Expressivity of Transformers: Logic, Circuits, and Formal Languages

Day 4: Encoders with Soft Attention

David Chiang (Univ. of Notre Dame, USA)
Jon Rawski (MIT/San Jose State Univ., USA)
Lena Strobl (Umeå University, Sweden)
Andy Yang (Univ. of Notre Dame, USA)

1 August 2024

Today's Goals

- **Describe** the concept of soft attention and its role in transformer encoders.
- **Explain** the upper and lower bounds of computational expressivity in soft attention models.
- **Identify** key arithmetic predicates, counting quantifiers, and their relevance in soft attention encoders.

Soft attention is hard

In the previous two days, we gave exact equivalences:

$$\text{masked UHATs} = \text{star-free}$$

$$\text{AHAT decoders with intermediate steps} = \text{Turing-complete}$$

Today, we turn to softmax-attention transformers (SMATs), which we don't have an exact characterization of.

Instead, we have upper bounds and lower bounds.

Upper bound

Extending FO

We saw already that FO is equivalent to UHATs, but FO is not powerful enough for SMATs. It's not hard to write a SMAT for:

$$\text{MAJORITY} = \{\mathbf{w} \in \{\texttt{0}, \texttt{1}\}^* \mid \mathbf{w} \text{ has more } \texttt{1}\text{'s than } \texttt{0}\text{'s}\}.$$

We need to extend FO with:

- Arithmetic predicates
- Majority or counting quantifiers

Arithmetic predicates

- We can increase the expressivity of FO by adding more predicates besides $<$. The logic FO$[+, \times]$ extends FO with predicates:

$$\begin{aligned} w, I \models x + y = z &\quad \text{if } I(x) + I(y) = I(z) \\ w, I \models x \times y = z &\quad \text{if } I(x)I(y) = I(z) \end{aligned}$$

- FO$[+, \times]$ is also called FO$[\mathrm{BIT}]$

Exercise

Write a formula $\mathrm{ODD}(x)$ that tests whether x is odd.

Arithmetic predicates

Things that can be defined in FO[$+, \times$]:

- The sum of $O(\log n)$ numbers with $O(\log n)$ bits each [Immerman, 1999]
- The product of $O(\log n)$ numbers with $O(\log n)$ bits each [Hesse et al., 2002]
- x^y (special case of above)

Theorem (Barrington et al., 1990)

FO[$+, \times$] *defines exactly the languages in* DLOGTIME-*uniform* AC^0*.*

Circuit Complexity Classes

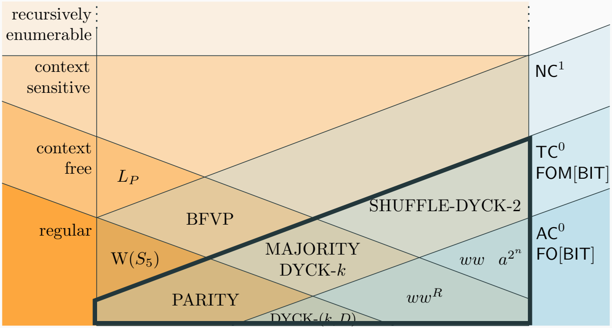
Definition (TC^k)

TC^k is the class of languages that can be recognized by families of circuits with

1. unbounded fan-in,
2. $O(\text{poly}(n))$ size,
3. $O((\log n)^k)$ depth, and
4. MAJORITY gates, which output 1 iff at least half of their inputs are 1.

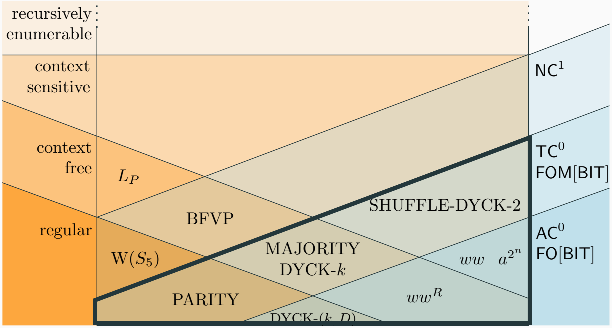
Definition (TC^0)

TC^0 is the class of languages that can be recognized by families of circuits with unbounded fan-in, $O(\text{poly}(n))$ size, $O(1)$ depth, and MAJORITY gates.



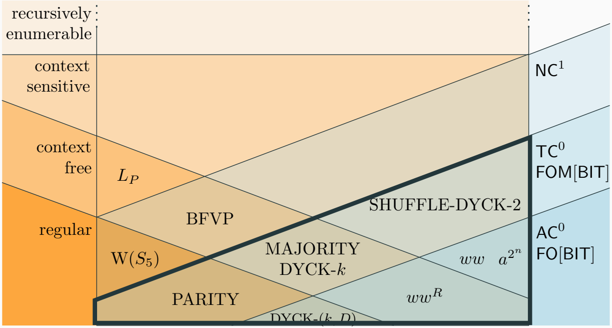
Definition (NC^1)

NC^1 is the class of languages that can be recognized by families of circuits with fan-in at most 2, $O(\text{poly}(n))$ size, and $O((\log n))$ depth.



