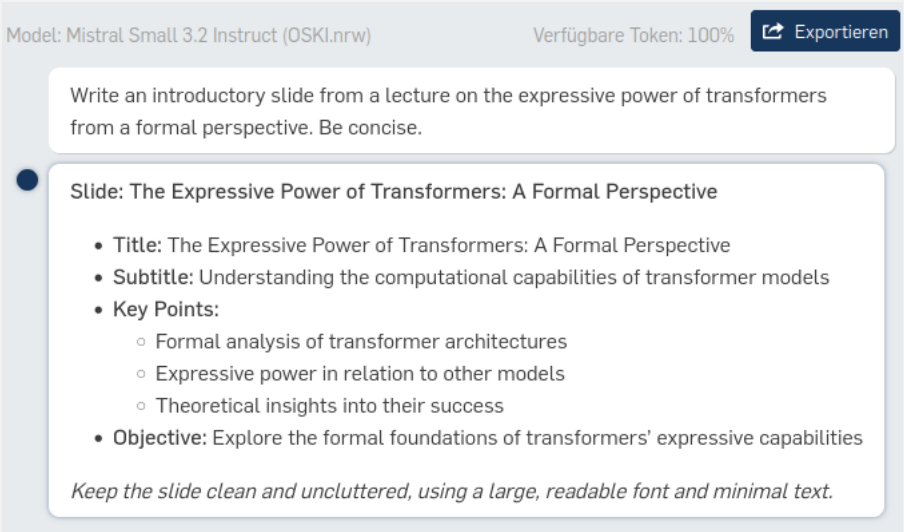
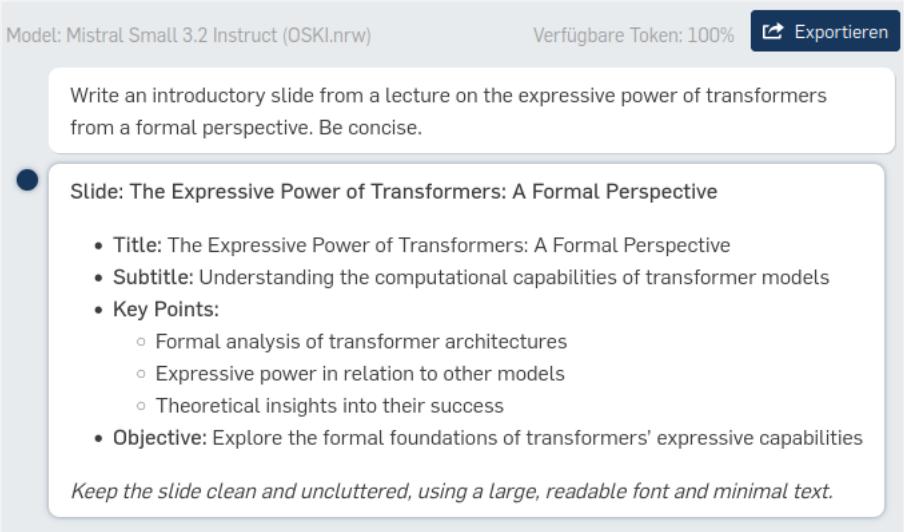
Chapter E.1:

Expressivity of transformers

Massive usage of LLMs



Massive usage of LLMs



On which tasks can we expect good results?

Analysis of transformers

Behind every recent model: a **transformer**

- an **attention mechanism** weigths the importance of different input elements
- better processing of **long-range dependencies**
- highly **parallelizable**

Analysis of transformers

Behind every recent model: a **transformer**

- an **attention mechanism** weigths the importance of different input elements
- better processing of **long-range dependencies**
- highly **parallelizable**

How to analyse transformers?

Analysis of transformers

Behind every recent model: a **transformer**

- an **attention mechanism** weigths the importance of different input elements
- better processing of **long-range dependencies**
- highly **parallelizable**

How to analyse transformers?

Empirically:

- **Benchmarking:** train the model on a corpus and evaluate it on a chosen benchmark

Analysis of transformers

Behind every recent model: a **transformer**

- an **attention mechanism** weigths the importance of different input elements
- better processing of **long-range dependencies**
- highly **parallelizable**

How to analyse transformers?

Empirically:

- **Benchmarking:** train the model on a corpus and evaluate it on a chosen benchmark
- **Exploration:** use correlation techniques to investigate unlown parameters of black-box LLM

Analysis of transformers

Behind every recent model: a **transformer**

- an **attention mechanism** weigths the importance of different input elements
- better processing of **long-range dependencies**
- highly **parallelizable**

How to analyse transformers?

Empirically:

- **Benchmarking:** train the model on a corpus and evaluate it on a chosen benchmark
- **Exploration:** use correlation techniques to investigate unlown parameters of black-box LLM

Theoretically:

1. Devise formal transformer models
2. Prove things!

Analysis of transformers

Behind every recent model: a **transformer**

- an **attention mechanism** weigths the importance of different input elements
- better processing of **long-range dependencies**
- highly **parallelizable**

How to analyse transformers?

Empirically:

- **Benchmarking:** train the model on a corpus and evaluate it on a chosen benchmark
- **Exploration:** use correlation techniques to investigate unlown parameters of black-box LLM

Theoretically:

1. Devise formal transformer models
2. Prove things!

Two directions:

- **Learnability:** What problems can or can't be solved after a training phase.
  $\longrightarrow$ Hard, the research is in an early stage

Analysis of transformers

Behind every recent model: a **transformer**

- an **attention mechanism** weigths the importance of different input elements
- better processing of **long-range dependencies**
- highly **parallelizable**

How to analyse transformers?

Empirically:

- **Benchmarking:** train the model on a corpus and evaluate it on a chosen benchmark
- **Exploration:** use correlation techniques to investigate unlown parameters of black-box LLM

Theoretically:

1. Devise formal transformer models
2. Prove things!

Two directions:

- **Learnability:** What problems can or can't be solved after a training phase.
  $\longrightarrow$ Hard, the research is in an early stage
- **Expressibility:** What can the model express?
  $\longrightarrow$ This lecture.
  $\longrightarrow$ not expressible $\Rightarrow$ not learnable

Analysis of transformers

Behind every recent model: a **transformer**

- an **attention mechanism** weigths the importance of different input elements
- better processing of **long-range dependencies**
- highly **parallelizable**

How to analyse transformers?

Empirically:

- **Benchmarking:** train the model on a corpus and evaluate it on a chosen benchmark
- **Exploration:** use correlation techniques to investigate unlown parameters of black-box LLM

Theoretically:

1. Devise formal transformer models
2. Prove things!

Two directions:

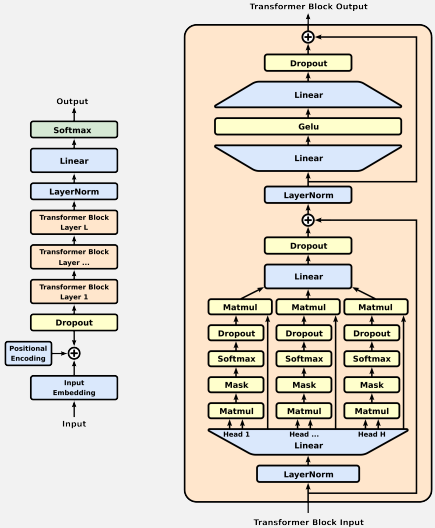
- **Learnability:** What problems can or can't be solved after a training phase.
  $\longrightarrow$ Hard, the research is in an early stage
- **Expressibility:** What can the model express?
  $\longrightarrow$ This lecture.
  $\longrightarrow$ not expressible $\Rightarrow$ not learnable

How to proceed?

- Model transformers are formal languages recognizers
- Import knowledge from **circuits, logic, automata,...**

Formal models for transformers

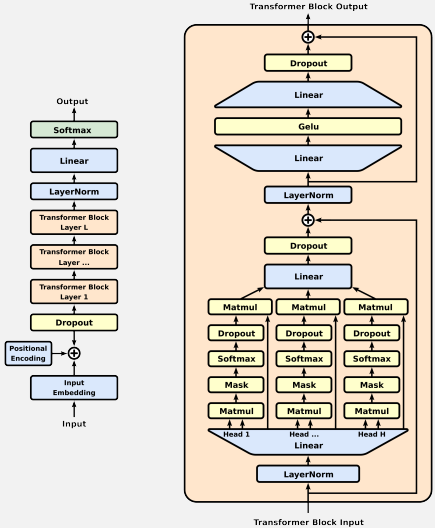
A formal model



GPT-3

Fix an alphabet Σ
A formal transformer will define a function $\Sigma^* \to \{0, 1\}$

A formal model

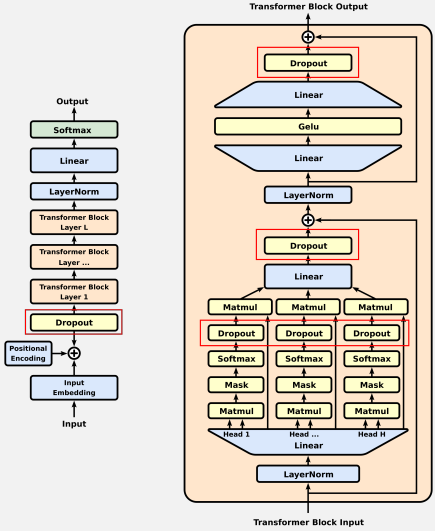


GPT-3

Fix an alphabet Σ
A formal transformer will define a function $\Sigma^* \to \{0, 1\}$

We will not model the following components, at least for now:

A formal model



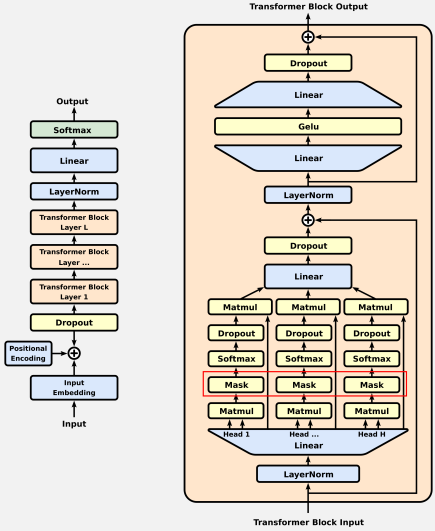
GPT-3

Fix an alphabet Σ
A formal transformer will define a function $\Sigma^* \to \{0,1\}$

We will not model the following components, at least for now:

- **Dropout:** Some connections are deactivated during learning to increase the robustess of the learning.
  $\to$ Irrelevant for expressivity

A formal model



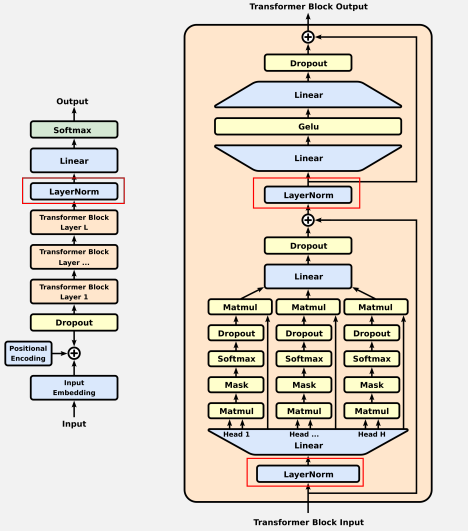
GPT-3

Fix an alphabet Σ
A formal transformer will define a function
$\Sigma^* \to \{0, 1\}$

We will not model the following components, at least for now:

- **Dropout:** Some connections are deactivated during learning to increase the robustess of the learning.
  $\to$ Irrelevant for expressivity
- **Masking:** Attention is restricted to previous inputs to help maintaing causality
  $\to$ Can increase or decrease expressivity

A formal model



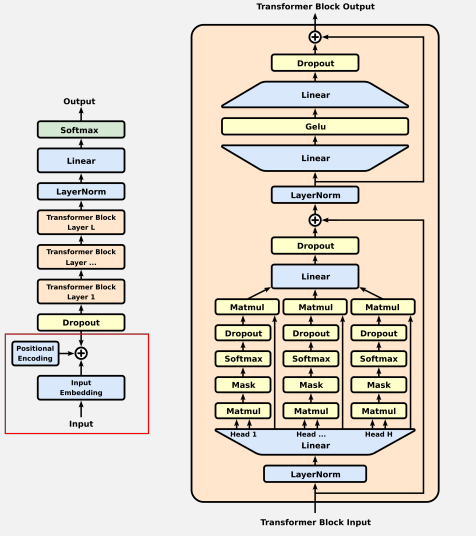
GPT-3

Fix an alphabet Σ
A formal transformer will define a function $\Sigma^* \to \{0, 1\}$

We will not model the following components, at least for now:

- **Dropout:** Some connections are deactivated during learning to increase the robustess of the learning.
  $\to$ Irrelevant for expressivity
- **Masking:** Attention is restricted to previous inputs to help maintaing causality
  $\to$ Can increase or decrease expressivity
- **Layer norm:** Normalization (mean $= 0$ and variance $= 1$) of the values of a layer, to speed-up the training.
  $\to$ Usually increase expressivity

