Chapter 8

Transformers with intermediate steps

In this section, we are going to look at one of the two earliest results on transformer expressivity (Pérez et al., 2021):

> For any Turing machine M, there is a transformer decoder with average-hard attention and intermediate steps that simulates M.

We start by defining some key terms.

8.1 Background

8.1.1 Average-hard attention

As is common, this proof simplifies attention by making it focus attention only on the positions with the maximum score ($\mathbf{s}$). If there is more than one maximal position, attention is distributed evenly among them.

For any vector $\mathbf{x} \in \mathbb{R}^d$, define $M(\mathbf{x}) = \{i \in [n] \mid \forall j \in [n], \mathbf{x}[j] \leq \mathbf{x}[i]\}$ to be the set of indices of the maximal elements of $\mathbf{x}$. In *average*-hard attention, Eq. (5.8) is replaced with:

$$\alpha[i, j] = \frac{\mathbb{I}[j \in M(\mathbf{s}[i, :])]}{|M(\mathbf{s}[i, :])|}. \tag{8.1}$$

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).

8.1.2 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.

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, $\mathbf{y}_{<t} = y_0 \cdots y_{t-1}$, where $y_0 = \text{BOS}$. The output is a probability distribution $\hat{p}(y_t \mid \mathbf{y}_{<t})$ over the next symbol,

$$\begin{aligned} out\colon \mathbb{R}^d &\to \mathbb{R} \\ \mathbf{x} &\mapsto \mathbf{W}\mathbf{x} + \mathbf{b} \qquad (8.2) \\ \hat{p}(\cdot \mid \mathbf{y}_{<t}) &= \text{softmax}(out(tfr(\mathbf{y}_{<t})[t-1])) \qquad (8.3) \end{aligned}$$

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

To sample a string, we first sample y_1 from $\hat{p}(y_1 \mid \text{BOS})$, then, for each time step $t > 1$, sample y_t from $\hat{p}(y_t \mid \mathbf{y}_{<t})$. The process stops when $y_t = \text{EOS}$. Because each sampled output symbol becomes part of the input at the next time step, this kind of model is called *autoregressive*. See Fig. 8.1a.

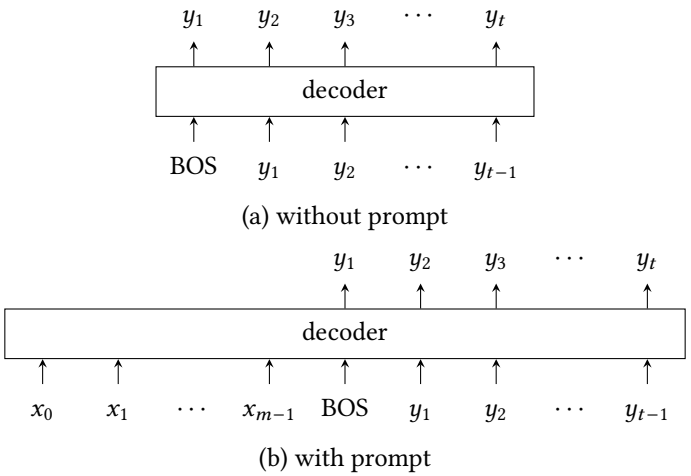
In most (not all) theoretical papers about transformer decoders, we want the decoder to output a single next symbol instead of a probability distribution over next symbols. To do this, we can select the argmax of $out(tfr(\mathbf{y}_{<t})[t-1])$ instead. Warning: In general, selecting the argmax at each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

We can also provide a *prompt* $\mathbf{x}$ 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 $\mathbf{x} \cdot \mathbf{y}_{<t}$, and the output is y_t. See Fig. 8.1b.