Circuit Complexity Classes

We will show that transformers (with $O(\log n)$ precision) are in TC^0. It's widely believed that $\mathsf{TC}^0 \neq \mathsf{NC}^1$ (as in the figure). If so, then NC^1-complete languages do not belong to TC^0. Consequently, we don't think that transformers can recognize them either.

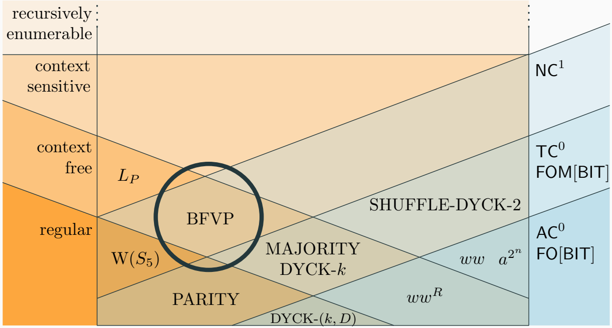


Let's look at two examples.

Circuit Complexity Classes

Example (Boolean Formula Value Problem (BFVP))

The BFVP is to decide whether a Boolean formula (with constants 0 and 1, no variables) is true or not. This problem is context-free and NC^1-complete.



Uniform TC^0 and NC^1 Languages

Example (Boolean Formula Value Problem (BFVP))

The BFVP is to decide whether a Boolean formula (with constants 0 and 1, no variables) is true or not.

Examples:

$$1 \in \mathsf{BFVP} \qquad 0 \notin \mathsf{BFVP}$$
$$1 \wedge 1 \in \mathsf{BFVP} \qquad 1 \wedge 0 \notin \mathsf{BFVP}$$
$$0 \vee (1 \wedge 1) \in \mathsf{BFVP} \qquad 1 \wedge (1 \wedge 0) \notin \mathsf{BFVP}$$

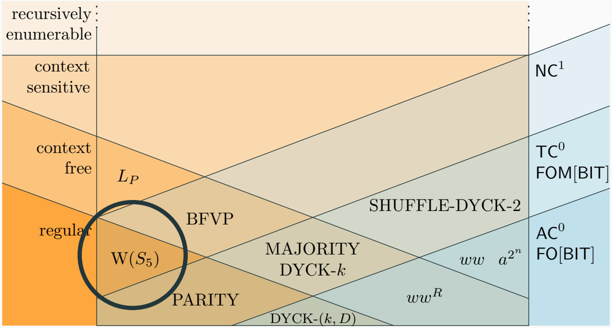
This problem is context-free and NC^1-complete.

Boolean Formulas and Compositional Semantics

- Evaluating Boolean formulas is crucial for computations in compositional semantics.
- Boole's work on logical descriptions aimed to codify a "language of thought."
- Modern semantic theory, influenced by lambda calculus (Montague, Partee), builds on this foundation.
- The relationship between neural networks and compositional behavior has been debated for decades (Fodor, Pylyshyn, Smolensky).

Example (Word Problem for S_5)

S_5 is the set of all permutations of $\{1, 2, 3, 4, 5\}$. The word problem for S_5 is regular and NC^1-complete.



Example (Word Problem for S_5)

S_5 is the set of all permutations of $\{1, 2, 3, 4, 5\}$. For simplicity, let's just consider

$$\begin{aligned} \mathsf{s} &= (12) && \text{swap 1 and 2} \\ \mathsf{c} &= (12345) && \text{cycle } 1 \mapsto 2, 2 \mapsto 3, \ldots, 5 \mapsto 1 \end{aligned}$$

Does a sequence of permutations equal the identity permutation?

$$\begin{aligned} \varepsilon &\in \mathsf{W}(S_5) & & \\ \mathsf{ss} &\in \mathsf{W}(S_5) & \mathsf{s} &\notin \mathsf{W}(S_5) \\ \mathsf{ccccc} &\in \mathsf{W}(S_5) & \mathsf{cccc} &\notin \mathsf{W}(S_5) \\ \mathsf{scccccs} &\in \mathsf{W}(S_5) & & \end{aligned}$$

This problem is regular and NC^1-complete.