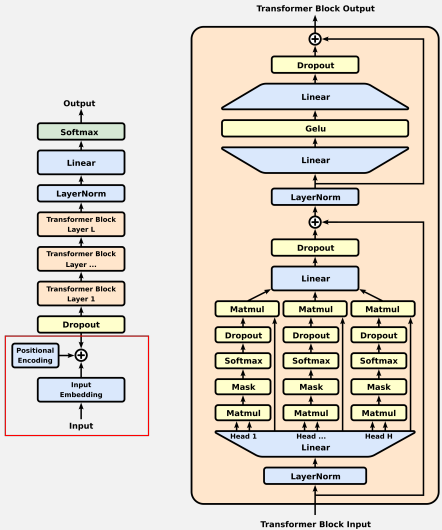
A formal model - input encoding



GPT-3

The transfomer receives an input $x_1 \cdots x_n \in \Sigma^*$

A formal model - input encoding

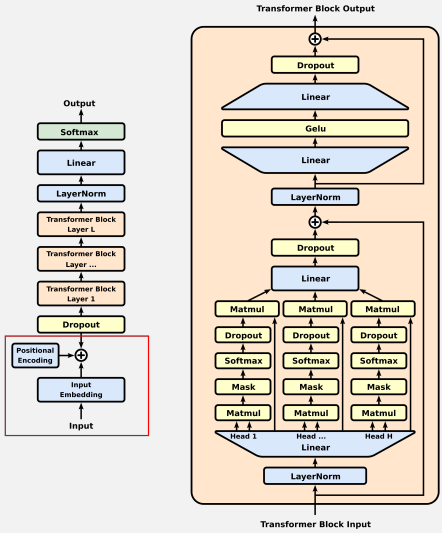


GPT-3

The transfomer receives an input $x_1 \cdots x_n \in \Sigma^*$

It will work with values in $\mathbb{R}$, for a global parameter d

A formal model - input encoding



GPT-3

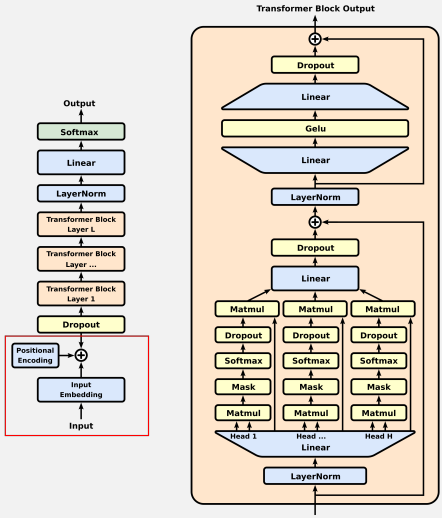
The transfomer receives an input $x_1 \cdots x_n \in \Sigma^*$

It will work with values in $\mathbb{R}$, for a global parameter d

First, the input in Σ^n must be translated into values in $(\mathbb{R}^d)^n$. Two ingredients:

- **Word embedding:** a function $\mathbf{we} : \Sigma \to \mathbb{R}^d$
- **Positional endoding:** a function $\mathbf{pe} : \{1, \ldots, n\} \to \mathbb{R}^d$

A formal model - input encoding



GPT-3

The transfomer receives an input $x_1 \cdots x_n \in \Sigma^*$

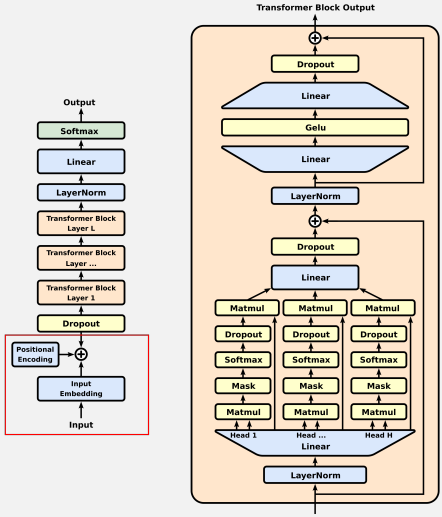
It will work with values in $\mathbb{R}$, for a global parameter d

First, the input in Σ^n must be translated into values in $(\mathbb{R}^d)^n$. Two ingredients:

- **Word embedding:** a function $\mathbf{we} : \Sigma \to \mathbb{R}^d$
- **Positional endoding:** a function $\mathbf{pe} : \{1, \ldots, n\} \to \mathbb{R}^d$

Define $y_i = \mathbf{we}(x_i) + \mathbf{pe}(i)$

A formal model - input encoding



GPT-3

The transfomer receives an input $x_1 \cdots x_n \in \Sigma^*$

It will work with values in $\mathbb{R}$, for a global parameter d

First, the input in Σ^n must be translated into values in $(\mathbb{R}^d)^n$. Two ingredients:

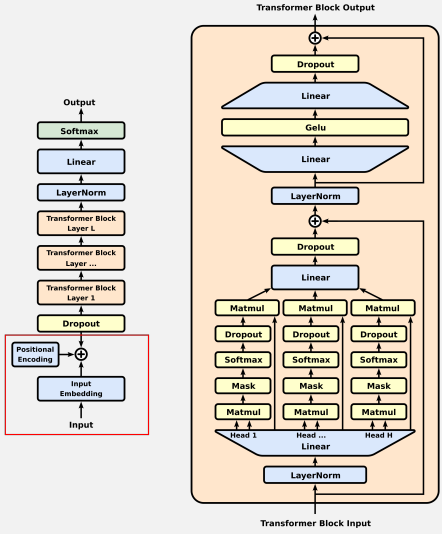
- **Word embedding:** a function $\mathbf{we} : \Sigma \to \mathbb{R}^d$
- **Positional endoding:** a function $\mathbf{pe} : \{1, \ldots, n\} \to \mathbb{R}^d$

Define $y_i = \mathbf{we}(x_i) + \mathbf{pe}(i)$

The original encoding:

$$\mathbf{pe}(i)[j] = \begin{cases} \sin\left(\frac{i}{10000^{j/d}}\right) & \text{if } j \text{ is even} \\ \cos\left(\frac{i}{10000^{j/d}}\right) & \text{if } j \text{ is odd} \end{cases}$$

A formal model - input encoding



GPT-3

The transfomer receives an input $x_1 \cdots x_n \in \Sigma^*$

It will work with values in $\mathbb{R}$, for a global parameter d

First, the input in Σ^n must be translated into values in $(\mathbb{R}^d)^n$. Two ingredients:

- **Word embedding:** a function $\mathbf{we} : \Sigma \to \mathbb{R}^d$
- **Positional endoding:** a function $\mathbf{pe} : \{1, \ldots, n\} \to \mathbb{R}^d$

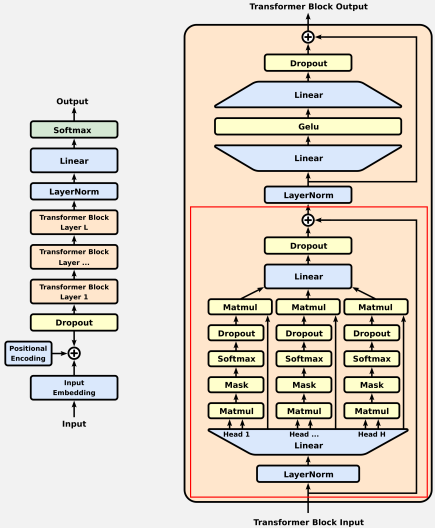
Define $y_i = \mathbf{we}(x_i) + \mathbf{pe}(i)$

The original encoding:

$$\mathbf{pe}(i)[j] = \begin{cases} \sin\left(\frac{i}{10000^{j/d}}\right) & \text{if } j \text{ is even} \\ \cos\left(\frac{i}{10000^{j/d}}\right) & \text{if } j \text{ is odd} \end{cases}$$

Many other ones: i, $\frac{i}{n}$, $\frac{1}{i}$, $\frac{1}{i^2}$, no $\mathbf{pe}$, ...

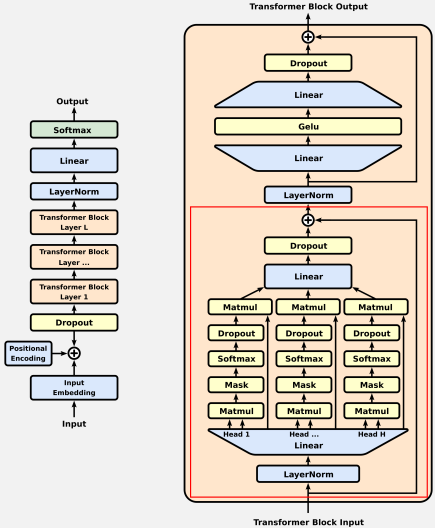
A formal model - attention mechanism



GPT-3

So far: $y_1, \ldots, y_n \in (\mathbb{R}^d)^n$

A formal model - attention mechanism

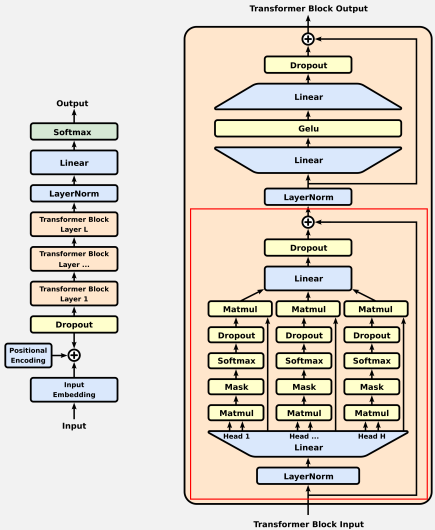


GPT-3

So far: $y_1, \ldots, y_n \in (\mathbb{R}^d)^n$

Parameters Q, K **and** V: matrices of size $d \times d$ $\longrightarrow$ query, key and value

A formal model - attention mechanism



GPT-3

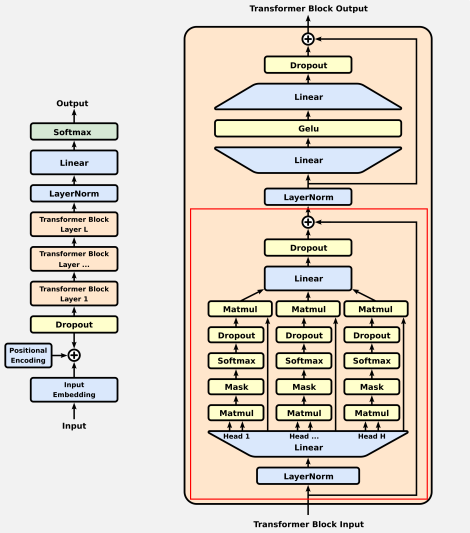
So far: $y_1, \ldots, y_n \in (\mathbb{R}^d)^n$

Parameters Q, K **and** V: matrices of size $d \times d$ $\quad \longrightarrow$ query, key and value

The **softmax** function approximates max:

$$\mathbf{softmax}(s_1, \ldots, s_n) = \frac{1}{\sum s_i}(e^{s_1}, \ldots, e^{s_n})$$

A formal model - attention mechanism



GPT-3

So far: $y_1, \ldots, y_n \in (\mathbb{R}^d)^n$

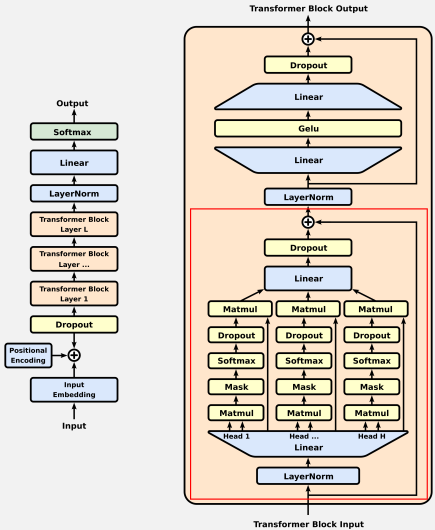
Parameters Q, K **and** V: matrices of size $d \times d$ $\longrightarrow$ query, key and value

The **softmax** function approximates max:

$$\mathbf{softmax}(s_1, \ldots, s_n) = \frac{1}{\sum s_i}(e^{s_1}, \ldots, e^{s_n})$$

- For $i \leq i \leq n$, compute $q_i = Qy_i$, $k_i = Ky_i$ and $v_i = Vy_i$

A formal model - attention mechanism



GPT-3

So far: $y_1, \ldots, y_n \in (\mathbb{R}^d)^n$

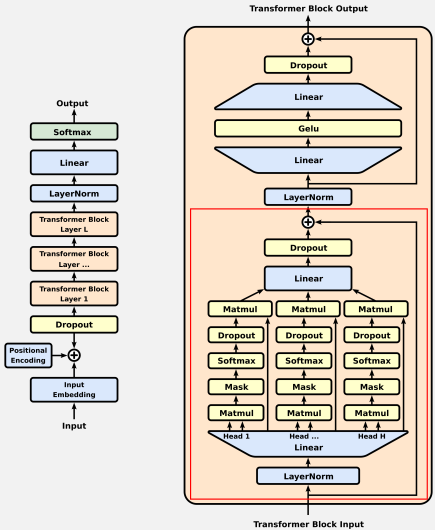
Parameters Q, K **and** V: matrices of size $d \times d$ $\quad \rightarrow$ query, key and value

The **softmax** function approximates max:

$$\mathbf{softmax}(s_1, \ldots, s_n) = \frac{1}{\sum s_i}(e^{s_1}, \ldots, e^{s_n})$$

- For $i \leq i \leq n$, compute $q_i = Qy_i$, $k_i = Ky_i$ and $v_i = Vy_i$
- **Attention scores:** $s_{i,j} = q_i^\top k_j$

A formal model - attention mechanism



GPT-3

So far: $y_1, \ldots, y_n \in (\mathbb{R}^d)^n$

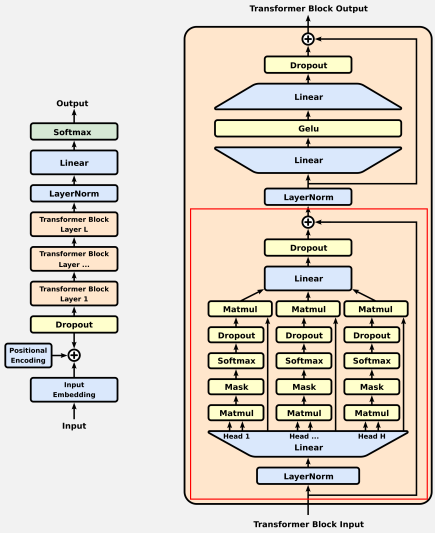
Parameters Q, K **and** V: matrices of size $d \times d$ $\longrightarrow$ query, key and value

The **softmax** function approximates max:

$$\mathbf{softmax}(s_1, \ldots, s_n) = \frac{1}{\sum s_i}(e^{s_1}, \ldots, e^{s_n})$$