8.1.3 Intermediate steps

In many applications of transformer decoders, the prompt $\mathbf{x}$ is some kind of question, and the desired output $\mathbf{y}$ is the answer. For example, $\mathbf{x} = \texttt{101*101}$ and $\mathbf{y} = \texttt{10201}$. 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 8.1. Let f be a transformer decoder. For any string $\mathbf{x} \in \Sigma^*$, we say that f, on prompt $\mathbf{x}$, outputs y after T intermediate steps if there is a string $\mathbf{y} = y_0 \cdots y_{T+1}$ with $y_0 = \text{BOS}$, $y_{T+1} = y$, such that for all $t \in 1, \ldots, T+1$, we have $f(\mathbf{x} \cdot \mathbf{y}_{<t}) = \mathbf{e}_{y_t}$.



(a) without prompt

(b) with prompt

Figure 8.1: Generating strings from a transformer decoder.

8.2 Main result

Theorem 8.2. *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 $\mathbf{w} \in \Sigma^*$:*

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

The rest of this chapter 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.

The key idea in most of these proofs is that a transformer has no memory to store the contents of the tape. All it has is the intermediate steps, which it uses to record *changes* to the contents of the tape. Whenever it needs to read a cell of the tape, it must use the record of changes to reconstruct the contents of the cell.

Let $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$. The alphabet of $f.tfr$ is

$$\Sigma' = \Sigma \cup \{\text{BOS}, \text{ACC}, \text{REJ}\} \cup (Q \times \Gamma \times \{-1, +1\}).$$

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 $i = 0, 1, \ldots$ 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: $h^{(i)} = \sum_{j=0}^{i} m^{(j-1)}$.
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 $b^{(j)} \neq \bot$, 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 4.10 is simulated by:

i	tape	$x^{(i)}$	$q^{(i)}$	$b^{(i-1)}$	$m^{(i-1)}$	$h^{(i)}$	$a^{(i)}$	$y^{(i)}$
0		1	$\bot$	$\bot$	0	0	1	1
1		1	$\bot$	$\bot$	0	0	1	1
2		BOS	$\bot$	$\bot$	0	0	1	$(q_0, 1, 0)$
3	$\hat{1}1\text{␣}\cdots$	$(q_0, 1, 0)$	q_0	1	0	0	1	$(q_1, \text{␣}, +1)$
4	$\text{␣}\hat{1}\text{␣}\cdots$	$(q_1, \text{␣}, +1)$	q_1	␣	+1	1	1	$(q_2, \text{x}, +1)$
5	$\text{␣x}\hat{\text{␣}}\cdots$	$(q_2, \text{x}, +1)$	q_2	x	+1	2	␣	$(q_4, \text{␣}, -1)$
6	$\text{␣}\hat{\text{x}}\text{␣}\cdots$	$(q_4, \text{␣}, -1)$	q_4	␣	−1	1	x	$(q_4, \text{x}, -1)$
7	$\hat{\text{␣}}\text{x␣}\cdots$	$(q_4, \text{x}, -1)$	q_4	x	−1	0	␣	$(q_1, \text{␣}, +1)$
8	$\text{␣}\hat{\text{x}}\text{␣}\cdots$	$(q_1, \text{␣}, +1)$	q_1	␣	+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:

$$\mathbf{A}^{(0)}[i] = \begin{bmatrix} \mathbf{e}_{x^{(i)}} \\ 1/(i+1) \\ 1 \\ i \\ i^2 \end{bmatrix}. \tag{8.4}$$

Step 1 is piecewise linear, so it can be computed by a FFNN (Theorem 3.7). Afterwards, the activation vector at position i is:

$$\mathbf{A}^{(1)}[i] = \begin{bmatrix} \mathbf{e}_{x^{(i)}} \\ 1/(i+1) \\ 1 \\ i \\ i^2 \\ \mathbf{e}_{q^{(i)}} \\ \mathbf{e}_{b^{(i-1)}} \\ m^{(i-1)} \end{bmatrix}. \tag{8.5}$$

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

$$\mathbf{Q}[i] = 0 \qquad \mathbf{K}[j] = 0 \qquad \mathbf{V}[j] = \begin{bmatrix} \mathbf{0} \\ m^{(i-1)} \end{bmatrix}. \tag{8.6}$$

Uniform self-attention computes an average, not a sum. Since at time step i, there are $i+1$ positions to average over, the result is $h^{(i)}/(i+1)$, not $h^{(i)}$. We'll correct this in the next step.

