been proposed for this. All of them use average-hard attention, and all of them further modify the transformer in some way: changing dot-product to something else (Pérez et al., 2021), adding components to the position embedding (Pérez et al., 2021; Barceló et al., 2024; Strobl, Angluin, et al., 2024), or applying layer normalization only to selected components (Merrill and Sabharwal, 2024). The construction here is closest to that of Strobl, Angluin, et al. (2024).

It uses two self-attention layers. The first layer changes $h^{(i)}/i$ to $h^{(i)}$ by treating the position embeddings as a lookup table (Barceló et al., 2024).

$$\mathbf{q}^{(i)} = \begin{bmatrix} 2h^{(i)}/i \\ -1/i \end{bmatrix} \qquad \mathbf{k}^{(j)} = \begin{bmatrix} j \\ j^2 \end{bmatrix} \qquad \mathbf{v}^{(j)} = \begin{bmatrix} \mathbf{0} \\ j \\ j^2 \\ \mathbf{e}_{x^{(j)}} \end{bmatrix} \tag{6.25}$$

$$\mathbf{q}^{(i)} \cdot \mathbf{k}^{(j)} = \frac{1}{i} j(2h^{(i)} - j). \tag{6.26}$$

Then the attention scores are uniquely maximized when $j = h^{(i)}$:



So the attention layer outputs $h^{(i)}$, and also some other quantities which we'll need shortly. The activation vector at position i is:

$$\mathbf{H}^{(3.5)}[i] = \begin{bmatrix} \mathbf{e}_{x^{(i)}} \\ 1/i \\ 1 \\ i \\ i^2 \\ \mathbf{e}_{q^{(i)}} \\ \mathbf{e}_{b^{(i-1)}} \\ m^{(i-1)} \\ h^{(i)}/i \\ h^{(i)} \\ (h^{(i)})^2 \\ \mathbf{e}_{x^{(h^{(i)})}} \end{bmatrix}. \tag{6.27}$$

Recall that we need to search positions $j < i$ such that $h^{(j)} = h^{(i)}$. But future-masked attention looks at positions $j \le i$. To get around this, we search positions $j \le i$ such that $h^{(j-1)} = h^{(i)}$. So we will need

$$h^{(i-1)} = h^{(i)} - m^{(i-1)} \tag{6.28}$$

$$(h^{(i-1)})^2 = (h^{(i)})^2 - 2h^{(i)} m^{(i-1)} + (m^{(i-1)})^2 \tag{6.29}$$

$$= \begin{cases} (h^{(i)})^2 + 2h^{(i)} + 1 & \text{if } m^{(i-1)} = -1 \\ (h^{(i)})^2 - 2h^{(i)} + 1 & \text{if } m^{(i-1)} = +1 \end{cases} \tag{6.30}$$

which can both be computed using a FFN. So

$$\mathbf{H}^{(4)}[i] = \begin{bmatrix} \mathbf{e}_{x^{(i)}} \\ 1/i \\ 1 \\ i \\ i^2 \\ \mathbf{e}_{q^{(i)}} \\ \mathbf{e}_{b^{(i-1)}} \\ m^{(i-1)} \\ h^{(i)}/i \\ h^{(i)} \\ (h^{(i)})^2 \\ \mathbf{e}_{x^{(h^{(i)})}} \\ h^{(i-1)} \\ (h^{(i-1)})^2 \end{bmatrix}. \tag{6.31}$$

The second self-attention layer uses another variation of the lookup trick:

$$\mathbf{q}^{(i)} = \begin{bmatrix} 2h^{(i)} \\ -1 \\ \frac{1}{2i} \end{bmatrix} \qquad \mathbf{k}^{(j)} = \begin{bmatrix} h^{(j-1)} \\ (h^{(j-1)})^2 \\ j \end{bmatrix} \qquad \mathbf{v}^{(j)} = \begin{bmatrix} \mathbf{0} \\ h^{(j-1)} \\ \mathbf{e}_{b^{(j-1)}} \end{bmatrix} \tag{6.32}$$

$$\mathbf{q}^{(i)} \cdot \mathbf{k}^{(j)} = \underbrace{h^{(j-1)}(2h^{(i)} - h^{(j-1)})}_{\text{find } h^{(j-1)} = h^{(i)}} + \underbrace{\frac{j}{2i}}_{\text{find rightmost}} \tag{6.33}$$