- For $i \leq i \leq n$, compute $q_i = Qy_i$, $k_i = Ky_i$ and $v_i = Vy_i$
- **Attention scores:** $s_{i,j} = q_i^\top k_j$
- **Attention weights:** $\alpha_{i,1}, \ldots, \alpha_{i,n} = \mathbf{softmax}(s_{i,1}, \ldots, s_{i,n})$

A formal model - attention mechanism



GPT-3

So far: $y_1, \ldots, y_n \in (\mathbb{R}^d)^n$

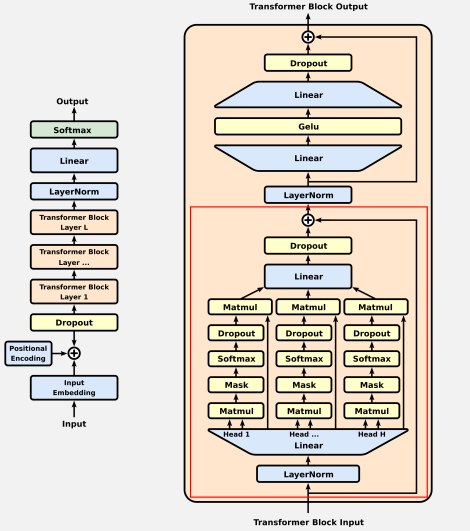
Parameters Q, K **and** V: matrices of size $d \times d$ $\quad \rightarrow$ query, key and value

The **softmax** function approximates max:

$$\mathbf{softmax}(s_1, \ldots, s_n) = \frac{\mathbf{1}}{\sum s_i}(e^{s_1}, \ldots, e^{s_n})$$

- For $i \leq i \leq n$, compute $q_i = Qy_i$, $k_i = Ky_i$ and $v_i = Vy_i$
- **Attention scores:** $s_{i,j} = q_i^\top k_j$
- **Attention weights:** $\alpha_{i,1}, \ldots, \alpha_{i,n} = \mathbf{softmax}(s_{i,1}, \ldots, s_{i,n})$
- $z_i = \sum_{i=1}^{n} \alpha_{i,j} v_j \quad + y_i$

A formal model - attention mechanism



GPT-3

So far: $y_1, \ldots, y_n \in (\mathbb{R}^d)^n$

Parameters Q, K **and** V: matrices of size $d \times d$ $\quad \rightarrow$ query, key and value

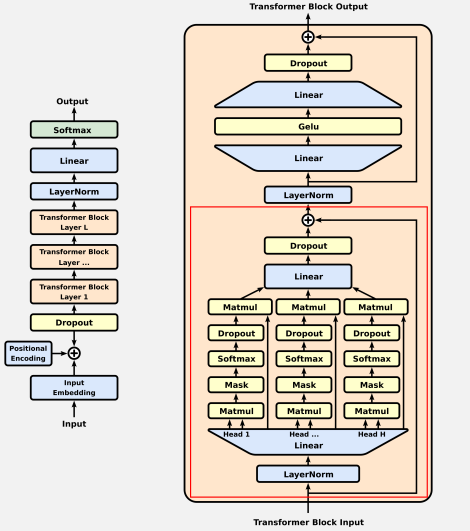
The **softmax** function approximates max:

$$\mathbf{softmax}(s_1, \ldots, s_n) = \frac{\mathbf{1}}{\sum s_i}(e^{s_1}, \ldots, e^{s_n})$$

- For $i \leq i \leq n$, compute $q_i = Qy_i$, $k_i = Ky_i$ and $v_i = Vy_i$
- **Attention scores:** $s_{i,j} = q_i^\top k_j$
- **Attention weights:** $\alpha_{i,1}, \ldots, \alpha_{i,n} = \mathbf{softmax}(s_{i,1}, \ldots, s_{i,n})$
- $z_i = \sum_{i=1}^{n} \alpha_{i,j} v_j \quad + y_i$

In practice, many αs are close to 0:
Unique hard-attention: replace softmax by the function that assigns $\mathbf{1}$ to the leftmost maximal position

A formal model - attention mechanism



GPT-3

So far: $y_1, \ldots, y_n \in (\mathbb{R}^d)^n$

Parameters Q, K **and** V: matrices of size $d \times d$ $\quad \rightarrow$ query, key and value

The **softmax** function approximates max:

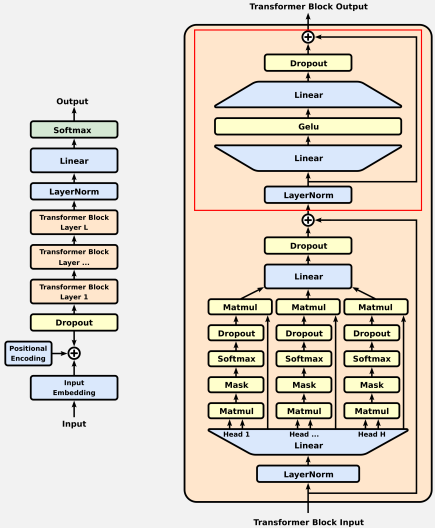
$$\mathbf{softmax}(s_1, \ldots, s_n) = \frac{\mathbf{1}}{\sum s_i}(e^{s_1}, \ldots, e^{s_n})$$

- For $i \leq i \leq n$, compute $q_i = Qy_i$, $k_i = Ky_i$ and $v_i = Vy_i$
- **Attention scores:** $s_{i,j} = q_i^\top k_j$
- **Attention weights:** $\alpha_{i,1}, \ldots, \alpha_{i,n} = \mathbf{softmax}(s_{i,1}, \ldots, s_{i,n})$
- $z_i = \sum_{i=1}^{n} \alpha_{i,j} v_j \quad + y_i$

In practice, many αs are close to 0:
Unique hard-attention: replace softmax by the function that assigns $\mathbf{1}$ to the leftmost maximal position

Multi-head does not add expressivity
$\rightarrow$ Use one layer per head

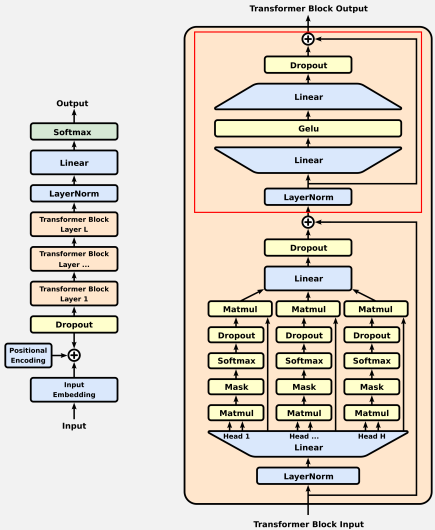
A formal model - feed-forward network



GPT-3

So far: $z_1, \ldots, z_n \in (\mathbb{R}^d)^n$

A formal model - feed-forward network



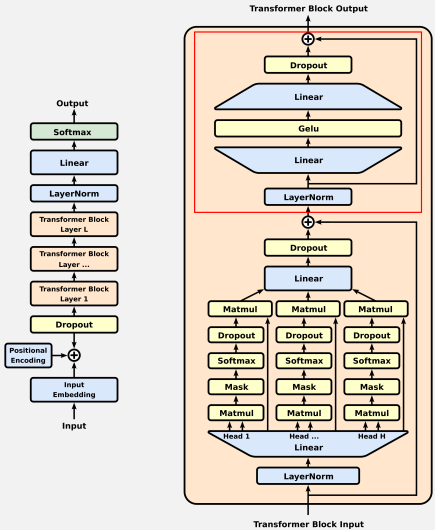
GPT-3

So far: $z_1, \ldots, z_n \in (\mathbb{R}^d)^n$

This will be processed by the most basic neural network: a **FFN**
It depends on:

- a hidden dimension d'
- matrices $W_1 \in \mathbb{R}^{d' \times d}$ and $W_2 \in \mathbb{R}^{d \times d'}$
- vectors $b_1 \in \mathbb{R}^{d'}$ and $b_2 \in \mathbb{R}^d$

A formal model - feed-forward network



GPT-3

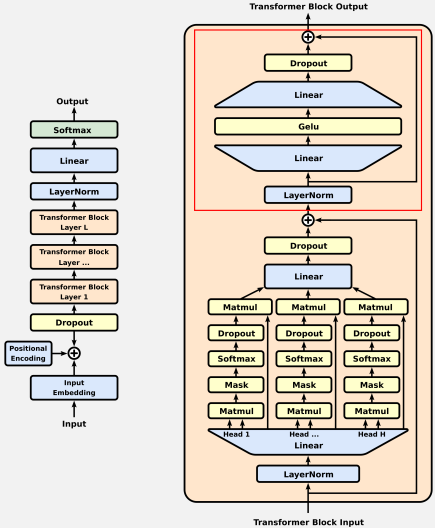
So far: $z_1, \ldots, z_n \in (\mathbb{R}^d)^n$

This will be processed by the most basic neural network: a **FFN**
It depends on:

- a hidden dimension d'
- matrices $W_1 \in \mathbb{R}^{d' \times d}$ and $W_2 \in \mathbb{R}^{d \times d'}$
- vectors $b_1 \in \mathbb{R}^{d'}$ and $b_2 \in \mathbb{R}^d$

The **rectified linear unit:** $\mathbb{R}^d \to \mathbb{R}^d$ is $\mathbf{ReLU}(x) = \max(0, x)$ applied component-wise

A formal model - feed-forward network



GPT-3

So far: $z_1, \ldots, z_n \in (\mathbb{R}^d)^n$

This will be processed by the most basic neural network: a **FFN**
It depends on:

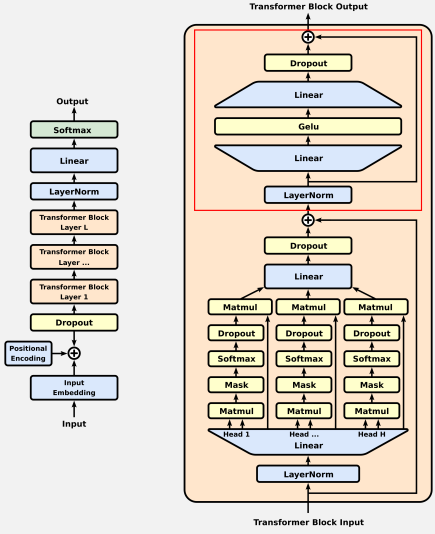
- a hidden dimension d'
- matrices $W_1 \in \mathbb{R}^{d' \times d}$ and $W_2 \in \mathbb{R}^{d \times d'}$
- vectors $b_1 \in \mathbb{R}^{d'}$ and $b_2 \in \mathbb{R}^d$

The **rectified linear unit:** $\mathbb{R}^d \to \mathbb{R}^d$ is $\mathbf{ReLU}(x) = \max(0, x)$ applied component-wise

For $1 \leq i \leq n$, define

- $h_i = W_1 z_i + b_1$
- $g_i = \mathbf{ReLU}(h_i)$
- $y_i^{(1)} = W_2 g_i + b_2 + z_i$

A formal model - feed-forward network



GPT-3

So far: $z_1, \ldots, z_n \in (\mathbb{R}^d)^n$

This will be processed by the most basic neural network: a **FFN**
It depends on:

- a hidden dimension d'
- matrices $W_1 \in \mathbb{R}^{d' \times d}$ and $W_2 \in \mathbb{R}^{d \times d'}$
- vectors $b_1 \in \mathbb{R}^{d'}$ and $b_2 \in \mathbb{R}^d$

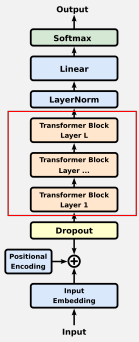
The **rectified linear unit:** $\mathbb{R}^d \to \mathbb{R}^d$ is $\mathbf{ReLU}(x) = \max(0, x)$ applied component-wise

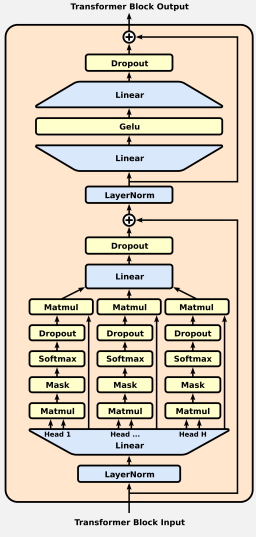
For $1 \leq i \leq n$, define

- $h_i = W_1 z_i + b_1$
- $g_i = \mathbf{ReLU}(h_i)$
- $y_i^{(1)} = W_2 g_i + b_2 + z_i$

It is known that **any function can be approximated** by a (multi-layer) FFN

A formal model - binary output

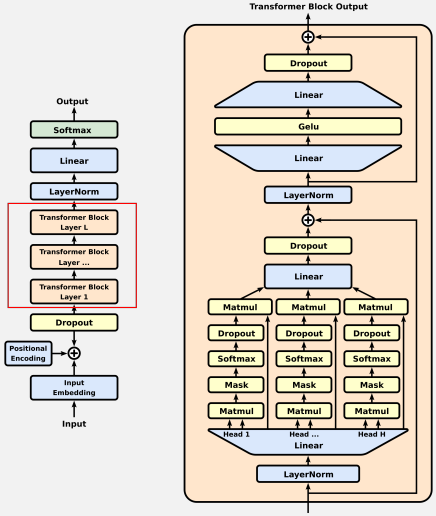




GPT-3

So far: $\boldsymbol{y}_1^{(1)}, \ldots, \boldsymbol{y}_n^{(1)} \in (\mathbb{R}^d)^n$

A formal model - binary output

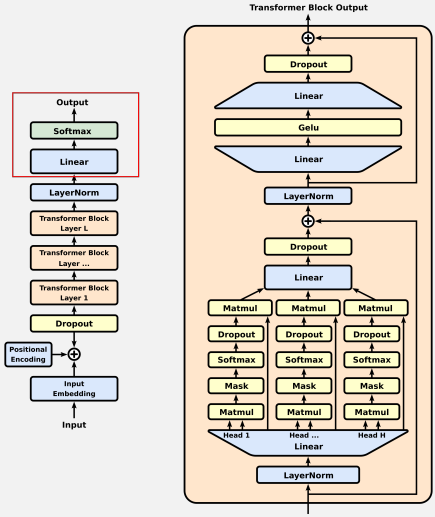


GPT-3

So far: $\boldsymbol{y_1^{(1)}, \ldots, y_n^{(1)} \in (\mathbb{R}^d)^n}$

For a parameter $\boldsymbol{L}$, we repeat this for $\boldsymbol{L}$ layers giving $\boldsymbol{y_1^{(L)}, \ldots, y_n^{(L)} \in (\mathbb{R}^d)^n}$

A formal model - binary output



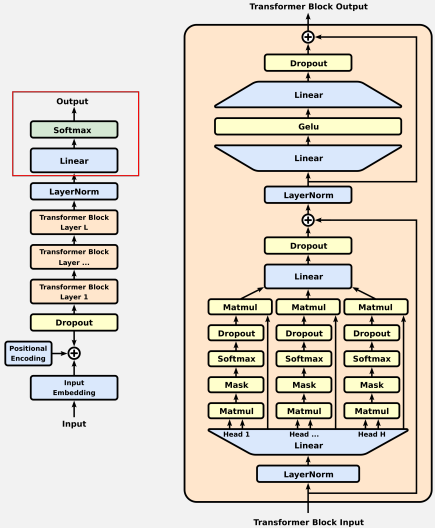
GPT-3

So far: $\boldsymbol{y_1^{(1)}, \ldots, y_n^{(1)} \in (\mathbb{R}^d)^n}$

For a parameter $\boldsymbol{L}$, we repeat this for $\boldsymbol{L}$ layers giving $\boldsymbol{y_1^{(L)}, \ldots, y_n^{(L)} \in (\mathbb{R}^d)^n}$

In practive generate, every $\boldsymbol{y_i^{(L)}}$ is mapped to a probability distribution via a softmax to generate the output.