GPT 3.5 and $W(S_5)$

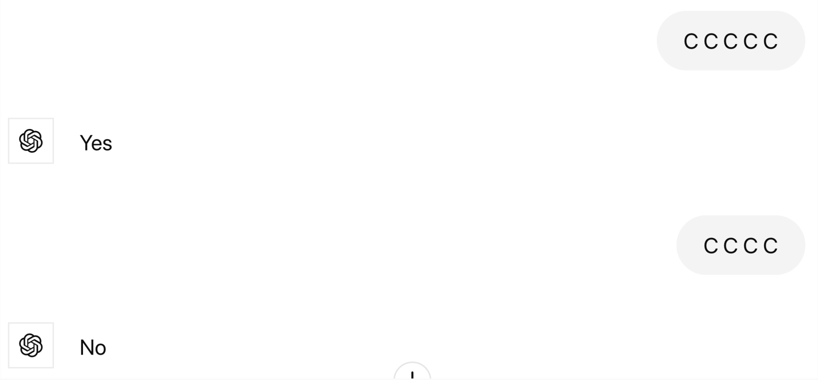
I have five cups, labeled A through E. When I say "S", swap the first two cups. When I say "C", move the first cup to the last position. When I saw "P", don't move any cups. Then please tell me if the cups are in order. Do not write anything except for a yes or no answer. Ready?

Ready.

S S

Yes

GPT 3.5 and $\mathrm{W}(S_5)$



GPT 3.5 and $W(S_5)$

S C C C C C C S

No

S C S

Yes

Natural Language Relevance of $\mathsf{W}(S_k)$

The languages $\mathsf{W}(S_k)$ have some relevance to natural language:

- They resemble expressions like *the child of the enemy of Ann* where the interpretation of *the child of* is (roughly) a permutation of possible referents [Paperno, 2022].
- They have been used to benchmark transformers' state-tracking abilities [Kim and Schuster, 2023].

Counting Quantifiers

FOC is first order logic with *counting terms* [van Benthem and Icard, 2023].

Example (Majority Language)

The majority language,

$$\mathrm{MAJORITY} = \{\mathbf{w} \in \{0,1\}^* \mid \mathbf{w} \text{ has more 1's than 0's}\}$$

can be defined by the FOC formula

$$\underbrace{(\#z.Q_0(z))}_{\text{number of 0's}} < \underbrace{(\#z.Q_1(z))}_{\text{number of 1's}}.$$

Counting Quantifiers

Exercise (Parity)

Write a FOC[+] formula for the language

$$\text{PARITY} = \{\mathbf{w} \in \{0, 1\}^* \mid \mathbf{w} \text{ has an odd number of 1's}\}.$$

FOC and TC^0

Equivalences:

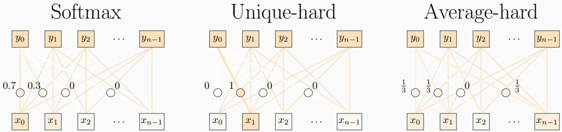
- $FOC[+, \times] = FOC[\times]$ [Lange, 2004]
- FOC = FOM (FO with *majority quantifiers*) and is more commonly called that
- $FOC[+, \times] =$ DLOGTIME-uniform TC^0

Things they can do:

- The sum of $O(\text{poly}(n))$ numbers with $O(\text{poly}(n))$ bits each
- The product of $O(\text{poly}(n))$ numbers with $O(\text{poly}(n))$ bits each [Hesse et al., 2002]
- Division and remainder of two $O(\text{poly}(n))$-bit numbers [Hesse et al., 2002]

Precision in transformers

- UHATs and AHATs only produce rational numbers, while soft attention produces real numbers.
- Upper bounds on the expressivity of SMATs involve limiting the precision of the numbers involved.



Produces	$\mathbb{R}$	$\mathbb{Q}$	$\mathbb{Q}$

Precision

- infinite
- $O(\log n)$
- $O(1)$

Precision in transformers

- UHATs and AHATs only produce rational numbers, while soft attention produces real numbers.
- Upper bounds on the expressivity of SMATs involve limiting the precision of the numbers involved.
- Merrill and Sabharwal [2023] argue that $O(1)$ precision is too small. You need $\log n$ bits just to store the number n or $1/n$!
- Instead, they use $O(\log n)$ bits of precision.

Floating-point numbers

$$\overbrace{\pm 1.bbb \cdots bbb}^{\text{mantissa/significand}} \times 2^{\overbrace{\pm bbb \cdots bbb}^{\text{exponent}}}$$

The significand and exponent combined have $p \in O(\log n)$ bits

Details aren't very important:

- How to apportion bits between the significand and exponent
- Does the sign bit count as a bit
- And so on.

Main result

Theorem (Merrill and Sabharwal, 2023)

For any $O(\log n)$*-bit floating-point transformer encoder* T *that recognizes a language* L*, there is a formula of* $\mathsf{FOC}[+, \times]$ *that defines* L*.*

Merrill and Sabharwal [2023]'s proof converted T to a family of threshold circuits, but we show how to go straight to $\mathsf{FOC}[+, \times]$.