The first term is maximized when $h^{(j-1)} = h^{(i)}$. If there is more than one such position j, then because of the second term, j^* is the rightmost such j. If there's

no such position j, then $h^{(j^*-1)} \neq h^{(i)}$.

$$\mathbf{H}^{(4.5)}[i] = \begin{bmatrix} \mathbf{e}_{x^{(i)}} \\ 1/i \\ 1 \\ i \\ i^2 \\ \mathbf{e}_{q^{(i)}} \\ \mathbf{e}_{b^{(i-1)}} \\ m^{(i-1)} \\ h^{(i)}/i \\ h^{(i)} \\ (h^{(i)})^2 \\ \mathbf{e}_{x^{(h^{(i)})}} \\ h^{(i-1)} \\ (h^{(i-1)})^2 \\ h^{(j^*-1)} \\ \mathbf{e}_{b^{(j^*-1)}} \end{bmatrix}. \tag{6.34}$$

Finally, we can use the FFNN to set

$$a^{(i)} = \begin{cases} b^{(j^*-1)} & h^{(j^*-1)} = h^{(i)} \text{ and } b^{(j^*-1)} \neq \bot \\ x^{(h^{(i)})} & \text{otherwise.} \end{cases} \tag{6.35}$$

So the activation vectors are:

$$\mathbf{H}^{(5)}[i] = \begin{bmatrix} \mathbf{e}_{x^{(i)}} \\ 1/i \\ 1 \\ i \\ i^2 \\ \mathbf{e}_{q^{(i)}} \\ \mathbf{e}_{b^{(i-1)}} \\ m^{(i-1)} \\ h^{(i)}/i \\ h^{(i)} \\ (h^{(i)})^2 \\ \mathbf{e}_{x^{(h^{(i)})}} \\ h^{(i-1)} \\ (h^{(i-1)})^2 \\ h^{(j^*-1)} \\ \mathbf{e}_{b^{(j^*-1)}} \\ \mathbf{e}_{a^{(i)}} \end{bmatrix}. \tag{6.36}$$

Step 4 is piecewise linear, so it can be computed by a FFNN (Theorem 3.6).

6.2.2 Encoders with unique-hard attention

Unique-hard attention has been studied in several papers (Hahn, 2020; Hao et al., 2022; Barceló et al., 2024). Here, we look at the results of Angluin et al. (2023), which exactly characterize transformers with *rightmost* hard attention and *strict* future masking.

Strict future masking means that each position i attends to positions $j < i$. If i is the leftmost position, all positions are masked out, so (following Merrill and Sabharwal (2024)) the attention output is just the zero vector.

Theorem 6.5. *For any transformer encoder T with rightmost hard attention and strict future masking, there is a closed formula of* FO *that defines the same language that T recognizes.*

The proof hinges on the fact that in a unique hard attention transformer, each activation vector depends on at most two vectors from the layer below. Because the network has fixed, finite depth, there is a fixed, finite number of possible activation vectors that it can compute.

Lemma 6.6. *Let T be a unique (leftmost or rightmost) hard attention transformer. There is a finite set $\mathbb{F} \subseteq \mathbb{R}$ such that for any input string w, all the attention scores and activation values computed by $T(w)$ belong to $\mathbb{F}$.*

Proof. We prove that the self-attention at layer ℓ has at most $(|\Sigma|+1)^{2^\ell}-1$ different possible output vectors, by induction on ℓ.

Base case ($\ell = 0$): Since there are no position embeddings, the embedding at position i is determined entirely by w_i, so there are at most $|\Sigma|$ possible activation vectors.