A formal model - binary output



GPT-3

So far: $\boldsymbol{y_1^{(1)}, \ldots, y_n^{(1)} \in (\mathbb{R}^d)^n}$

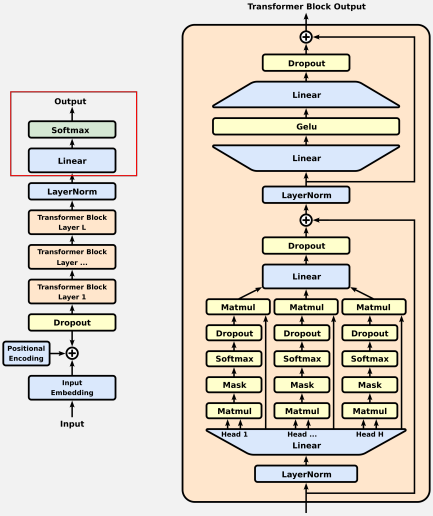
For a parameter $\boldsymbol{L}$, we repeat this for $\boldsymbol{L}$ layers giving $\boldsymbol{y_1^{(L)}, \ldots, y_n^{(L)} \in (\mathbb{R}^d)^n}$

In practive generate, every $\boldsymbol{y_i^{(L)}}$ is mapped to a probability distribution via a softmax to generate the output.

To have a language recognizer, consider classification:

- a vector $\boldsymbol{o \in \mathbb{R}^{1 \times d}}$
- Accept if $\boldsymbol{o^\top \cdot y_n^{(L)} > 0}$

A formal model - binary output



GPT-3

So far: $\boldsymbol{y_1^{(1)}, \ldots, y_n^{(1)} \in (\mathbb{R}^d)^n}$

For a parameter $\boldsymbol{L}$, we repeat this for $\boldsymbol{L}$ layers giving $\boldsymbol{y_1^{(L)}, \ldots, y_n^{(L)} \in (\mathbb{R}^d)^n}$

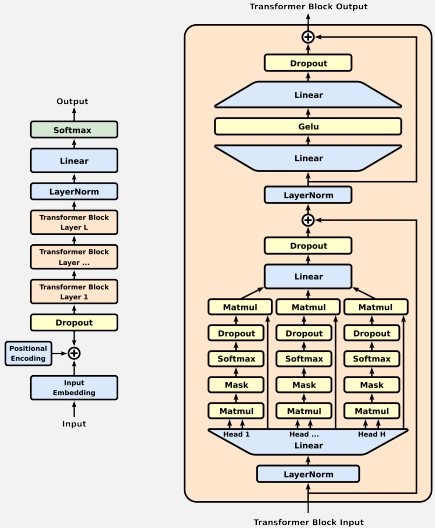
In practive generate, every $\boldsymbol{y_i^{(L)}}$ is mapped to a probability distribution via a softmax to generate the output.

To have a language recognizer, consider classification:

- a vector $\boldsymbol{o \in \mathbb{R}^{1 \times d}}$
- Accept if $\boldsymbol{o^\top \cdot y_n^{(L)} > 0}$

We defined a function $\boldsymbol{\Sigma^* \to \{0, 1\}}$

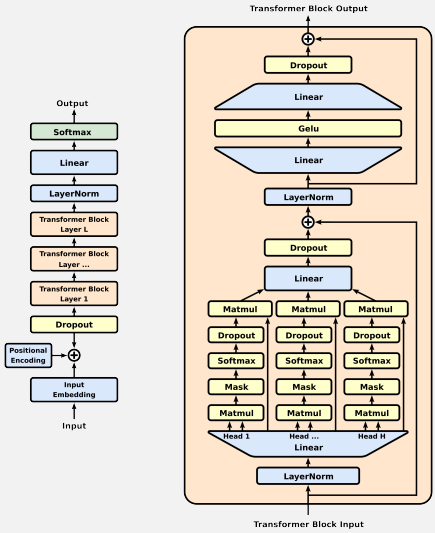
A formal model - choice of parameters



GPT-3

Many, many choices:

A formal model - choice of parameters

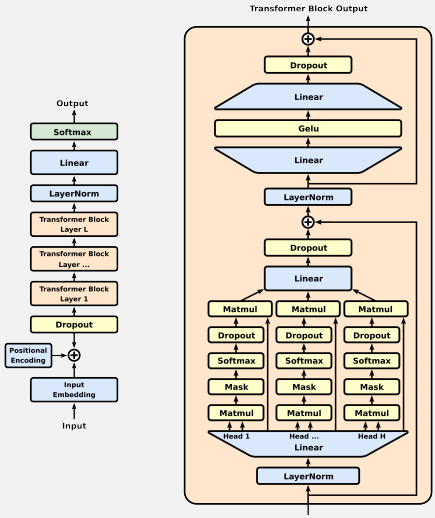


GPT-3

Many, many choices:

- How many layers? What is the dimension d?

A formal model - choice of parameters

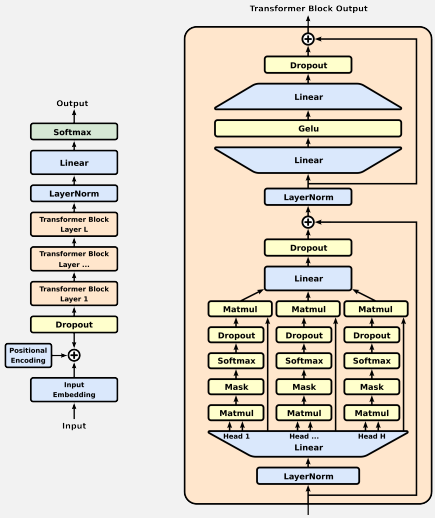


GPT-3

Many, many choices:

- How many layers? What is the dimension d?
- What kind of attention? We use **UHAT** for unique hard-attention and **SMAT** for softmax attention.

A formal model - choice of parameters

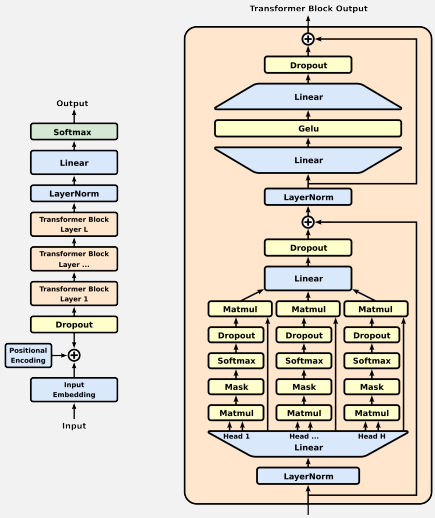


GPT-3

Many, many choices:

- How many layers? What is the dimension d?
- What kind of attention? We use **UHAT** for unique hard-attention and **SMAT** for softmax attention.
- What is the precision of the reals? It is usual to restrict which real numbers can be used. Typically: **fixed**, **log** or **arbitrary** precision.

A formal model - choice of parameters

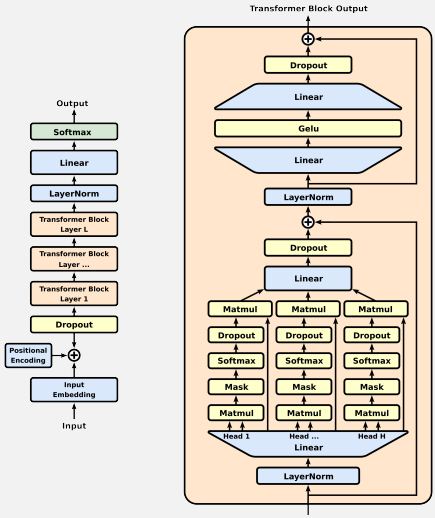


GPT-3

Many, many choices:

- How many layers? What is the dimension d?
- What kind of attention? We use **UHAT** for unique hard-attention and **SMAT** for softmax attention.
- What is the precision of the reals? It is usual to restrict which real numbers can be used. Typically: **fixed**, **log** or **arbitrary** precision.
- What are the positional encodings allowed?

A formal model - choice of parameters

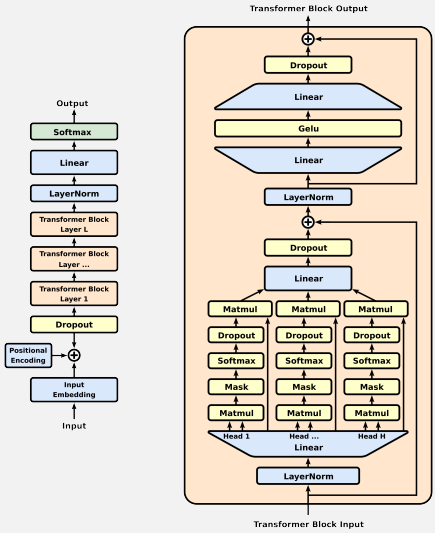


GPT-3

Many, many choices:

- How many layers? What is the dimension d?
- What kind of attention? We use **UHAT** for unique hard-attention and **SMAT** for softmax attention.
- What is the precision of the reals? It is usual to restrict which real numbers can be used. Typically: **fixed**, **log** or **arbitrary** precision.
- What are the positional encodings allowed?
- Is there masking? What kind?

A formal model - choice of parameters

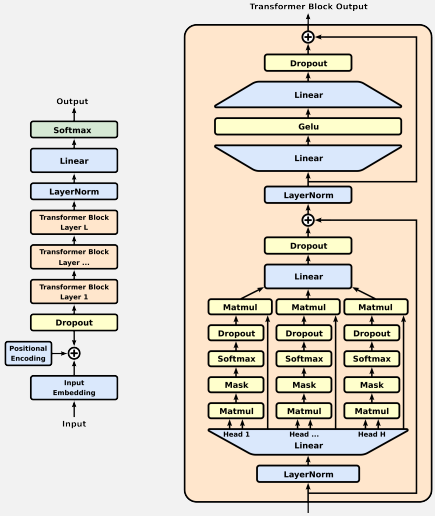


GPT-3

Many, many choices:

- How many layers? What is the dimension d?
- What kind of attention? We use **UHAT** for unique hard-attention and **SMAT** for softmax attention.
- What is the precision of the reals? It is usual to restrict which real numbers can be used. Typically: **fixed**, **log** or **arbitrary** precision.
- What are the positional encodings allowed?
- Is there masking? What kind?
- Is there a layer norm? At what level?

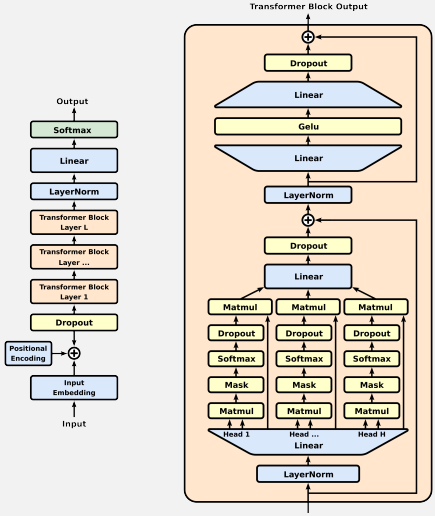
A formal model - recap



GPT-3

A transformers in given as: $d, d', L \in \mathbb{N}$, $\mathbf{we} : \Sigma \to \mathbb{R}^d$, $\mathbf{pe} : \{1, \ldots, n\} \to \mathbb{R}^d$, $Q^{(1)}, \ldots, Q^{(L)}$, $K^{(1)}, \ldots, K^{(L)}, V^{(1)}, \ldots, V^{(L)} \in \mathbb{R}^{d \times d}$, $W_1^{(1)}, \ldots, W_1^{(L)} \in \mathbb{R}^{d' \times d}$, $W_2^{(1)}, \ldots, W_2^{(L)} \in \mathbb{R}^{d \times d'}$, $b_1^{(1)}, \ldots, b_1^{(L)} \in \mathbb{R}^{d'}$, $b_2^{(1)}, \ldots, b_2^{(L)} \in \mathbb{R}^d$, $o \in \mathbb{R}^d$.