$$\mathbf{A}^{(2)}[i] = \begin{bmatrix} \mathbf{e}_{x^{(i)}} \\ 1/(i+1) \\ 1 \\ i \\ i^2 \\ \mathbf{e}_{q^{(i)}} \\ \mathbf{e}_{b^{(i-1)}} \\ m^{(i-1)} \\ h^{(i)}/(i+1) \end{bmatrix} \tag{8.7}$$

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

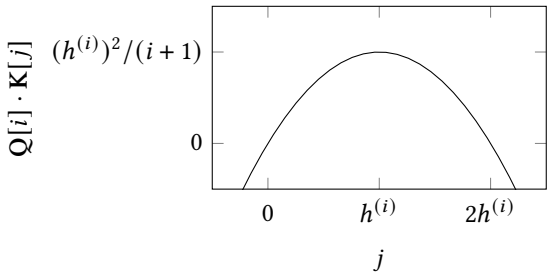
(Pérez et al., 2021), adding components to the position embedding (Pérez et al., 2021; Barceló et al., 2024; Strobl et al., 2024), or applying layer normalization only to selected components (Merrill and Sabharwal, 2024). The construction here is closest to that of Strobl et al. (2024).

It uses two self-attention layers. The first layer changes $h^{(i)}/(i+1)$ to $h^{(i)}$ by treating the position embeddings as a lookup table (Barceló et al., 2024).

$$\mathbf{Q}[i] = \begin{bmatrix} 2h^{(i)}/(i+1) \\ -1/(i+1) \end{bmatrix} \qquad \mathbf{K}[j] = \begin{bmatrix} j \\ j^2 \end{bmatrix} \qquad \mathbf{V}[j] = \begin{bmatrix} \mathbf{0} \\ j \\ j^2 \\ \mathbf{e}_{x^{(j)}} \end{bmatrix} \tag{8.8}$$

$$\mathbf{Q}[i] \cdot \mathbf{K}[j] = \frac{1}{i+1} j(2h^{(i)} - j). \tag{8.9}$$

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:

$$\mathbf{A}^{(3.5)}[i] = \begin{bmatrix} \mathbf{e}_{x^{(i)}} \\ 1/(i+1) \\ 1 \\ i \\ i^2 \\ \mathbf{e}_{q^{(i)}} \\ \mathbf{e}_{b^{(i-1)}} \\ m^{(i-1)} \\ h^{(i)}/(i+1) \\ h^{(i)} \\ (h^{(i)})^2 \\ \mathbf{e}_{x^{(h^{(i)})}} \end{bmatrix}. \tag{8.10}$$

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

$j \le i$ such that $h^{(j-1)} = h^{(i)}$. So we will need

$$h^{(i-1)} = h^{(i)} - m^{(i-1)} \tag{8.11}$$

$$(h^{(i-1)})^2 = (h^{(i)})^2 - 2h^{(i)}m^{(i-1)} + (m^{(i-1)})^2 \tag{8.12}$$

$$= \begin{cases} (h^{(i)})^2 + 2h^{(i)} + 1 & \text{if } m^{(i-1)} = -1 \\ (h^{(i)})^2 - 2h^{(i)} + 1 & \text{if } m^{(i-1)} = +1 \end{cases} \tag{8.13}$$

which can both be computed using a FFN. So

$$\mathbf{A}^{(4)}[i] = \begin{bmatrix} \mathbf{e}_{x^{(i)}} \\ 1/(i+1) \\ 1 \\ i \\ i^2 \\ \mathbf{e}_{q^{(i)}} \\ \mathbf{e}_{b^{(i-1)}} \\ m^{(i-1)} \\ h^{(i)}/(i+1) \\ h^{(i)} \\ (h^{(i)})^2 \\ \mathbf{e}_{x^{(h^{(i)})}} \\ h^{(i-1)} \\ (h^{(i-1)})^2 \end{bmatrix}. \tag{8.14}$$