Inductive step ($\ell > 0$): Assume that the output of the layer ℓ has at most $(|\Sigma|+1)^{2^\ell} - 1$ possible activation vectors, and consider layer $(\ell+1)$:

- The attention output at position i depends only on $\mathbf{H}^{(\ell)}[i]$ (because of the residual connection) and $\mathbf{H}^{(\ell)}[j_i]$ (where j_i is the position that i attends to). So the number of possible output activation vectors is at most

$$\begin{aligned}\left((|\Sigma|+1)^{2^\ell}-1\right)(|\Sigma|+1)^{2^\ell} &\le \left((|\Sigma|+1)^{2^\ell}-1\right)\left((|\Sigma|+1)^{2^\ell}+1\right)\\ &= \left((|\Sigma|+1)^{2^\ell}\right)^2 - 1\\ &= (|\Sigma|+1)^{2^{\ell+1}} - 1.\end{aligned}$$

- Because the FFNN and layernorms operate position-wise, they also have at most $(|\Sigma|+1)^{2^{\ell+1}} - 1$ possible activation vectors.

Therefore, the number of possible output vectors from a self-attention or FFNN at layer ℓ is at most $(|\Sigma|+1)^{2^\ell} - 1$.

Additionally, the attention score from position i to j depends only on $\mathbf{H}^{(\ell)}[i]$ and $\mathbf{H}^{(\ell)}[j]$, so there are at most $\left((|\Sigma|+1)^{2^\ell}-1\right)^2 \leq (|\Sigma|+1)^{2^{\ell+1}}$ possible attention scores.

Then $\mathbb{F}$ is the union over all layers of the possible attention scores and components of the possible activation vectors. □

Although it's not the most efficient way to do it, we can represent an activation value $X(w)$ as a set of FO formulas $(\phi_v)_{v\in\mathbb{F}}$, such that $w \models \phi_v$ iff $X(w) = v$. Angluin et al. (2023) show how to do this with $O(\log|\mathbb{F}|)$ formulas instead.

Definition 6.7. Let $k \geq 0$. A function $X\colon \Sigma^* \times \mathbb{N}^k \to \mathbb{F}$ is *definable* by FO formulas $(\chi_v(x_1,\ldots,x_k))_{v\in\mathbb{F}}$ if for all $w \in \Sigma^*$ and $v \in \mathbb{F}$, we have $X(w, i_1, \ldots, i_k) = v$ iff $w \models \chi_v(i_1,\ldots,i_k)$.

Lemma 6.8. *Let $\mathbb{F} \subseteq \mathbb{R}$ be a finite set. If $X\colon \Sigma^* \to \mathbb{F}$ is definable by* FO *formulas ϕ_v, and $f : \mathbb{F} \to \mathbb{F}$, then there are* FO *formulas ϕ'_v that define $f(X(w))$.*

Proof.

$$\phi'_v = \bigvee_{\substack{u\in\mathbb{F}\\ v=f(u)}} \phi_u. \tag{6.37}$$

□

Hopefully, it is easy to see how the above result generalizes to the cases where X maps to a sequence of activations and/or f is a function of two or more arguments.

Proof of Theorem 6.5. For every activation value $\mathbf{H}^{(\ell)}[i,k](w)$, we will construct a formula $\phi_{\ell,k,v}(x)$ such that $w \models \phi_{\ell,k,v}(i)$ iff $\mathbf{H}^{(\ell)}[i,k](w) = v$. We do this by induction on ℓ.

Case $\ell = 0$:

$$\phi_{0,k,v}(x) = \bigvee_{\substack{a\in\Sigma\\ emb(a)[k]=v}} Q_a(x). \tag{6.38}$$

Case $\ell > 0$: By Lemma 6.8, there are formulas $\text{score}_{\ell,v}(i,j)$ that define $\mathbf{s}^{(\ell)}[i,j]$, the attention score at layer ℓ from position i to j. Furthermore, there is a formula $\text{score}_{\ell,\leq}(i,j_1,j_2)$ that holds iff $\mathbf{s}^{(\ell)}[i,j_1] \leq \mathbf{s}^{(\ell)}[i,j_2]$ and formulas $\text{value}_{\ell,k,v}(j)$ that define $\mathbf{W}^{\mathrm{V}}(\mathbf{X}[j])$, the value at position j. Then we can define a formula that tests whether position x attends to position y:

$$\begin{aligned}\text{weight}_\ell(x,y) = y < x \wedge \forall y'[&(y' < y \to \text{score}_{\ell,\leq}(x,y',y))\\ &\wedge (y < y' < x \to \neg\text{score}_{\ell,\leq}(x,y,y'))].\end{aligned} \tag{6.39}$$

and formulas that define $\mathbf{Y}[x][k]$, the k-th component of the attention output at position x:

$$\mathrm{att}_{\ell,k,v}(x) = \begin{cases} \exists y[\mathrm{weight}_\ell(x,y) \wedge \mathrm{value}_{\ell,k,v}(y)] & v \neq 0 \\ \exists y[\mathrm{weight}_\ell(x,y) \wedge \mathrm{value}_{\ell,k,v}(y)] \vee \neg\exists y[y < x] & v = 0. \end{cases} \tag{6.40}$$

By Lemma 6.8 again, there are formulas $\phi_{\ell,k,v}$ that define the output of the FFNN.

Finally, the activation vector at the last position is defined by $\exists n.(\forall x.x \leq n) \wedge \phi_{L,k,v}(n)$, and by Lemma 6.8, there is a single closed formula that defines the output of T. □

To go in the other direction, we use the fact that PTL is equivalent to FO (Kamp, 1968) and do an easier conversion from PTL.

Theorem 6.9. *For any formula ϕ of* PTL*, there is a transformer encoder with rightmost hard attention and strict future masking that recognizes the same language that ϕ defines.*

The proof will be by induction on the structure of ϕ, so we need the following lemma for combining the translations of sister subformulas.

Lemma 6.10 (Parallel composition)**.** *Given transformer encoders without layer normalization*

$$tfr_1 \colon \Sigma^* \xrightarrow{\text{lp}} (\mathbb{R}^{d_1})^*$$
$$tfr_2 \colon \Sigma^* \xrightarrow{\text{lp}} (\mathbb{R}^{d_2})^*$$

there is a transformer

$$tfr_1 \oplus tfr_2 \colon \Sigma^* \xrightarrow{\text{lp}} (\mathbb{R}^{d_1+d_2})^*$$

such that for all strings $w \in \Sigma^*$,

$$(tfr_1 \oplus tfr_2)(w[1] \cdots w[n]) = \begin{bmatrix} tfr_1(w)[1] \\ tfr_2(w)[1] \end{bmatrix} \cdots \begin{bmatrix} tfr_1(w)[n] \\ tfr_2(w)[n] \end{bmatrix}. \tag{6.41}$$

Proof. Let d_1 and d_2 be the width of tfr_1 and tfr_2, and let $d = d_1 + d_2$. If one of tfr_1 and tfr_2 has fewer layers than the other, add trivial layers until they have the same number of layers L.

The new transformer has embedding layer

$$(tfr_1 \oplus tfr_2).emb(w[1] \cdots w[n]) = \begin{bmatrix} tfr_1.emb(w)[1] \\ tfr_2.emb(w)[1] \end{bmatrix} \cdots \begin{bmatrix} tfr_1.emb(w)[n] \\ tfr_2.emb(w)[n] \end{bmatrix}.$$

For each layer $\ell \in [L]$, let $f_1 = tfr_1.layer_\ell$ and $f_2 = tfr_2.layer_\ell$. Widen f_1 into a layer f_1' with width d as follows.

$$f_1'.\mathbf{W}^{\mathrm{Q}} = \begin{bmatrix} f_1.\mathbf{W}^{\mathrm{Q}} & \mathbf{0} \end{bmatrix} \qquad f_1'.\mathbf{W}^{\mathrm{K}} = \begin{bmatrix} f_1.\mathbf{W}^{\mathrm{K}} & \mathbf{0} \end{bmatrix}$$

$$f_1'.\mathbf{W}^{\mathrm{V}} = \begin{bmatrix} f_1.\mathbf{W}^{\mathrm{V}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$$

$$f_1'.lin_1.\mathbf{W} = \begin{bmatrix} f_1.lin_1.\mathbf{W} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \qquad f_1'.lin_1.\mathbf{b} = \begin{bmatrix} f_1.lin_1.\mathbf{b} \\ \mathbf{0} \end{bmatrix}$$

$$f_1'.lin_2.\mathbf{W} = \begin{bmatrix} f_1.lin_2.\mathbf{W} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \qquad f_1'.lin_2.\mathbf{b} = \begin{bmatrix} f_1.lin_2.\mathbf{b} \\ \mathbf{0} \end{bmatrix}$$