A formal model - recap



GPT-3

A transformers in given as: $d, d', L \in \mathbb{N}$, $\mathbf{we} : \Sigma \to \mathbb{R}^d$, $\mathbf{pe} : \{1, \ldots, n\} \to \mathbb{R}^d$, $Q^{(1)}, \ldots, Q^{(L)}$, $K^{(1)}, \ldots, K^{(L)}, V^{(1)}, \ldots, V^{(L)} \in \mathbb{R}^{d \times d}$, $W_1^{(1)}, \ldots, W_1^{(L)} \in \mathbb{R}^{d' \times d}$, $W_2^{(1)}, \ldots, W_2^{(L)} \in \mathbb{R}^{d \times d'}$, $b_1^{(1)}, \ldots, b_1^{(L)} \in \mathbb{R}^{d'}$, $b_2^{(1)}, \ldots, b_2^{(L)} \in \mathbb{R}^d$, $o \in \mathbb{R}^d$.

On input $x_1 \cdots x_n \in \Sigma^*$:

- $y_i = \mathbf{we}(x_i) + \mathbf{pe}(i)$
- $q_i^{(1)} = Q^{(1)} y_i$, $k_i^{(1)} = K^{(1)} y_i$ and $v_i^{(1)} = V^{(1)} y_i$
- $s_{i,j}^{(1)} = q_i^{(1)} k_j^{(1)}$
- $\alpha_{i,1}^{(1)}, \ldots, \alpha_{i,n}^{(1)} = \mathbf{softmax}(s_{i,1}^{(1)}, \ldots, s_{i,n}^{(1)})$
- $z_i^{(1)} = \sum_{i=1}^{n} \alpha_{i,j}^{(1)} v_j^{(1)} + y_i$
- $y_i^{(1)} = W_2^{(1)} \mathbf{ReLU}(W_1^{(1)} z_i^{(1)} + b_1^{(1)}) + b_2^{(1)} + z_i^{(1)}$
- $\ldots$
- $y_i^{(L)} = W_2^{(L)} \mathbf{ReLU}(W_1^{(L)} z_i^{(L)} + b_1^{(L)}) + b_2^{(L)} + z_i^{(L)}$
- Accept if $o^\top \cdot y_n^{(L)} > 0$

Some examples

A first example

For $\boldsymbol{E}$ an expression, $\mathbb{I}(\boldsymbol{E})$ is 1 if $\boldsymbol{E}$ is true and $\mathbf{0}$ otherwise

Example

MAJ (the language of strings with more 1s than 0s) can be recognized by a **SMAT** with **no PE** and **fixed precision**.

A first example

For $\boldsymbol{E}$ an expression, $\mathbb{I}(\boldsymbol{E})$ is 1 if $\boldsymbol{E}$ is true and $\mathbf{0}$ otherwise

Example

MAJ (the language of strings with more 1s than 0s) can be recognized by a **SMAT** with **no PE** and **fixed precision**.

Proof

The SMAT receives $\boldsymbol{x_1 \cdots x_n}$ with $\boldsymbol{k}$ ones. Fix $\boldsymbol{L = 1}$ and $\boldsymbol{d = 2}$

A first example

For E an expression, $\mathbb{I}(E)$ is 1 if E is true and 0 otherwise

Example

MAJ (the language of strings with more 1s than 0s) can be recognized by a **SMAT** with **no PE** and **fixed precision**.

Proof

The SMAT receives $x_1 \cdots x_n$ with k ones. Fix $L = 1$ and $d = 2$

Word embedding: Map $1 \mapsto (1, 0)$ and $0 \mapsto (-1, 0)$

A first example

For E an expression, $\mathbb{I}(E)$ is 1 if E is true and 0 otherwise

Example

MAJ (the language of strings with more 1s than 0s) can be recognized by a **SMAT** with **no PE** and **fixed precision**.

Proof

The SMAT receives $x_1 \cdots x_n$ with k ones. Fix $L = 1$ and $d = 2$

Word embedding: Map $1 \mapsto (1, 0)$ and $0 \mapsto (-1, 0)$

Attention Layer: Q and K are the all 0 matrices
All attention scores are $s_{i,j} = 0$ and attention weigths are $\alpha_{i,j} = \frac{1}{n}$
$V = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ and $z_i = (2\mathbb{I}(x_i = 1) - 1, \frac{2k-n}{n})$

A first example

For E an expression, $\mathbb{I}(E)$ is 1 if E is true and 0 otherwise

Example

MAJ (the language of strings with more 1s than 0s) can be recognized by a **SMAT** with **no PE** and **fixed precision**.

Proof

The SMAT receives $x_1 \cdots x_n$ with k ones. Fix $L = 1$ and $d = 2$

Word embedding: Map $1 \mapsto (1, 0)$ and $0 \mapsto (-1, 0)$

Attention Layer: Q and K are the all 0 matrices
All attention scores are $s_{i,j} = 0$ and attention weigths are $\alpha_{i,j} = \frac{1}{n}$
$V = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ and $z_i = (2\mathbb{I}(x_i = 1) - 1, \frac{2k-n}{n})$

FFN: Do nothing! W_2 and b_2 are all 0
Thus $y_i^{(1)} = z_i$

A first example

For E an expression, $\mathbb{I}(E)$ is 1 if E is true and 0 otherwise

Example

MAJ (the language of strings with more 1s than 0s) can be recognized by a **SMAT** with **no PE** and **fixed precision**.

Proof

The SMAT receives $x_1 \cdots x_n$ with k ones. Fix $L = 1$ and $d = 2$

Word embedding: Map $1 \mapsto (1, 0)$ and $0 \mapsto (-1, 0)$

Attention Layer: Q and K are the all 0 matrices
All attention scores are $s_{i,j} = 0$ and attention weigths are $\alpha_{i,j} = \frac{1}{n}$
$V = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ and $z_i = (2\mathbb{I}(x_i = 1) - 1, \frac{2k-n}{n})$

FFN: Do nothing! W_2 and b_2 are all 0
Thus $y_i^{(1)} = z_i$

Output: $o = (0, 1)$ and $o^\top y_n^{(1)} = \frac{2k-n}{n}$
It is > 0 iff the input is in MAJ

Parity

Example

PARITY can be recognized by a **SMAT** with the **PEs** $\frac{i}{n}$ **and** $(-1)^i$ and **fixed precision**.

Parity

Example

PARITY can be recognized by a **SMAT** with the **PEs** $\frac{i}{n}$ **and** $(-1)^i$ and **fixed precision**.

Proof

The SMAT receives $x_1 \cdots x_n$ with k ones.
Fix $L = 3$ and $d = 11$

Parity

Example

PARITY can be recognized by a **SMAT** with the **PEs** $\frac{i}{n}$ **and** $(-1)^i$ and **fixed precision**.

Proof

The SMAT receives $x_1 \cdots x_n$ with k ones.
Fix $L = 3$ and $d = 11$

Embeddings: Map $0 \mapsto (1, 0, \ldots, 0)$ and $1 \mapsto (0, 1, 0, \ldots, 0)$
So $y_i = (\mathbb{I}(x_i{=}0), \mathbb{I}(x_i{=}1), \frac{i}{n}, (-1)^i, \frac{1}{n} \ldots)$

Parity

Example

PARITY can be recognized by a **SMAT** with the **PEs** $\frac{i}{n}$ **and** $(-1)^i$ and **fixed precision**.

Proof

The SMAT receives $x_1 \cdots x_n$ with k ones.
Fix $L = 3$ and $d = 11$

Embeddings: Map $0 \mapsto (1, 0, \ldots, 0)$ and $1 \mapsto (0, 1, 0, \ldots, 0)$
So $y_i = (\mathbb{I}(x_i{=}0), \mathbb{I}(x_i{=}1), \frac{i}{n}, (-1)^i, \frac{1}{n} \ldots)$

Attention layer: Q, K and V are all 0 except $V_{6,2} = 1$
So $z_i = (\mathbb{I}(x_i = 0), \mathbb{I}(x_i = 1), \frac{i}{n}, (-1)^i, \frac{1}{n}, \frac{k}{n} \ldots)$

Parity

Example

PARITY can be recognized by a **SMAT** with the **PEs** $\frac{i}{n}$ **and** $(-1)^i$ and **fixed precision**.

Proof

The SMAT receives $x_1 \cdots x_n$ with k ones.
Fix $L = 3$ and $d = 11$

Embeddings: Map $0 \mapsto (1, 0, \ldots, 0)$ and $1 \mapsto (0, 1, 0, \ldots, 0)$
So $y_i = (\mathbb{I}(x_i{=}0), \mathbb{I}(x_i{=}1), \frac{i}{n}, (-1)^i, \frac{1}{n} \ldots)$

Attention layer: Q, K and V are all 0 except $V_{6,2} = 1$
So $z_i = (\mathbb{I}(x_i = 0), \mathbb{I}(x_i = 1), \frac{i}{n}, (-1)^i, \frac{1}{n}, \frac{k}{n} \ldots)$

FFN: With $d' = 3$:
$g_i = \frac{1}{n}(\max(0, k - i - 1), \max(0, k - i), \max(0, k - i + 1))$

Parity

Example

PARITY can be recognized by a **SMAT** with the **PEs** $\frac{i}{n}$ **and** $(-1)^i$ and **fixed precision**.

Proof

The SMAT receives $x_1 \cdots x_n$ with k ones.
Fix $L = 3$ and $d = 11$

Embeddings: Map $0 \mapsto (1, 0, \ldots, 0)$ and $1 \mapsto (0, 1, 0, \ldots, 0)$
So $y_i = (\mathbb{I}(x_i{=}0), \mathbb{I}(x_i{=}1), \frac{i}{n}, (-1)^i, \frac{1}{n} \ldots)$

Attention layer: Q, K and V are all 0 except $V_{6,2} = 1$
So $z_i = (\mathbb{I}(x_i = 0), \mathbb{I}(x_i = 1), \frac{i}{n}, (-1)^i, \frac{1}{n}, \frac{k}{n} \ldots)$

FFN: With $d' = 3$:
$g_i = \frac{1}{n}(\max(0, k - i - 1), \max(0, k - i), \max(0, k - i + 1))$

Using $\max(0, k - i - 1) - 2\max(0, k - i) + \max(0, k - i + 1) = \mathbb{I}(i = k)$
So $y_i^{(1)} = (\mathbb{I}(x_i = 0), \mathbb{I}(x_i = 1), \frac{i}{n}, (-1)^i, \frac{1}{n}, \frac{k}{n}, \frac{\mathbb{I}(i=k)}{n} \ldots)$
$\rightarrow$ Position 7 is not 0 only for $i = k$

Parity

Example

PARITY can be recognized by a **SMAT** with the **PEs** $\frac{i}{n}$ **and** $(-1)^i$ and **fixed precision**.

Proof

The SMAT receives $x_1 \cdots x_n$ with k ones.
Fix $L = 3$ and $d = 11$

Embeddings: Map $0 \mapsto (1, 0, \ldots, 0)$ and $1 \mapsto (0, 1, 0, \ldots, 0)$
So $y_i = (\mathbb{I}(x_i{=}0), \mathbb{I}(x_i{=}1), \frac{i}{n}, (-1)^i, \frac{1}{n} \ldots)$

Attention layer: Q, K and V are all 0 except $V_{6,2} = 1$
So $z_i = (\mathbb{I}(x_i = 0), \mathbb{I}(x_i = 1), \frac{i}{n}, (-1)^i, \frac{1}{n}, \frac{k}{n} \ldots)$

FFN: With $d' = 3$:
$g_i = \frac{1}{n}(\max(0, k - i - 1), \max(0, k - i), \max(0, k - i + 1))$

Using $\max(0, k - i - 1) - 2\max(0, k - i) + \max(0, k - i + 1) = \mathbb{I}(i = k)$
So $y_i^{(1)} = (\mathbb{I}(x_i = 0), \mathbb{I}(x_i = 1), \frac{i}{n}, (-1)^i, \frac{1}{n}, \frac{k}{n}, \frac{\mathbb{I}(i=k)}{n} \ldots)$
$\rightarrow$ Position 7 is not 0 only for $i = k$

Attention layer:

- $Q = (1, 1, \ldots)$
- $K = (0, 0, 0, -1, \ldots)$
- V is all 0 except $V_{8,9} = 1$
- Attention scores are much higher on the even positions
- $z_i = (0, \ldots, 0, e^{(-1)^k})$

Parity

Example

PARITY can be recognized by a **SMAT** with the **PEs** $\frac{i}{n}$ **and** $(-1)^i$ and **fixed precision**.

Proof

The SMAT receives $x_1 \cdots x_n$ with k ones.
Fix $L = 3$ and $d = 11$

Embeddings: Map $0 \mapsto (1, 0, ..., 0)$ and $1 \mapsto (0, 1, 0, ..., 0)$
So $y_i = (\mathbb{I}(x_i{=}0), \mathbb{I}(x_i{=}1), \frac{i}{n}, (-1)^i, \frac{1}{n} ...)$

Attention layer: Q, K and V are all 0 except $V_{6,2} = 1$
So $z_i = (\mathbb{I}(x_i = 0), \mathbb{I}(x_i = 1), \frac{i}{n}, (-1)^i, \frac{1}{n}, \frac{k}{n} ...)$

FFN: With $d' = 3$:
$g_i = \frac{1}{n}(\max(0, k - i - 1), \max(0, k - i), \max(0, k - i + 1))$

Using $\max(0, k - i - 1) - 2\max(0, k - i) + \max(0, k - i + 1) = \mathbb{I}(i = k)$
So $y_i^{(1)} = (\mathbb{I}(x_i = 0), \mathbb{I}(x_i = 1), \frac{i}{n}, (-1)^i, \frac{1}{n}, \frac{k}{n}, \frac{\mathbb{I}(i=k)}{n} ...)$
$\rightarrow$ Position 7 is not 0 only for $i = k$

Attention layer:

- $Q = (1, 1, ...)$
- $K = (0, 0, 0, -1, ...)$
- V is all 0 except $V_{8,9} = 1$
- Attention scores are much higher on the even positions
- $z_i = (0, ..., 0, e^{(-1)^k})$

FFN: trivial

Parity (2/2)

Example

PARITY can be recognized by a **SMAT** with the **PEs** $\frac{i}{n}$ **and** $(-1)^i$ and **fixed precision**.