What's in a Transformer?

- Addition ($+$), multiplication, comparison ($<$) of two numbers
- Exponential function (exp)
- Addition of n numbers ($\sum$)
- If layer normalization: division ($\div$), square root ($\sqrt{}$)

We just need to show that these can be done on $O(\log n)$-bit floating point numbers. Most of these operations can be reduced to operations on $O(\log n)$-bit integers.

Exercise

Explain why multiplication of two $O(\log n)$-bit floating point numbers is definable in $\mathrm{FOC}[+, \times]$. How about addition?

Summation (iterated addition)

Summation of n $O(\log n)$-bit floating point numbers: just convert every summand into a fixed-point number

$$
\begin{array}{r} 1.011 \cdot 2^{0} \\ 1.100 \cdot 2^{-5} \\ 1.111 \cdot 2^{-10} \\ +\, 1.101 \cdot 2^{-15} \\ \hline ?.??? \cdot 2^{?} \end{array}
\quad \rightsquigarrow \quad
\begin{array}{rl} & 1.011 \\ & 0.00001100 \\ & 0.0000000001111 \\ + & 0.000000000000001101 \\ \hline & 1.011011000111101101 \end{array}
$$

Each number has $2^{O(\log n)} = O(\mathrm{poly}(n))$ bits, and the sum of n numbers with $O(\mathrm{poly}(n))$ bits is still in $\mathrm{FOC}[+, \times]$.

Exponential function

Compute first p terms of Taylor series:

$$\begin{aligned}\exp x &= \sum_{i=0}^{p-1} \frac{x^i}{i!} \\ &= 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^{p-1}}{(p-1)!}\end{aligned}$$

If $p \in O(\log n)$:

- The error will be less than 2^{-p+1}.
- Each term in the series is an iterated product of $O(\log n)$ numbers, expressible in $\mathrm{FO}[+, \times]$.
- Summation of the terms can also be expressed in $\mathrm{FO}[+, \times]$.

Recap

- $O(\log n)$-precision transformer encoders only recognize languages in $\mathsf{FOC}[+, \times] = \mathsf{DLOGTIME}$-uniform TC^0.
- Assuming that $\mathsf{TC}^0 \neq \mathsf{NC}^1$, this implies that $O(\log n)$-precision transformer encoders cannot solve many interesting problems:
  - deciding whether a Boolean formula is true
  - deciding whether a sequence of permutations is the identity
  - reconstructing a chess board from chess moves [Merrill et al., 2024]

Open Questions

- Are $O(1)$-precision transformers equivalent to $\mathsf{FO} = \mathsf{LTL}$?
- At $O(\log n)$ precision, every operation except summation is in $\mathsf{FO}[+, \times]$. Is there a tighter bound than $\mathsf{FOC}[+, \times]$?
- At infinite precision, is it possible to find an upper bound?

Lower Bound

Softmax Attention and Related Work

- Bhattamishra et al. [2020] showed that one-state Parikh automata can be simulated by SMATs.
- Chiang et al. [2023] defined a logic called FOC[$+$; MOD] and showed that it can be simulated by SMATs.
- Barceló et al. [2024] defined an extension of LTL with counting, called LTL[$\#, +$], and showed that it can be simulated by AHATs.
- Perhaps surprisingly, there isn't a published proof that softmax-attention transformers can simulate LTL.

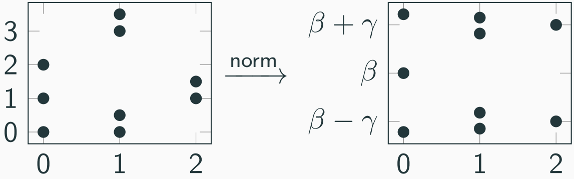
Simulating Temporal Logic with Softmax Attention

Now, we show that softmax-attention transformers can simulate a temporal logic without **since** but with a counting operator [Yang and Chiang, 2024]. We call this logic $\mathsf{K}_t[\#, +]$.

Layer Normalization

Our proof relies crucially on *layer normalization*

- Position-wise function $\mathbb{R}^d \to \mathbb{R}^d$
- Scales and shifts the components of a vector to have mean β and standard deviation γ



- In practice β and γ are learned; here, we choose them

Dyck-1

Set of all strings over $\{(,)\}$ that are balanced and nested. That is

- Total number of (equals total number of)
- At no point in the string is the number of) greater than the number of (

We can test for these constraints using $\mathsf{K}_t[\#, +]$.