Similarly, widen f_2 into a layer f_2' with width d, but using the bottom half of the activation vectors:

$$f_2'.\mathbf{W}^{\mathrm{Q}} = \begin{bmatrix} \mathbf{0} & f_2.\mathbf{W}^{\mathrm{Q}} \end{bmatrix} \qquad f_2'.\mathbf{W}^{\mathrm{K}} = \begin{bmatrix} \mathbf{0} & f_2.\mathbf{W}^{\mathrm{K}} \end{bmatrix}$$

$$f_2'.\mathbf{W}^{\mathrm{V}} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & f_2.\mathbf{W}^{\mathrm{V}} \end{bmatrix}$$

$$f_2'.lin_1.\mathbf{W} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & f_2.lin_1.\mathbf{W} \end{bmatrix} \qquad f_2'.lin_1.\mathbf{b} = \begin{bmatrix} \mathbf{0} \\ f_2.lin_1.\mathbf{b} \end{bmatrix}$$

$$f_2'.lin_2.\mathbf{W} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & f_2.lin_2.\mathbf{W} \end{bmatrix} \qquad f_2'.lin_2.\mathbf{b} = \begin{bmatrix} \mathbf{0} \\ f_2.lin_2.\mathbf{b} \end{bmatrix}$$

Then stack f_1' on top of f_2', or the other way around. (If we had multi-head attention, we could have combined f_1 and f_2 into a single layer.) □

Proof of Theorem 6.9. For any PTL formula ϕ, there is a transformer tfr_ϕ and an index k such that iff $tfr_\phi(w)[i,k] = \mathbb{I}[w, i \models \phi]$. We show this by induction on the structure of ϕ.

Base case $\phi = Q_a$ for some $a \in \Sigma$: Then tfr_ϕ is just an embedding function

$$tfr_\phi(w)[i] = \left[\mathbb{I}[w[i] = a]\right]. \tag{6.42}$$

If $\phi = \neg\phi_1$: By the induction hypothesis, there is a transformer tfr_1 simulating ϕ_1. For simplicity, we write activation vectors with just the components we're interested in:

$$tfr_{\phi_1}(w)[i] = \begin{bmatrix} \mathbb{I}[w, i \models \phi_1] \\ 0 \end{bmatrix}. \tag{6.43}$$

We can add a trivial self-attention layer and, by Theorem 3.5, a FFNN to simulate $\phi = \neg\phi_1$:

$$ffn_\neg(tfr_{\phi_1}(w))[i] = \begin{bmatrix} 0 \\ \mathbb{I}[w, i \models \neg\phi_1] \end{bmatrix}. \tag{6.44}$$

If $\phi = \phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$: By the induction hypothesis, there are transformers tfr_1 and tfr_2 simulating ϕ_1 and ϕ_2, respectively. Combine these using Lemma 6.10, add a trivial self-attention layer and, by Theorem 3.5, a FFNN to simulate ϕ.

If $\phi = \phi_1$ **since** ϕ_2: By the induction hypothesis, there are transformers tfr_1 and tfr_2 simulating ϕ_1 and ϕ_2, respectively. Combine these using Lemma 6.10. For simplicity, we write activation vectors as

$$\mathbf{H}_i(w) = \begin{bmatrix} \mathbb{I}[w, i \models \phi_1] \\ \mathbb{I}[w, i \models \phi_2] \\ 0 \end{bmatrix} \tag{6.45}$$

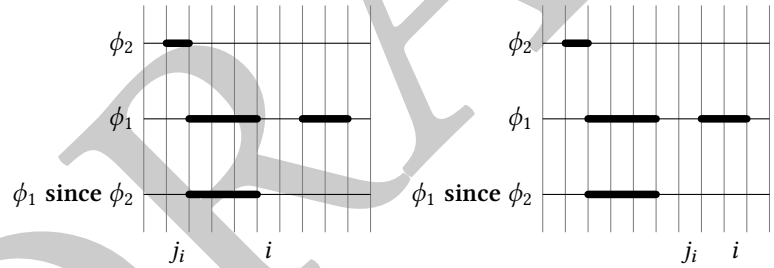
suppressing all other coordinates.