Proof

The SMAT receives $x_1 \cdots x_n$ with k ones.
Fix $L = 3$ and $d = 11$

- $y_i = (\mathbb{I}(x_i{=}0), \mathbb{I}(x_i{=}1), \frac{i}{n}, (-1)^i, \frac{1}{n} \ldots)$
- $z_i = (0, \ldots, 0, e^{(-1)^k})$

Parity (2/2)

Example

PARITY can be recognized by a **SMAT** with the **PEs** $\frac{i}{n}$ **and** $(-1)^i$ and **fixed precision**.

Proof

The SMAT receives $x_1 \cdots x_n$ with k ones.
Fix $L = 3$ and $d = 11$

- $y_i = (\mathbb{I}(x_i{=}0), \mathbb{I}(x_i{=}1), \frac{i}{n}, (-1)^i, \frac{1}{n}, \ldots)$
- $z_i = (0, \ldots, 0, e^{(-1)^k})$

Attention layer:

- $Q = (1, 1, \ldots)$
- $K = (0, 0, 0, 1, \ldots)$
- V is all 0 except $V_{8,10} = -1$
- Attention scores are much higher on the odd positions
- $z_i = (0, \ldots, 0, -e^{(-1)^{k+1}})$

Parity (2/2)

Example

PARITY can be recognized by a **SMAT** with the **PEs** $\frac{i}{n}$ **and** $(-1)^i$ and **fixed precision**.

Proof

The SMAT receives $x_1 \cdots x_n$ with k ones.
Fix $L = 3$ and $d = 11$

- $y_i = (\mathbb{I}(x_i{=}0), \mathbb{I}(x_i{=}1), \frac{i}{n}, (-1)^i, \frac{1}{n} \ldots)$
- $z_i = (0, \ldots, 0, e^{(-1)^k})$

Attention layer:

- $Q = (1, 1, \ldots)$
- $K = (0, 0, 0, 1, \ldots)$
- V is all 0 except $V_{8,10} = -1$
- Attention scores are much higher on the odd positions
- $z_i = (0, \ldots, 0, -e^{(-1)^{k+1}})$

FFN: Adds the components 9 and 10

Parity (2/2)

Example

PARITY can be recognized by a **SMAT** with the **PEs** $\frac{i}{n}$ **and** $(-1)^i$ and **fixed precision**.

Proof

The SMAT receives $x_1 \cdots x_n$ with k ones.
Fix $L = 3$ and $d = 11$

- $y_i = (\mathbb{I}(x_i{=}0), \mathbb{I}(x_i{=}1), \frac{i}{n}, (-1)^i, \frac{1}{n} \ldots)$
- $z_i = (0, \ldots, 0, e^{(-1)^k})$

Attention layer:

- $Q = (1, 1, \ldots)$
- $K = (0, 0, 0, 1, \ldots)$
- V is all 0 except $V_{8,10} = -1$
- Attention scores are much higher on the odd positions
- $z_i = (0, \ldots, 0, -e^{(-1)^{k+1}})$

FFN: Adds the components 9 and 10

Output Adds the components 9 and 10 in 11
This gives $e^{(-1)^k} - e^{(-1)^{k+1}}$

Limitations of transformers

UHAT in $\mathbf{AC^0}$

We can leverage what we saw for circuits to gain knowledge about transformers.

UHAT in $\mathbf{AC^0}$

We can leverage what we saw for circuits to gain knowledge about transformers.

Theorem E.1.1

Every language recognized by a **unique hard-attention** transformer with **arbitrary PEs** and **arbitrary precision** is in $\mathbf{AC^0}$:

$$\mathbf{UHAT} \subseteq \mathbf{AC^0}.$$

UHAT in $\mathbf{AC^0}$

We can leverage what we saw for circuits to gain knowledge about transformers.

Theorem E.1.1

Every language recognized by a **unique hard-attention** transformer with **arbitrary PEs** and **arbitrary precision** is in $\mathbf{AC^0}$:

$$\mathbf{UHAT} \subseteq \mathbf{AC^0}.$$

In particular, **PARITY**, **MAJORITY** and **EQUALITY** are not recognizable by a **UHAT**.

UHAT in $\mathbf{AC^0}$

We can leverage what we saw for circuits to gain knowledge about transformers.

Theorem E.1.1

Every language recognized by a **unique hard-attention** transformer with **arbitrary PEs** and **arbitrary precision** is in $\mathbf{AC^0}$:

$\mathbf{UHAT} \subseteq \mathbf{AC^0}$.

In particular, **PARITY**, **MAJORITY** and **EQUALITY** are not recognizable by a **UHAT**.

Fix one transformer $\boldsymbol{T}$, with:

$\mathbf{we}, \mathbf{pe}, \boldsymbol{Q}^{(r)}, \boldsymbol{K}^{(r)}, \boldsymbol{V}^{(r)}, \boldsymbol{W_1}^{(r)}, \boldsymbol{W_2}^{(r)}, \boldsymbol{b_1}^{(r)}, \boldsymbol{b_2}^{(r)}, \boldsymbol{o}$.

The dimensions $\boldsymbol{d}, \boldsymbol{d'}$ and the number of layers $\boldsymbol{L}$ are constants.

UHAT in $\mathbf{AC^0}$

We can leverage what we saw for circuits to gain knowledge about transformers.

Theorem E.1.1

Every language recognized by a **unique hard-attention** transformer with **arbitrary PEs** and **arbitrary precision** is in $\mathbf{AC^0}$:

$\mathbf{UHAT} \subseteq \mathbf{AC^0}$.

In particular, **PARITY**, **MAJORITY** and **EQUALITY** are not recognizable by a **UHAT**.

Fix one transformer $\boldsymbol{T}$, with:

$\mathbf{we}, \mathbf{pe}, \boldsymbol{Q}^{(r)}, \boldsymbol{K}^{(r)}, \boldsymbol{V}^{(r)}, \boldsymbol{W_1}^{(r)}, \boldsymbol{W_2}^{(r)}, \boldsymbol{b_1}^{(r)}, \boldsymbol{b_2}^{(r)}, \boldsymbol{o}$.

The dimensions $\boldsymbol{d}, \boldsymbol{d'}$ and the number of layers $\boldsymbol{L}$ are constants.

For every input length $\boldsymbol{n}$, we build one circuit $\boldsymbol{C_n}$. The proof is **non-uniform**: all constants depending only on $\boldsymbol{T}$ and $\boldsymbol{n}$ may be hard-wired in $\boldsymbol{C_n}$.

UHAT in $\mathbf{AC^0}$

We can leverage what we saw for circuits to gain knowledge about transformers.

Theorem E.1.1

Every language recognized by a **unique hard-attention** transformer with **arbitrary PEs** and **arbitrary precision** is in $\mathbf{AC^0}$:

$\mathbf{UHAT} \subseteq \mathbf{AC^0}$.

In particular, **PARITY**, **MAJORITY** and **EQUALITY** are not recognizable by a **UHAT**.

Fix one transformer $\boldsymbol{T}$, with:

$\mathbf{we}, \mathbf{pe}, \boldsymbol{Q}^{(r)}, \boldsymbol{K}^{(r)}, \boldsymbol{V}^{(r)}, \boldsymbol{W}_1^{(r)}, \boldsymbol{W}_2^{(r)}, \boldsymbol{b}_1^{(r)}, \boldsymbol{b}_2^{(r)}, \boldsymbol{o}.$

The dimensions $\boldsymbol{d}, \boldsymbol{d'}$ and the number of layers $\boldsymbol{L}$ are constants.

For every input length $\boldsymbol{n}$, we build one circuit $\boldsymbol{C_n}$. The proof is **non-uniform**: all constants depending only on $\boldsymbol{T}$ and $\boldsymbol{n}$ may be hard-wired in $\boldsymbol{C_n}$.

Standard fact: every Boolean function on $\boldsymbol{O(\log n)}$ input bits can be implemented by an $\mathbf{AC^0}$ circuit.

Input encoding: first names

Fix n. We argue that every layer values in the UHAT can only take poly many values.
$\longrightarrow$ we abstract them away

Input encoding: first names

Fix n. We argue that every layer values in the UHAT can only take poly many values.
$\longrightarrow$ we abstract them away

A **name** at layer l is a string $\eta \in \{0,1\}^{B_l}$.
Its meaning is given by the map

$$\tau_l : \{0,1\}^{B_l} \longrightarrow \mathbb{R}^d.$$

Input encoding: first names

Fix n. We argue that every layer values in the UHAT can only take poly many values.
$\longrightarrow$ we abstract them away

A **name** at layer l is a string $\eta \in \{0,1\}^{B_l}$.
Its meaning is given by the map

$$\tau_l : \{0,1\}^{B_l} \longrightarrow \mathbb{R}^d.$$

The invariant of the simulation is

$$\tau_l(\eta_i^{(l)}) = y_i^{(l)}.$$

The circuit computes the name $\eta_i^{(r)}$, τ_r tells us what this name denotes.
$\longrightarrow$ If B_l is of size logarithmic, we can retrieve this value

Input encoding: first names

Fix n. We argue that every layer values in the UHAT can only take poly many values.
$\longrightarrow$ we abstract them away

A **name** at layer l is a string $\eta \in \{0,1\}^{B_l}$.
Its meaning is given by the map

$$\tau_l : \{0,1\}^{B_l} \longrightarrow \mathbb{R}^d.$$

The invariant of the simulation is

$$\tau_l(\eta_i^{(l)}) = y_i^{(l)}.$$

The circuit computes the name $\eta_i^{(r)}$, τ_r tells us what this name denotes.
$\longrightarrow$ If B_l is of size logarithmic, we can retrieve this value

For $r = 0$, choose binary encodings

$$\mathbf{enc}_\Sigma : \Sigma \longrightarrow \{0,1\}^s,$$

$$\mathbf{enc}_n : \{1, \ldots, n\} \longrightarrow \{0,1\}^\lambda,$$

where s is constant and $\lambda = \lceil \log_2(n+1) \rceil$.

Input encoding: first names

Fix n. We argue that every layer values in the UHAT can only take poly many values.
$\rightarrow$ we abstract them away

A **name** at layer l is a string $\eta \in \{0,1\}^{B_l}$.
Its meaning is given by the map

$$\tau_l : \{0,1\}^{B_l} \to \mathbb{R}^d.$$

The invariant of the simulation is

$$\tau_l(\eta_i^{(l)}) = y_i^{(l)}.$$

The circuit computes the name $\eta_i^{(r)}$, τ_r tells us what this name denotes.
$\rightarrow$ If B_l is of size logarithmic, we can retrieve this value

For $r = 0$, choose binary encodings

$$\mathbf{enc}_\Sigma : \Sigma \to \{0,1\}^s,$$

$$\mathbf{enc}_n : \{1,\dots,n\} \to \{0,1\}^\lambda,$$

where s is constant and $\lambda = \lceil \log_2(n+1) \rceil$.

The name of position i before the first layer is

$$\eta_i^{(0)} = \mathbf{enc}_\Sigma(x_i) \parallel \mathbf{enc}_n(i)$$

and $B_0 = s + \lambda = O(\log n)$.
Every bit can be computed by a constant size circuit.

Input encoding: first names

Fix n. We argue that every layer values in the UHAT can only take poly many values.
$\rightarrow$ we abstract them away

A **name** at layer l is a string $\eta \in \{0,1\}^{B_l}$.
Its meaning is given by the map

$$\tau_l : \{0,1\}^{B_l} \to \mathbb{R}^d.$$

The invariant of the simulation is

$$\tau_l(\eta_i^{(l)}) = y_i^{(l)}.$$

The circuit computes the name $\eta_i^{(r)}$, τ_r tells us what this name denotes.
$\rightarrow$ If B_l is of size logarithmic, we can retrieve this value

For $r = 0$, choose binary encodings

$$\mathbf{enc}_\Sigma : \Sigma \to \{0,1\}^s,$$

$$\mathbf{enc}_n : \{1,\dots,n\} \to \{0,1\}^\lambda,$$

where s is constant and $\lambda = \lceil \log_2(n+1) \rceil$.

The name of position i before the first layer is

$$\eta_i^{(0)} = \mathbf{enc}_\Sigma(x_i) \,\|\, \mathbf{enc}_n(i)$$

and $B_0 = s + \lambda = O(\log n)$.
Every bit can be computed by a constant size circuit.

Define τ_0 on valid names by

$$\tau_0\big(\mathbf{enc}_\Sigma(a) \,\|\, \mathbf{enc}_n(i)\big) = \mathbf{we}(a) + \mathbf{pe}(i),$$

and arbitrarily on invalid bit strings. Therefore

$$\tau_0(\eta_i^{(0)}) = \mathbf{we}(x_i) + \mathbf{pe}(i) = y_i^{(0)},$$

proving the base case of the invariant.

Q, K: scores become comparisons

Assume by induction that the circuit has names $\eta_i^{(l-1)}$ and that $\tau_{l-1}(\eta_i^{(l-1)}) = y_i^{(l-1)}$.

The attention scores only depends on the names:

$$s^{(l)}(\eta, \eta') = \big(Q^{(l)}\tau_{l-1}(\eta)\big)^\top \big(K^{(l)}\tau_{l-1}(\eta')\big).$$

Q, K: scores become comparisons

Assume by induction that the circuit has names $\eta_i^{(l-1)}$ and that $\tau_{l-1}(\eta_i^{(l-1)}) = y_i^{(l-1)}$.

The attention scores only depends on the names:

$$s^{(l)}(\eta, \eta') = \big(Q^{(l)}\tau_{l-1}(\eta)\big)^\top \big(K^{(l)}\tau_{l-1}(\eta')\big).$$

For fixed output position i, candidate j beats candidate j' iff $s^{(l)}(\eta_i^{(l-1)}, \eta_j^{(l-1)}) \geq s^{(l)}(\eta_i^{(l-1)}, \eta_{j'}^{(l-1)})$.