$\mathsf{K}_t[\#, +]$ **Example**

Dyck-1 is the language of nested and balanced parentheses.

$$\phi = (\#[Q_{(}] = \#[Q_{)}]) \wedge (\#[\#[Q_{)}] > \#[Q_{(}]] = 0)$$

	(	(	(	)	)	(	)	)
$Q_{(}$	T	T	T	F	F	T	F	F

$\mathsf{K}_t[\#, +]$ **Example**

Dyck-1 is the language of nested and balanced parentheses.

$$\phi = (\#[Q_(] = \#[Q_)]) \wedge (\#[\#[Q_)] > \#[Q_(]] = 0)$$

	(	(	(	)	)	(	)	)
$Q_($	T	T	T	F	F	T	F	F
$Q_)$	F	F	F	T	T	F	T	F

$\mathsf{K}_t[\#, +]$ **Example**

Dyck-1 is the language of nested and balanced parentheses.

$$\phi = (\#[Q_(] = \#[Q_)]) \land (\#[\#[Q_)] > \#[Q_(]] = 0)$$

	(	(	(	)	)	(	)	)
$Q_($	T	T	T	F	F	T	F	F
$Q_)$	F	F	F	T	T	F	T	F
$\#[Q_(]$	1	2	3	3	3	4	4	4

$\mathsf{K}_t[\#, +]$ **Example**

Dyck-1 is the language of nested and balanced parentheses.

$$\phi = (\#[Q_{(}] = \#[Q_{)}]) \wedge (\#[\#[Q_{)}] > \#[Q_{(}]] = 0)$$

	(	(	(	)	)	(	)	)
$Q_{(}$	T	T	T	F	F	T	F	F
$Q_{)}$	F	F	F	T	T	F	T	F
$\#[Q_{(}]$	1	2	3	3	3	4	4	4
$\#[Q_{)}]$	0	0	0	1	2	2	3	4

$\mathsf{K}_t[\#, +]$ **Example**

Dyck-1 is the language of nested and balanced parentheses.

$$\phi = (\#[Q_(] = \#[Q_)]) \wedge (\#[\#[Q_)] > \#[Q_(]] = 0)$$

	(	(	(	)	)	(	)	)
$Q_($	T	T	T	F	F	T	F	F
$Q_)$	F	F	F	T	T	F	T	F
$\#[Q_(]$	1	2	3	3	3	4	4	4
$\#[Q_)]$	0	0	0	1	2	2	3	4
$\#[\#[Q_)] > \#[Q_(]]$	0	0	0	0	0	0	0	0

$\mathsf{K}_t[\#, +]$ **Example**

Dyck-1 is the language of nested and balanced parentheses.

$$\phi = (\#[Q_(] = \#[Q_)]) \wedge (\#[\#[Q_)] > \#[Q_(]] = 0)$$

	(	(	(	)	)	(	)	)
$Q_($	T	T	T	F	F	T	F	F
$Q_)$	F	F	F	T	T	F	T	F
$\#[Q_(]$	1	2	3	3	3	4	4	4
$\#[Q_)]$	0	0	0	1	2	2	3	4
$\#[\#[Q_)] > \#[Q_(]]$	0	0	0	0	0	0	0	0
$\#[Q_(] = \#[Q_)]$	F	F	F	F	F	F	F	T

$\mathsf{K}_t[\#, +]$ **Example**

Dyck-1 is the language of nested and balanced parentheses.

$$\phi = (\#[Q_(] = \#[Q_)]) \wedge (\#[\#[Q_)] > \#[Q_(]] = 0)$$

	(	(	(	)	)	(	)	)
$Q_($	T	T	T	F	F	T	F	F
$Q_)$	F	F	F	T	T	F	T	F
$\#[Q_(]$	1	2	3	3	3	4	4	4
$\#[Q_)]$	0	0	0	1	2	2	3	4
$\#[\#[Q_)] > \#[Q_(]]$	0	0	0	0	0	0	0	0
$\#[Q_(] = \#[Q_)]$	F	F	F	F	F	F	F	T
$\#[\#[Q_)] > \#[Q_(]] = 0$	T	T	T	T	T	T	T	T

$\mathsf{K}_t[\#, +]$ **Example**

Dyck-1 is the language of nested and balanced parentheses.

$$\phi = (\#[Q_(] = \#[Q_)]) \wedge (\#[\#[Q_)] > \#[Q_(]] = 0)$$

	(	(	(	)	)	(	)	)
$Q_($	T	T	T	F	F	T	F	F
$Q_)$	F	F	F	T	T	F	T	F
$\#[Q_(]$	1	2	3	3	3	4	4	4
$\#[Q_)]$	0	0	0	1	2	2	3	4
$\#[\#[Q_)] > \#[Q_(]]$	0	0	0	0	0	0	0	0
$\#[Q_(] = \#[Q_)]$	F	F	F	F	F	F	F	T
$\#[\#[Q_)] > \#[Q_(]] = 0$	T	T	T	T	T	T	T	T
ϕ	F	F	F	F	F	F	F	T

$\mathsf{K}_t[\#, +]$

The syntax of $\mathsf{K}_t[\#, +]$ is defined as follows:

$$
\begin{aligned}
t &::= \#[\phi_1] \\
&\quad \mid t_1 + t_2 \\
\phi &::= Q_\sigma && \sigma \in \Sigma \\
&\quad \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \\
&\quad \mid t_1 = t_2 \mid t_1 < t_2
\end{aligned}
$$

Other operators ($\vee$, $\rightarrow$, $>$, $\leq$, $\geq$) can be defined in terms of the ones above.