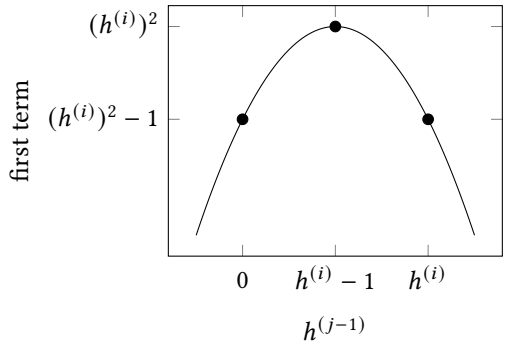
The second self-attention layer uses another variation of the lookup trick:

$$\mathbf{Q}[i] = \begin{bmatrix} 2h^{(i)} \\ -1 \\ \frac{1}{2(i+1)} \end{bmatrix} \qquad \mathbf{K}[j] = \begin{bmatrix} h^{(j-1)} \\ (h^{(j-1)})^2 \\ j \end{bmatrix} \qquad \mathbf{V}[j] = \begin{bmatrix} \mathbf{0} \\ h^{(j-1)} \\ \mathbf{e}_{b^{(j-1)}} \end{bmatrix} \tag{8.15}$$

$$\mathbf{Q}[i] \cdot \mathbf{K}[j] = \underbrace{h^{(j-1)}(2h^{(i)} - h^{(j-1)})}_{\text{find } h^{(j-1)} = h^{(i)}} + \underbrace{\frac{j}{2(i+1)}}_{\text{find rightmost}} \tag{8.16}$$

The first term is maximized when $h^{(j-1)} = h^{(i)}$, in which case it attains its maximum of $(h^{(i)})^2$.

If there is more than one such position j, then because of the second term, j^* is the rightmost such j. This second term has to be increasing in j, but it also has to be small enough that it cannot make the second-highest score as big as the highest score. Since $h^{(j-1)}$ is an integer, the difference between the first- and second-best values of the first term is 1:



So we make the second term $\frac{j}{2(i+1)} \leq \frac{1}{2} < 1$.

This gives

$$\mathbf{H}^{(4)}[i] = \begin{bmatrix} \mathbf{e}_{x^{(i)}} \\ 1/(i+1) \\ 1 \\ i \\ i^2 \\ \mathbf{e}_{q^{(i)}} \\ \mathbf{e}_{b^{(i-1)}} \\ m^{(i-1)} \\ h^{(i)}/(i+1) \\ h^{(i)} \\ (h^{(i)})^2 \\ \mathbf{e}_{x^{(h^{(i)})}} \\ h^{(i-1)} \\ (h^{(i-1)})^2 \\ h^{(j^*-1)} \\ \mathbf{e}_{b^{(j^*-1)}} \end{bmatrix}. \tag{8.17}$$

Finally, we can use the FFNN to set

$$a^{(i)} = \begin{cases} b^{(j^*-1)} & h^{(j^*-1)} = h^{(i)} \text{ and } b^{(j^*-1)} \neq \perp \\ x^{(h^{(i)})} & \text{otherwise.} \end{cases} \tag{8.18}$$

So the activation vectors are:

$$\mathbf{A}^{(5)}[i] = \begin{bmatrix} \mathbf{e}_{x^{(i)}} \\ 1/(i+1) \\ 1 \\ i \\ i^2 \\ \mathbf{e}_{q^{(i)}} \\ \mathbf{e}_{b^{(i-1)}} \\ m^{(i-1)} \\ h^{(i)}/(i+1) \\ h^{(i)} \\ (h^{(i)})^2 \\ \mathbf{e}_{x^{(h^{(i)})}} \\ h^{(i-1)} \\ (h^{(i-1)})^2 \\ h^{(j^*-1)} \\ \mathbf{e}_{b^{(j^*-1)}} \\ \mathbf{e}_{a^{(i)}} \end{bmatrix}. \tag{8.19}$$

Step 4 is piecewise linear, so it can be computed by a FFNN (Theorem 3.7).

Exercise 8.3. In Sipser's definition of a TM, if the head is at position 0 and moves to the left, then it remains at position 0. What happens in the above simulation of a TM? How would you fix it?