Then we add a self-attention layer:

$$\mathbf{q}^{(i)} = \begin{bmatrix} 1 \end{bmatrix} \qquad \mathbf{k}^{(j)} = \begin{bmatrix} \mathbb{I}[\neg(w, j \models \phi_1) \vee (w, j \models \phi_2)] \end{bmatrix} \tag{6.46}$$

$$\mathbf{v}^{(j)} = \begin{bmatrix} 0 \\ 0 \\ w, j \models \phi_2 \end{bmatrix}. \tag{6.47}$$

For any position i, the position j_i that receives attention is the rightmost one left of i that either satisfies ϕ_2 (in which case ϕ_1 must be satisfied from j_i to i exclusive) or does not satisfy ϕ_1 (in which case ϕ_2 must not be satisfied from j_i to i exclusive). These two possibilities are pictured below:



Then i satisfies (ϕ_1 **since** ϕ_2) if and only if j_i satisfies ϕ_2. So the value tests for whether ϕ_2 is satisfied.

At the leftmost position ($i = 1$), (ϕ_1 **since** ϕ_2) is false, so it's correct that the attention outputs the zero vector. □

Since this construction uses only position-independent queries (0 or 1), a perhaps surprising consequence is that every transformer encoder with rightmost hard attention and strict future masking is equivalent to one that uses only position-independent queries.

6.2.3 Encoders with soft attention

In this section, we consider transformers that don't have the added power of intermediate steps, and don't have the restriction of unique-hard attention. This seems to be the most difficult case to pin down. One key issue is that while unique-hard and average-hard attention only produce rational numbers, soft attention produces real numbers. So far, attempts to obtain upper bounds on the expressivity of soft-attention transformers involve limiting the precision of the numbers involved.

Actual computers, of course, use floating-point numbers with a constant (that is, independent of n) bits. But Merrill and Sabharwal (2023a) argue that in $O(1)$ precision, attention cannot attend uniformly to a string of length n, because for large enough n, the attention weights (α) would all round down to zero. Instead, they use $O(\log n)$ bits of precision.

Theorem 6.11 (Merrill and Sabharwal, 2023a)**.** *For any $O(\log n)$-precision transformer encoder T that recognizes a language L, there is a formula of* FOM[BIT] *that defines L.*

Merrill and Sabharwal (2023a) use floating-point numbers, of the form $m \cdot 2^e$, where the mantissa m has $O(\log n)$ bits including a sign bit, and the exponent e has $O(\log n)$ bits including a sign bit. In theoretical papers, I think it's more common to use a fixed-point representation, in which e is constant. With fixed-point numbers, it appears to be possible to improve the result to $O(n)$ bits of precision. We will consider both possibilities.

Proof. Merrill and Sabharwal (2023a)'s proof converted T to a family of threshold circuits, but we show how to go straight to FOM[BIT].

Transformers only use a handful of operations: addition, multiplication, division, max, exp, and iterated addition. It suffices to show that these operations can be defined in FO[BIT] on $O(\log n)$-bit fixed- or floating-point numbers.

Addition and *multiplication*, already defined on integers in Theorem 5.19, are generalized to $c \log n$ bit integers (where $c > 1$) by Schweikardt (2005, Theorem 3.4bd). Then fixed-point addition and multiplication can be defined using the following facts:

$$(m_1 \cdot 2^e) + (m_2 \cdot 2^e) = (m_1 + m_2) \cdot 2^e \tag{6.48}$$

$$(m_1 \cdot 2^e) \cdot (m_2 \cdot 2^e) = (m_1 m_2 \cdot 2^e) \cdot 2^e \tag{6.49}$$