More Examples

Language	Formula
a^*b^*	$\#[Q_a \wedge (\#[Q_b] \geq 1)] = 0$
$a^*b^*a^*$	$\#[Q_b \wedge \#[Q_a \wedge (\#[Q_b] \geq 1)] \geq 1] = 0$
Dyck-1	$(\#[Q_(] = \#[Q_)]) \wedge (\#[\#[Q_)] > \#[Q_(]] = 0)$
$a^n b^n c^n$	$\#[Q_b \wedge (\#[Q_c] = 0)] = \#[Q_b]$ $\wedge \#[Q_a \wedge (\#[Q_b \vee Q_c] = 0)] = \#[Q_a]$ $\wedge \#[Q_a] = \#[Q_b] \wedge \#[Q_b] = \#[Q_c] \wedge \#[Q_c] = \#[Q_a]$
`hello`	$\#[\top] = 5 \wedge Q_o \wedge \#[Q_l \wedge \#[Q_e \wedge \#[Q_h] = 1] = 1] = 2$

Main Result

Theorem (Yang and Chiang, 2024)

For any formula ϕ *of* $\mathsf{K}_t[\#, +]$ *that defines a language* L*, there is a transformer encoder that recognizes* L*.*

Key Idea 1

Use uniform attention to count.

Problem:

- Uniform attention doesn't count; it averages

$$\frac{|count_i|}{i+1}$$

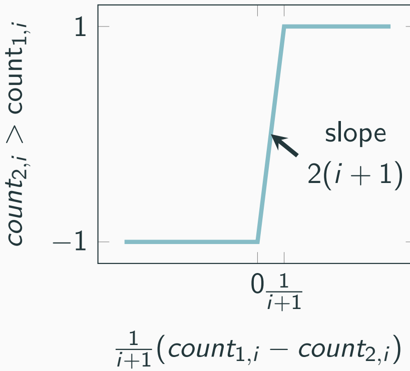
how to get rid of this?

- Position embedding tricks (Day 3) require average-hard attention
- Instead: Just keep the $\frac{1}{i+1}$ for now

Key Idea 2

Implement $count_{2,i} > \mathrm{count}_{1,i}$ as $\frac{1}{i+1}(count_{2,i} - \mathrm{count}_{1,i}) > 0$.

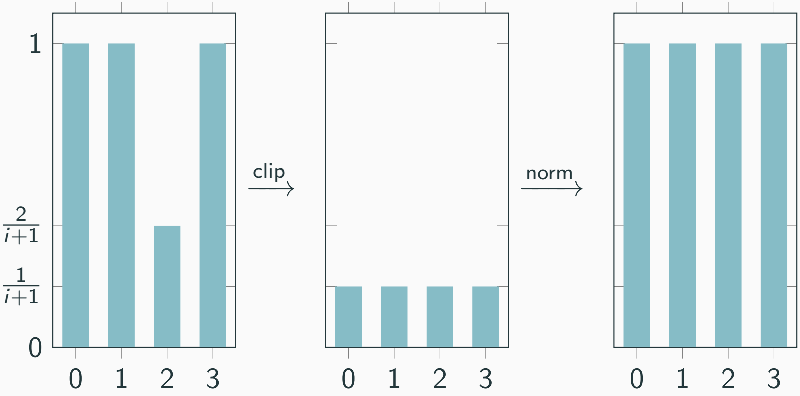
Problem: We need a function like



Slope not bounded (i.e., not Lipschitz continuous) $\Rightarrow$ can't be computed by FFNN

Key Idea 3

Use layer normalization to make everything ± 1.



Boolean and Count Representations

- To represent Boolean values, we use the following representations:

$$\text{true} : \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$\text{false} : \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

- To represent the integer count_i in position i, we use:

$$\begin{bmatrix} \frac{\text{count}_i}{i+1} \\ -\frac{\text{count}_i}{i+1} \end{bmatrix}$$

- These representations have zero mean so that layer normalization does not shift them up or down

Counting

In the following, we show how to simulate a $\#$ term in $\mathsf{K}_t[\#, +]$ using a uniform attention layer.