Q, K: scores become comparisons

Assume by induction that the circuit has names $\eta_i^{(l-1)}$ and that $\tau_{l-1}(\eta_i^{(l-1)}) = y_i^{(l-1)}$.

The attention scores only depends on the names:

$$s^{(l)}(\eta, \eta') = \big(Q^{(l)}\tau_{l-1}(\eta)\big)^\top \big(K^{(l)}\tau_{l-1}(\eta')\big).$$

For fixed output position i, candidate j beats candidate j' iff

$$s^{(l)}(\eta_i^{(l-1)}, \eta_j^{(l-1)}) \geq s^{(l)}(\eta_i^{(l-1)}, \eta_{j'}^{(l-1)}).$$

Define the Boolean predicate

$$\mathbf{GE}_r(\eta, \eta', \eta'') = \mathbb{I}\big(s^{(l)}(\eta, \eta') \geq s^{(l)}(\eta, \eta'')\big).$$

Q, K: scores become comparisons

Assume by induction that the circuit has names $\eta_i^{(l-1)}$ and that $\tau_{l-1}(\eta_i^{(l-1)}) = y_i^{(l-1)}$.
The attention scores only depends on the names:

$$s^{(l)}(\eta, \eta') = \big(Q^{(l)}\tau_{l-1}(\eta)\big)^\top \big(K^{(l)}\tau_{l-1}(\eta')\big).$$

For fixed output position i, candidate j beats candidate j' iff

$$s^{(l)}(\eta_i^{(l-1)}, \eta_j^{(l-1)}) \geq s^{(l)}(\eta_i^{(l-1)}, \eta_{j'}^{(l-1)}).$$

Define the Boolean predicate

$$\mathbf{GE}_r(\eta, \eta', \eta'') = \mathbb{I}\big(s^{(l)}(\eta, \eta') \geq s^{(l)}(\eta, \eta'')\big).$$

Its input is of size log, hence it can be computed in $\mathbf{AC}^0$.

Q, K: scores become comparisons

Assume by induction that the circuit has names $\eta_i^{(l-1)}$ and that $\tau_{l-1}(\eta_i^{(l-1)}) = y_i^{(l-1)}$.
The attention scores only depends on the names:

$$s^{(l)}(\eta, \eta') = \big(Q^{(l)}\tau_{l-1}(\eta)\big)^\top \big(K^{(l)}\tau_{l-1}(\eta')\big).$$

For fixed output position i, candidate j beats candidate j' iff

$$s^{(l)}(\eta_i^{(l-1)}, \eta_j^{(l-1)}) \geq s^{(l)}(\eta_i^{(l-1)}, \eta_{j'}^{(l-1)}).$$

Define the Boolean predicate

$$\mathbf{GE}_r(\eta, \eta', \eta'') = \mathbb{I}\big(s^{(l)}(\eta, \eta') \geq s^{(l)}(\eta, \eta'')\big).$$

Its input is of size log, hence it can be computed in $\mathbf{AC}^0$.

This is enough information to compute the attention weights.

Unique hard attention becomes a selector

For a fixed layer l and output position i, compute for all candidates j

$$m_{i,j} = \bigwedge_{j'=1}^{n} \mathbf{GE}_l(\eta_i^{(l-1)}, \eta_j^{(l-1)}, \eta_{j'}^{(l-1)}).$$

So $m_{i,j} = 1$ says that j is a maximizer of the score.

Unique hard attention becomes a selector

For a fixed layer l and output position i, compute for all candidates j

$$m_{i,j} = \bigwedge_{j'=1}^{n} \mathbf{GE}_l(\eta_i^{(l-1)}, \eta_j^{(l-1)}, \eta_{j'}^{(l-1)}).$$

So $m_{i,j} = 1$ says that j is a maximizer of the score.

Unique hard attention takes the **leftmost** maximizer. Encoded by:

$$a_{i,j} = m_{i,j} \wedge \bigwedge_{j'<j} \neg m_{i,j'}.$$

Exactly one $a_{i,j}$ is true.

Unique hard attention becomes a selector

For a fixed layer l and output position i, compute for all candidates j

$$m_{i,j} = \bigwedge_{j'=1}^{n} \mathbf{GE}_l(\eta_i^{(l-1)}, \eta_j^{(l-1)}, \eta_{j'}^{(l-1)}).$$

So $m_{i,j} = 1$ says that j is a maximizer of the score.

Unique hard attention takes the **leftmost** maximizer. Encoded by:

$$a_{i,j} = m_{i,j} \wedge \bigwedge_{j'<j} \neg m_{i,j'}.$$

Exactly one $a_{i,j}$ is true.

Let j^* be the selected position. Copy every bit t of the selected name:

$$\sigma_i^{(r)}[t] = \bigvee_{j=1}^{n} \left(a_{i,j} \wedge \eta_j^{(r-1)}[t]\right).$$

Unique hard attention becomes a selector

For a fixed layer l and output position i, compute for all candidates j

$$m_{i,j} = \bigwedge_{j'=1}^{n} \mathbf{GE}_l(\eta_i^{(l-1)}, \eta_j^{(l-1)}, \eta_{j'}^{(l-1)}).$$

So $m_{i,j} = 1$ says that j is a maximizer of the score.

Unique hard attention takes the **leftmost** maximizer. Encoded by:

$$a_{i,j} = m_{i,j} \wedge \bigwedge_{j'<j} \neg m_{i,j'}.$$

Exactly one $a_{i,j}$ is true.

Let j^* be the selected position. Copy every bit t of the selected name:

$$\sigma_i^{(r)}[t] = \bigvee_{j=1}^{n} \left(a_{i,j} \wedge \eta_j^{(r-1)}[t]\right).$$

Therefore

$$\sigma_i^{(r)} = \eta_{j^*}^{(r-1)} \qquad \text{where} j^* = \min \arg\max_j \quad (q_i^{(r)})^\top k_j^{(r)}.$$

This is exactly the unique hard-attention choice.

V and FFN update the name

In the transformer, after selecting j^*,

$$v_{j^*}^{(l)} = V^{(l)} y_{j^*}^{(l-1)}, \qquad z_i^{(l)} = v_{j^*}^{(l)} + y_i^{(l-1)}.$$

V and FFN update the name

In the transformer, after selecting j^*,

$$v_{j^*}^{(l)} = V^{(l)} y_{j^*}^{(l-1)}, \qquad z_i^{(l)} = v_{j^*}^{(l)} + y_i^{(l-1)}.$$

The circuit records the two old names

$$\eta_i^{(l)} = \left(\eta_i^{(l-1)}, \sigma_i^{(l)}\right).$$

Since $\sigma_i^{(l)} = \eta_{j^*}^{(l-1)}$, this name determines

$$z_i^{(l)} = V^{(l)} \tau_{l-1}(\sigma_i^{(l)}) + \tau_{l-1}(\eta_i^{(l-1)}).$$

V and FFN update the name

In the transformer, after selecting j^*,

$$v_{j^*}^{(l)} = V^{(l)} y_{j^*}^{(l-1)}, \qquad z_i^{(l)} = v_{j^*}^{(l)} + y_i^{(l-1)}.$$

The circuit records the two old names

$$\eta_i^{(l)} = \left(\eta_i^{(l-1)}, \sigma_i^{(l)}\right).$$

Since $\sigma_i^{(l)} = \eta_{j^*}^{(l-1)}$, this name determines

$$z_i^{(l)} = V^{(l)} \tau_{l-1}(\sigma_i^{(l)}) + \tau_{l-1}(\eta_i^{(l-1)}).$$

The FFN part is

$$h_i^{(l)} = W_1^{(l)} z_i^{(l)} + b_1^{(l)}, \qquad g_i^{(l)} = \mathbf{ReLU}(h_i^{(l)}),$$
$$y_i^{(l)} = W_2^{(l)} g_i^{(l)} + b_2^{(l)} + z_i^{(l)}.$$

V and FFN update the name

In the transformer, after selecting j^*,

$$v_{j^*}^{(l)} = V^{(l)} y_{j^*}^{(l-1)}, \qquad z_i^{(l)} = v_{j^*}^{(l)} + y_i^{(l-1)}.$$

The circuit records the two old names

$$\eta_i^{(l)} = \left(\eta_i^{(l-1)}, \sigma_i^{(l)}\right).$$

Since $\sigma_i^{(l)} = \eta_{j^*}^{(l-1)}$, this name determines

$$z_i^{(l)} = V^{(l)} \tau_{l-1}(\sigma_i^{(l)}) + \tau_{l-1}(\eta_i^{(l-1)}).$$

The FFN part is

$$h_i^{(l)} = W_1^{(l)} z_i^{(l)} + b_1^{(l)}, \qquad g_i^{(l)} = \mathbf{ReLU}(h_i^{(l)}),$$
$$y_i^{(l)} = W_2^{(l)} g_i^{(l)} + b_2^{(l)} + z_i^{(l)}.$$

Define τ_l by the formula above. Then the new name satisfies

$$\tau_l(\eta_i^{(l)}) = y_i^{(l)}.$$

Thus $V^{(l)}$, $W_1^{(l)}, W_2^{(l)}$, $b_1^{(l)}, b_2^{(l)}$, and $\mathbf{ReLU}$ are all represented in τ_l.

V and FFN update the name

In the transformer, after selecting j^*,

$$v_{j^*}^{(l)} = V^{(l)} y_{j^*}^{(l-1)}, \qquad z_i^{(l)} = v_{j^*}^{(l)} + y_i^{(l-1)}.$$

The circuit records the two old names

$$\eta_i^{(l)} = \left(\eta_i^{(l-1)}, \sigma_i^{(l)}\right).$$

Since $\sigma_i^{(l)} = \eta_{j^*}^{(l-1)}$, this name determines

$$z_i^{(l)} = V^{(l)} \tau_{l-1}(\sigma_i^{(l)}) + \tau_{l-1}(\eta_i^{(l-1)}).$$

The FFN part is

$$h_i^{(l)} = W_1^{(l)} z_i^{(l)} + b_1^{(l)}, \qquad g_i^{(l)} = \mathbf{ReLU}(h_i^{(l)}),$$
$$y_i^{(l)} = W_2^{(l)} g_i^{(l)} + b_2^{(l)} + z_i^{(l)}.$$

Define τ_l by the formula above. Then the new name satisfies

$$\tau_l(\eta_i^{(l)}) = y_i^{(l)}.$$

Thus $V^{(l)}$, $W_1^{(l)}, W_2^{(l)}$, $b_1^{(l)}, b_2^{(l)}$, and $\mathbf{ReLU}$ are all represented in τ_l.

The name length is doubled at each layer:
$\longrightarrow$ always $O(\log(n))$

Output and conclusion

The transformer accepts by testing the last position:

$$o^\top y_n^{(L)} > 0.$$

By the invariant, the circuit has the name $\eta_n^{(L)}$, and

$$\tau_L(\eta_n^{(L)}) = y_n^{(L)}.$$

Output and conclusion

The transformer accepts by testing the last position:

$$o^\top y_n^{(L)} > 0.$$

By the invariant, the circuit has the name $\eta_n^{(L)}$, and

$$\tau_L(\eta_n^{(L)}) = y_n^{(L)}.$$

Define the final Boolean predicate

$$\mathbf{Out}(\eta) = \mathbb{I}\big(o^\top \tau_L(\eta) > 0\big).$$

It is a function of one $O(\log n)$-bit name, hence has an $\mathbf{AC^0}$ circuit.

Output and conclusion

The transformer accepts by testing the last position:

$$o^\top y_n^{(L)} > 0.$$

By the invariant, the circuit has the name $\eta_n^{(L)}$, and

$$\tau_L(\eta_n^{(L)}) = y_n^{(L)}.$$

Define the final Boolean predicate

$$\mathbf{Out}(\eta) = \mathbb{I}\big(o^\top \tau_L(\eta) > 0\big).$$

It is a function of one $O(\log n)$-bit name, hence has an $\mathbf{AC^0}$ circuit.

Therefore, for every n, the circuit C_n satisfies

$$C_n(x_1 \cdots x_n) = 1 \iff T \text{ accepts } x_1 \cdots x_n.$$

Output and conclusion

The transformer accepts by testing the last position:

$$o^\top y_n^{(L)} > 0.$$

By the invariant, the circuit has the name $\eta_n^{(L)}$, and

$$\tau_L(\eta_n^{(L)}) = y_n^{(L)}.$$

Define the final Boolean predicate

$$\mathbf{Out}(\eta) = \mathbb{I}\big(o^\top \tau_L(\eta) > 0\big).$$

It is a function of one $O(\log n)$-bit name, hence has an $\mathbf{AC^0}$ circuit.

Therefore, for every n, the circuit C_n satisfies

$$C_n(x_1 \cdots x_n) = 1 \iff T \text{ accepts } x_1 \cdots x_n.$$

Theorem E.1.2

Every language recognized by a **unique hard-attention** transformer with **arbitrary PEs** and **arbitrary precision** is in $\mathbf{AC^0}$:

$$\mathbf{UHAT} \subseteq \mathbf{AC^0}.$$

Other limitations

Definition

The class $\mathbf{TC^0}$ is the set of language that can be computed with a family of circuits of **polynomial size** and **constant depth** thta have acces to **MAJORITY gates**.

Other limitations

Definition

The class $\mathbf{TC^0}$ is the set of language that can be computed with a family of circuits of **polynomial size** and **constant depth** thta have acces to **MAJORITY gates**.

Fact E.1.3

We have $\mathbf{ACC^0} \subseteq \mathbf{TC^0}$.

Other limitations

Definition

The class $\mathbf{TC^0}$ is the set of language that can be computed with a family of circuits of **polynomial size** and **constant depth** thta have acces to **MAJORITY gates**.

Fact E.1.3

We have $\mathbf{ACC^0} \subseteq \mathbf{TC^0}$.

We can study **SMAT** with a proof similar to the one for **UHAT**:

Other limitations

Definition

The class $\mathbf{TC^0}$ is the set of language that can be computed with a family of circuits of **polynomial size** and **constant depth** thta have acces to **MAJORITY gates**.