Similarly for floating-point:

$$(m_1 \cdot 2^{e_1}) + (m_2 \cdot 2^{e_2}) = \begin{cases} (m_1 + m_2 \cdot 2^{e_2 - e_1}) \cdot 2^{e_1} & e_1 \geq e_2 \\ (m_1 \cdot 2^{e_1 - e_2} + m_2) \cdot 2^{e_2} & e_1 \leq e_2 \end{cases} \tag{6.50}$$

$$(m_1 \cdot 2^{e_1}) \cdot (m_2 \cdot 2^{e_2}) = m_1 m_2 \cdot 2^{e_1 + e_2}. \tag{6.51}$$

In all of the above, to get mantissas to be integers with the right number of bits, some rounding may be necessary. *Division* can be defined in terms of multiplication.

Iterated addition on fixed-point numbers reduces to iterated addition on integers:

$$\sum_{i=1}^{n}(m_i \cdot 2^e) = \left(\sum_{i=1}^{n} m_i\right) \cdot 2^e. \tag{6.52}$$

The sum of the m_i is only a mild extension of Theorem 5.26. After counting the bits in each column, we end up with $c \log n$ numbers with $\log n$ bits each. We sum each block of $\log n$ numbers and then add the resulting c block-sums.

On floating-point numbers, iterated addition is more difficult:

$$\sum_{i=1}^{n}(m_i \cdot 2^{e_i}) = \left(\sum_{i=1}^{n} \underbrace{(m_i \cdot 2^{e_i - e})}_{(*)}\right) \cdot 2^e \tag{6.53}$$

where $(*)$ is rounded off to the nearest integer. The problem is that if some of the m_i are negative, the sum could end up much smaller than the largest summand. For example, suppose mantissas have 50 bits, and we want to compute

$$1 \cdot 2^0 + -1 \cdot 2^0 + 1 \cdot 2^{-100} = 1 \cdot 2^{-100}.$$

If we choose e to be the maximum of the e_i, then $1 \cdot 2^{-100}$ would round off to 0, giving a sum of 0. (This is known as *catastrophic cancellation*.) Instead, to make the sum exact, Merrill and Sabharwal (2023b) choose e to be the *minimum* of the e_i, which makes each $(*)$ into a $O(\text{poly}(n))$-bit integer. Iterated addition of these so-called long integers is still possible in FOM[BIT] (Barrington and Maciel, 2000, Lecture 7). But if we had started with $O(n)$ bits, we would at this point have an exponential number of bits.

For the *exponential function* ($\exp x$), first observe that

$$\exp x = \exp_2(x/\log 2) \tag{6.54}$$

$$= \exp_2(\lfloor x/\log 2 \rfloor) \exp_2(x/\log 2 - \lfloor x/\log 2 \rfloor) \tag{6.55}$$

$$= \exp_2(\lfloor x/\log 2 \rfloor) \exp(\underbrace{x - \lfloor x/\log 2 \rfloor \log 2}_{r}). \tag{6.56}$$

The first factor can be computed by shifting. The second factor can be approximated by a Taylor series (Merrill, p.c.; Hesse et al., 2002, Corollary 6.5):

$$\exp r = \sum_{i=0}^{\infty} \frac{1}{i!} r^i = \sum_{i=0}^{k} \frac{1}{i!} r^i + R_k \tag{6.57}$$

where the Lagrange remainder term R_k is, for some z in $(0, r)$,

$$R_k = \frac{\exp z}{(k+1)!} r^{k+1} < \frac{\exp r}{(k+1)!} r^{k+1} < \frac{2}{(k+1)!} r^{k+1} \leq \frac{2}{2^{k+1}} r^{k+1} < \frac{1}{2^k}. \tag{6.58}$$

Letting $k = c \log n + 1$ ensures that the approximation is good to $c \log n$ bits.

So we compute Eq. (6.57) minus the remainder term R_k. Each term is an iterated product of $O(\log n)$ numbers, which can be expressed in FO[BIT] (Hesse et al., 2002, Theorem 5.1), and the summation of $c \log n + 2$ terms can also be expressed in FO[BIT]. □

Bibliography

Angluin, Dana, David Chiang, and Andy Yang (2023). Masked hard-attention transformers and Boolean RASP recognize exactly the star-free languages. arXiv:2310.13897.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). Layer normalization. In: *NIPS 2016 Deep Learning Symposium*.