Lemma

Let $\mathbf{A}[*, 2k : 2k+1]$ *store a sequence of Boolean values* $\phi(i)$ *as defined above. For any* i*, let* $C(i)$ *be the number of positions* $j \leq i$ *such that* $\mathbf{A}[j, 2k : 2k+1]$ *is true. Then there is a transformer block that computes, at each position* i*, and in two other dimensions* $2k', 2k'+1$*, the values* $-\frac{C(i)}{i+1}$ *and* $\frac{C(i)}{i+1}$*.*

Uniform Attention for Averaging

We compute in position i of dimension k, the value $C(i)_k$, which is the average of all values up to position i in dimension k, The expression reduces to:

$$
\begin{aligned}
c_{i,k} &= \frac{\sum_{j=0}^{i} \exp(s_{ij})[W^{(V)}A_{*,j}]_k}{\sum_{j=0}^{i} \exp(s_{ij})} \\
&= \frac{\sum_{j=0}^{i}[W^{(V)}A_{*,j}]_k}{\sum_{j=0}^{i} 1} \\
&= \frac{\sum_{j=0}^{i} A_{k,j}}{i+1}
\end{aligned}
$$

Counting Vector

Since Booleans are stored as ± 1, the count we compute actually ends up being $2C(i)+1$, which we can easiliy correct with a FFNN.

$$\begin{bmatrix} \vdots \\ -2\phi(i)-1 \\ 2\phi(i)+1 \\ \vdots \\ -\frac{2}{i+1}\sum_i \phi(i)-1 \\ \frac{2}{i+1}\sum_i \phi(i)+1 \\ \vdots \end{bmatrix}$$

Counting Vector

Since Booleans are stored as ± 1, the count we compute actually ends up being $2C(i)+1$, which we can easiliy correct with a FFNN.

$$\begin{bmatrix} \vdots \\ -2\phi(i)-1 \\ 2\phi(i)+1 \\ \vdots \\ -\frac{2}{i+1}\sum_i \phi(i)-1 \\ \frac{2}{i+1}\sum_i \phi(i)+1 \\ \vdots \end{bmatrix} \xrightarrow{\text{FFNN}} \begin{bmatrix} \vdots \\ -2\phi(i)-1 \\ 2\phi(i)+1 \\ \vdots \\ -\frac{1}{i+1}\sum_i \phi(i) \\ \frac{1}{i+1}\sum_i \phi(i) \\ \vdots \end{bmatrix}$$

Linear Constraints

$\mathsf{K}_t[\#, +]$ can express any linear constraint on counts, that is, constraints of the form

$$\sum_{k \in K} a_k C_k(i) \geq 0$$

where the C_k are count terms, the a_k are integer coefficients, and K is a finite set of indices.

Testing Linear Constraints

Recall that a count C_k is stored as a pair of numbers $\frac{C_k}{i+1}$. Thus we need to test if

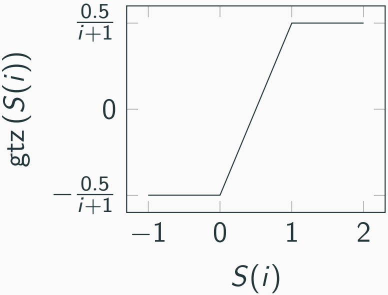
$$\sum_{k \in K} a_k \frac{C_k(i)}{i+1} \geq 0 \iff \frac{1}{i+1} + \sum_{k \in K} a_k \frac{C_k(i)}{i+1} \geq \frac{1}{i+1}$$

Which boils down to testing if a value is $\geq \frac{1}{i+1}$.

Clipping

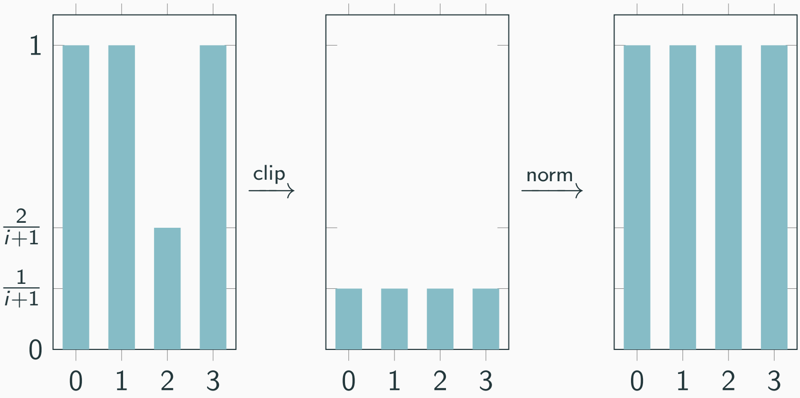
To test whether this is nonnegative, we construct a feed-forward layer that computes the function, given any input $S(i)$:

$$\text{gtz}\left(S(i), \frac{1}{i+1}\right) = \min\left(\frac{0.5}{i+1}, \frac{S(i)}{i+1} - \frac{0.5}{i+1}\right) - \min\left(0, \frac{S(i)}{i+1}\right)$$



Returning to Boolean Values

Use layer normalization to make everything ± 1.



Modal Depth

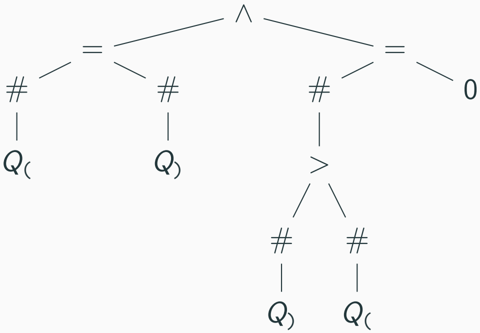
Observe that all count values become $\pm\frac{1}{i+1}$ after clipping and layer normalization. Thus we need to make sure that count values are never needed after a comparison.

It helps to organize the construction by modal depth The *modal depth* of a formula ϕ or term C, which we notate as $\text{md}(\phi)$, is the maximum level of nesting of $\#$ terms. That is,

$$
\begin{aligned}
\text{md}(Q_\sigma) &= 0 & \text{md}(C_1 + C_2) &= \max(\text{md}(C_1), \text{md}(C_2)) \\
\text{md}(1) &= 0 & \text{md}(C_1 \leq C_2) &= \max(\text{md}(C_1), \text{md}(C_2)) \\
\text{md}(\neg\phi) &= \text{md}(\phi) & \text{md}(\phi_1 \wedge \phi_2) &= \max(\text{md}(\phi_1), \text{md}(\phi_2)) \\
\text{md}(\#[\phi]) &= 1 + \text{md}(\phi)
\end{aligned}
$$

Modal Depth Example

$$\phi = (\#[Q_(] = \#[Q_)]) \land (\#[\#[Q_)] > \#[Q_(]] = 0)$$



Observe that the bottommost counts $\#[Q_(]$ and $\#[Q_)]$ are never needed after the comparison $<$

Main Result

Theorem

For every $\mathsf{K}_t[\#, +]$ *formula* ϕ*, there exists a masked transformer encoder which simulates* ϕ*.*

Proof.

Induct on modal depth of ϕ and apply the constructions described in previous slides! $\square$

Recap: Lower Bound of Soft Attention

- While AHATs can simulate LTL$[\#, +]$, there is currently no published proof that SMATs can simulate the full LTL logic.
- Uniform attention is used in SMATs to simulate counting operations.
- Layernorm is used to rescale very small values back to Booleans.

References i

Pablo Barceló, Alexander Kozachinskiy, Anthony Widjaja Lin, and Vladimir Podolskii. Logical languages accepted by transformer encoders with hard attention. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL `https://openreview.net/forum?id=gbrHZq07mq`.

David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990. doi:https://doi.org/10.1016/0022-0000(90)90022-D.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the Ability and Limitations of Transformers to Recognize Formal Languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7096–7116, 2020. doi:10.18653/v1/2020.emnlp-main.576.

David Chiang, Peter Cholak, and Anand Pillay. Tighter bounds on the expressivity of transformer encoders. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pages 5544–5562, 2023. URL `https://proceedings.mlr.press/v202/chiang23a.html`.

William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65(4):695–716, 2002. doi:https://doi.org/10.1016/S0022-0000(02)00025-9.

Neil Immerman. *Descriptive Complexity*. Springer, 1999.

Najoung Kim and Sebastian Schuster. Entity tracking in language models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3835–3855, 2023. doi:10.18653/v1/2023.acl-long.213. URL `https://aclanthology.org/2023.acl-long.213`.

Klaus-Jörn Lange. Some results on majority quantifiers over words. In *Proceedings of the 19th IEEE Annual Conference on Computational Complexity*, pages 123–129, 2004. doi:10.1109/CCC.2004.1313817.

References ii

William Merrill and Ashish Sabharwal. A logic for expressing log-precision transformers. In *Advances in Neural Information Processing Systems*, volume 36, pages 52453–52463, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/a48e5877c7bf86a513950ab23b360498-Abstract-Conference.html.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. In *Proc. ICML*, 2024. URL https://arxiv.org/abs/2404.08819.

Denis Paperno. On learning interpreted languages with recurrent models. *Computational Linguistics*, 48(2): 471–482, June 2022. doi:10.1162/coli_a_00431. URL https://aclanthology.org/2022.cl-2.7.

Johan van Benthem and Thomas Icard. Interleaving logic and counting. *The Bulletin of Symbolic Logic*, 29(4): 503–587, 2023. doi:10.1017/bsl.2023.30.

Andy Yang and David Chiang. Counting like transformers: Compiling temporal counting logic into softmax transformers. In *Proceedings of the Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2404.04393. To appear.