Barceló, Pablo, Alexander Kozachinskiy, Anthony Widjaja Lin, and Vladimir Podolskii (2024). Logical languages accepted by transformer encoders with hard attention. In: *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*.

Barrington, David Mix and Alexis Maciel (2000). Advanced course on computational complexity. CMI-PCMI Undergraduate Program.

Bhattamishra, Satwik, Arkil Patel, and Navin Goyal (2020). On the computational power of Transformers and its implications in sequence modeling. In: *Proceedings of the 24th Conference on Computational Natural Language Learning (CoNLL)*, pp. 455–475.

Chiang, David and Peter Cholak (May 2022). Overcoming a theoretical limitation of self-attention. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 7654–7664.

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

Hao, Yiding, Dana Angluin, and Robert Frank (2022). Formal language recognition by hard attention Transformers: perspectives from circuit complexity. In: *Transactions of the Association for Computational Linguistics* 10, pp. 800–810.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). Deep residual learning for image recognition. arXiv:1512.03385.

Hesse, William, Eric Allender, and David A. Mix Barrington (2002). Uniform constant-depth threshold circuits for division and iterated multiplication. In: *Journal of Computer and System Sciences* 65.4, pp. 695–716.

Kamp, Johan Anthony Willem (1968). *Tense Logic and the Theory of Linear Order*. PhD thesis. University of California, Los Angeles.

Merrill, William, Vivek Ramanujan, Yoav Goldberg, Roy Schwartz, and Noah A. Smith (2021). Effects of parameter norm growth during transformer training: inductive bias from gradient descent. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1766–1781.

Merrill, William and Ashish Sabharwal (2023a). A logic for expressing log-precision transformers. In: *Advances in Neural Information Processing Systems*. Vol. 36, pp. 52453–52463.

— (2023b). The parallelism tradeoff: limitations of log-precision transformers. In: *Transactions of the Association for Computational Linguistics* 11, pp. 531–545.

— (2024). The expressive power of transformers with chain of thought. In: *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*.

Merrill, William, Ashish Sabharwal, and Noah A. Smith (2022). Saturated transformers are constant-depth threshold circuits. In: *Transactions of the Association for Computational Linguistics* 10, pp. 843–856.

Nye, Maxwell et al. (2022). Show your work: scratchpads for intermediate computation with language models. In: *Proceedings of the Workshop on Deep Learning for Code (DL4C)*.

Pérez, Jorge, Pablo Barceló, and Javier Marinkovic (2021). Attention is Turing-complete. In: *Journal of Machine Learning Research* 22, 75:1–75:35.

Schweikardt, Nicole (July 2005). Arithmetic, first-order logic, and counting quantifiers. In: *ACM Transactions on Computational Logic* 6.3, pp. 634–671.

Sipser, Michael (2013). *Introduction to the Theory of Computation*. 3rd. Cengage Learning.

Strobl, Lena, Dana Angluin, David Chiang, Jonathan Rawski, and Ashish Sabharwal (2024). Transformers as transducers. arXiv:2404.02040.

Strobl, Lena, William Merrill, Gail Weiss, David Chiang, and Dana Angluin (2024). What formal languages can transformers express? A survey. In: *Transactions of the Association for Computational Linguistics*. To appear.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin (2017). Attention is all you need. In: *Advances in Neural Information Processing Systems 30 (NeurIPS)*.

Wang, Qiang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao (2019). Learning deep Transformer models for machine translation. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Wei, Jason, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou (2022). Chain-of-thought prompt-

ing elicits reasoning in large language models. In: *Advances in Neural Information Processing Systems 35 (NeurIPS)*, pp. 24824–24837.

Xu, Kelvin, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio (2015). Show, attend and tell: neural image caption generation with visual attention. In: *Proceedings of the 32nd International Conference on Machine Learning*, pp. 2048–2057.

Yao, Shunyu, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan (Aug. 2021). Self-attention networks can process bounded hierarchical languages. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pp. 3770–3785.