Fact E.1.3

We have $\mathbf{ACC^0} \subseteq \mathbf{TC^0}$.

We can study **SMAT** with a proof similar to the one for **UHAT**:

Theorem E.1.4

Every language recognized by a **softmax attention** transformer with **arbitrary PEs** and **log precision** is in $\mathbf{TC^0}$:

$\mathbf{SMAT} \subseteq \mathbf{TC^0}$.

Other limitations

Definition

The class $\mathbf{TC^0}$ is the set of language that can be computed with a family of circuits of **polynomial size** and **constant depth** thta have acces to **MAJORITY gates**.

Fact E.1.3

We have $\mathbf{ACC^0} \subseteq \mathbf{TC^0}$.

We can study **SMAT** with a proof similar to the one for **UHAT**:

Theorem E.1.4

Every language recognized by a **softmax attention** transformer with **arbitrary PEs** and **log precision** is in $\mathbf{TC^0}$:

$\mathbf{SMAT} \subseteq \mathbf{TC^0}$.

Moreover, severe restrictions of the model only recognizes regular languages:

Other limitations

Definition

The class $\mathbf{TC^0}$ is the set of language that can be computed with a family of circuits of **polynomial size** and **constant depth** thta have acces to **MAJORITY gates**.

Fact E.1.3

We have $\mathbf{ACC^0} \subseteq \mathbf{TC^0}$.

We can study **SMAT** with a proof similar to the one for **UHAT**:

Theorem E.1.4

Every language recognized by a **softmax attention** transformer with **arbitrary PEs** and **log precision** is in $\mathbf{TC^0}$:

$\mathbf{SMAT} \subseteq \mathbf{TC^0}$.

Moreover, severe restrictions of the model only recognizes regular languages:

Theorem E.1.5

The language recognized by a **masked unique-hard attention** transformer with **no PEs** and **finite precision** are precisely the **star-free languages**.

Turing-completeness

Chain-of-thoughts

In practice, transformers are not simply recognizers, but generates an **output** one token at a time

Chain-of-thoughts

In practice, transformers are not simply recognizers, but generates an **output** one token at a time

To enhance the performances, it is empirically better to split a query in several prompts

Chain-of-thoughts

In practice, transformers are not simply recognizers, but generates an **output** one token at a time

To enhance the performances, it is empirically better to split a query in several prompts

$\longrightarrow$ we enhance transformers with the ability to generate several outputs before answering

Chain-of-thoughts

In practice, transformers are not simply recognizers, but generates an **output** one token at a time

To enhance the performances, it is empirically better to split a query in several prompts

$\longrightarrow$ we enhance transformers with the ability to generate several outputs before answering

We now allow a **Chain-of-thought**:

$\gamma_0, \gamma_1, \gamma_2, \cdots$

At each step, the same fixed transformer is applied to the current prefix and outputs the next thought as a letter.

Chain-of-thoughts

In practice, transformers are not simply recognizers, but generates an **output** one token at a time

To enhance the performances, it is empirically better to split a query in several prompts

$\longrightarrow$ we enhance transformers with the ability to generate several outputs before answering

We now allow a **Chain-of-thought**:

$\gamma_0, \gamma_1, \gamma_2, \cdots$

At each step, the same fixed transformer is applied to the current prefix and outputs the next thought as a letter.

How to generate a letter?
Let d'' be a dimension for the thoughts, and $o \in \mathbb{R}^{d'' \times d}$. Return $o^\top y_n^{(L)}$ instead of checking if it is > 0.

Chain-of-thoughts

In practice, transformers are not simply recognizers, but generates an **output** one token at a time

To enhance the performances, it is empirically better to split a query in several prompts

$\longrightarrow$ we enhance transformers with the ability to generate several outputs before answering

We now allow a **Chain-of-thought**:

$$\gamma_0, \gamma_1, \gamma_2, \cdots$$

At each step, the same fixed transformer is applied to the current prefix and outputs the next thought as a letter.

How to generate a letter?
Let d'' be a dimension for the thoughts, and $o \in \mathbb{R}^{d'' \times d}$. Return $o^\top y_n^{(L)}$ instead of checking if it is > 0.

CoT

Let T be a transformer that generates a thought. For an input x and a fresh symbol $\#$, define recursively:

$$\mathbf{CoT}_t(x) = x \# \gamma_1 \cdots \gamma_t$$
$$\gamma_{t+1} = T(\mathbf{CoT}_t(x)).$$

The infinite word

$$\gamma_1 \gamma_2 \gamma_3 \cdots$$

is the **Chain-of-thought generated by T on x**.

Chain-of-thoughts

In practice, transformers are not simply recognizers, but generates an **output** one token at a time

To enhance the performances, it is empirically better to split a query in several prompts

$\longrightarrow$ we enhance transformers with the ability to generate several outputs before answering

We now allow a **Chain-of-thought**:

$$\gamma_0, \gamma_1, \gamma_2, \cdots$$

At each step, the same fixed transformer is applied to the current prefix and outputs the next thought as a letter.

How to generate a letter?
Let d'' be a dimension for the thoughts, and $o \in \mathbb{R}^{d'' \times d}$. Return $o^\top y_n^{(L)}$ instead of checking if it is > 0.

CoT

Let T be a transformer that generates a thought. For an input x and a fresh symbol $\#$, define recursively:

$$\mathbf{CoT}_t(x) = x \# \gamma_1 \cdots \gamma_t$$
$$\gamma_{t+1} = T(\mathbf{CoT}_t(x)).$$

The infinite word

$$\gamma_1 \gamma_2 \gamma_3 \cdots$$

is the **Chain-of-thought generated by T on x**.

We assume that the word embeddings for thoughts are linear functions.

Chain-of-thoughts

In practice, transformers are not simply recognizers, but generates an **output** one token at a time

To enhance the performances, it is empirically better to split a query in several prompts

$\longrightarrow$ we enhance transformers with the ability to generate several outputs before answering

We now allow a **Chain-of-thought**:

$$\gamma_0, \gamma_1, \gamma_2, \cdots$$

At each step, the same fixed transformer is applied to the current prefix and outputs the next thought as a letter.

How to generate a letter?
Let d'' be a dimension for the thoughts, and $o \in \mathbb{R}^{d'' \times d}$. Return $o^\top y_n^{(L)}$ instead of checking if it is > 0.

CoT

Let T be a transformer that generates a thought. For an input x and a fresh symbol $\#$, define recursively:

$$\mathbf{CoT}_t(x) = x \# \gamma_1 \cdots \gamma_t$$
$$\gamma_{t+1} = T(\mathbf{CoT}_t(x)).$$

The infinite word

$$\gamma_1 \gamma_2 \gamma_3 \cdots$$

is the **Chain-of-thought generated by T on x**.

We assume that the word embeddings for thoughts are linear functions.

We say that x is **accepted by T with CoT** if the thought $(0, \cdots, 0)$ is generated at some point. It is rejected if $(0, \cdots, 1)$ is generated.

Turing-completeness

On input $x_1 \cdots x_n \# \gamma_1 \cdots \gamma_t$, we will use the positional encodings among:

$$i, i^2, \frac{1}{i+1}, n, \mathbb{I}[i < n]$$

Turing-completeness

On input $x_1 \cdots x_n \# \gamma_1 \cdots \gamma_t$, we will use the positional encodings among:

$$i, i^2, \frac{1}{i+1}, n, \mathbb{I}[i < n]$$

With that, relatively weak transformers are Turing-complete.

Theorem E.1.6

Let L be a language computed by a Turing machine M. Then there is a **UHAT** T with **CoT**, **log precision** and the **preceeding PEs** that accepts L.

Turing-completeness

On input $x_1 \cdots x_n \# \gamma_1 \cdots \gamma_t$, we will use the positional encodings among:

$$i, i^2, \frac{1}{i+1}, n, \mathbb{I}[i < n]$$

With that, relatively weak transformers are Turing-complete.

Theorem E.1.6

Let L be a language computed by a Turing machine M. Then there is a **UHAT** T with **CoT**, **log precision** and the **preceeding PEs** that accepts L.

If M stops after $\tau(|x|)$ steps, then the generated Chain-of-thought has size $O(\tau(|x|))$.

Turing-completeness

On input $x_1 \cdots x_n \# \gamma_1 \cdots \gamma_t$, we will use the positional encodings among:

$$i, i^2, \frac{1}{i+1}, n, \mathbb{I}[i < n]$$

With that, relatively weak transformers are Turing-complete.

Theorem E.1.6

Let L be a language computed by a Turing machine M. Then there is a **UHAT** T with **CoT**, **log precision** and the **preceeding PEs** that accepts L.

If M stops after $\tau(|x|)$ steps, then the generated Chain-of-thought has size $O(\tau(|x|))$.

In the following, the TM $M = (\Gamma, Q, \delta, s_0)$ has a single tape and performs one action at a time: the transition function is

$$\delta : \Sigma \times Q \to A \times Q.$$

where

$$A = \{\leftarrow, \downarrow, \rightarrow\} \cup \{\mathbf{WRITE}(\sigma) : \sigma \in \Gamma\}.$$

Generating the first thought

The thoughts will be of the form, with $d'' = 1 + |A| + |Q|$,

$$\gamma_t = (i_t, a_t, q_t).$$

where after t simulated actions:

$i_t =$ current head position,

$a_t =$ action just performed in one-hot,

$q_t =$ current state in one-hot.

Generating the first thought

The thoughts will be of the form, with $d'' = 1 + |A| + |Q|$,

$$\gamma_t = (i_t, a_t, q_t).$$

where after t simulated actions:

$i_t =$ current head position,

$a_t =$ action just performed in one-hot,

$q_t =$ current state in one-hot.

The first context is just

$$\mathbf{CoT}_0(x) = x_1 \cdots x_n \#.$$

The first thought is:

$$\gamma_1 = (0, \downarrow, s_0).$$

Generating the first thought

The thoughts will be of the form, with $d'' = 1 + |A| + |Q|$,

$$\gamma_t = (i_t, a_t, q_t).$$

where after t simulated actions:

$i_t =$ current head position,

$a_t =$ action just performed in one-hot,

$q_t =$ current state in one-hot.

The first context is just

$$\mathbf{CoT_0}(x) = x_1 \cdots x_n \#.$$

The first thought is:

$$\gamma_1 = (0, \downarrow, s_0).$$

How does the transformer know this is the first call?
The output is read at the last position of the current context. The last letter is # only for $\mathbf{CoT_0}(x)$,

Generating the first thought

The thoughts will be of the form, with $d'' = 1 + |A| + |Q|$,

$$\gamma_t = (i_t, a_t, q_t).$$

where after t simulated actions:

$i_t =$ current head position,

$a_t =$ action just performed in one-hot,

$q_t =$ current state in one-hot.

The first context is just

$$\mathbf{CoT_0}(x) = x_1 \cdots x_n \#.$$

The first thought is:

$$\gamma_1 = (0, \downarrow, s_0).$$

How does the transformer know this is the first call?
The output is read at the last position of the current context. The last letter is # only for $\mathbf{CoT_0}(x)$,

Give the separator a special embedding coordinate:

$$\mathbf{we}(\#)[\mathbf{sep}] = 1,$$

$$\mathbf{we}(a)[\mathbf{sep}] = 0 \text{ for } a \neq \#.$$

At the last position, the feed-forward part can test this coordinate.

Generating the first thought

The thoughts will be of the form, with $d'' = 1 + |A| + |Q|$,

$$\gamma_t = (i_t, a_t, q_t).$$

where after t simulated actions:

i_t = current head position,

a_t = action just performed in one-hot,

q_t = current state in one-hot.

The first context is just

$$\mathbf{CoT_0}(x) = x_1 \cdots x_n \#.$$

The first thought is:

$$\gamma_1 = (0, \downarrow, s_0).$$

How does the transformer know this is the first call?
The output is read at the last position of the current context. The last letter is # only for $\mathbf{CoT_0}(x)$,

Give the separator a special embedding coordinate:

$$\mathbf{we}(\#)[\mathbf{sep}] = 1,$$

$$\mathbf{we}(a)[\mathbf{sep}] = 0 \text{ for } a \neq \#.$$

At the last position, the feed-forward part can test this coordinate.

A FFN can test whether this coordinate is 1.
Let $u = y_m^{(L)}[\mathbf{sep}]$.
Since in our construction $u \in \{0, 1\}$, the gate

$$g_{\mathbf{init}} = \mathbf{ReLU}(u - \tfrac{1}{2}) - \mathbf{ReLU}(u - \tfrac{3}{2})$$

satisfies

$$g_{\mathbf{init}} = \begin{cases} \frac{1}{2} & \text{if the last letter is } \#, \\ 0 & \text{otherwise.} \end{cases}$$

Multiplying the constant vector encoding $(0, \downarrow, s_0)$ by $2g_{\mathbf{init}}$ forces the first output to be

$$T(x_1 \cdots x_n \#) = (0, \downarrow, s_0).$$

Computation of next thoughts?

Suppose the latest thought is

$$\xi_t = (i_t, a_t, q_t).$$

To produce ξ_{t+1}, the transformer must know the current tape symbol at cell i_t. Call it σ_t.

Computation of next thoughts?

Suppose the latest thought is

$$\xi_t = (i_t, a_t, q_t).$$

To produce ξ_{t+1}, the transformer must know the current tape symbol at cell i_t. Call it σ_t.

Then it applies the finite transition table

$$\delta(\sigma_t, q_t) = (a_{t+1}, q_{t+1}).$$

Computation of next thoughts?

Suppose the latest thought is

$$\xi_t = (i_t, a_t, q_t).$$

To produce ξ_{t+1}, the transformer must know the current tape symbol at cell i_t. Call it σ_t.

Then it applies the finite transition table

$$\delta(\sigma_t, q_t) = (a_{t+1}, q_{t+1}).$$

The next head position is

$$i_{t+1} = \begin{cases} i_t + 1 & \text{if } a_{t+1} = \rightarrow, \\ i_t - 1 & \text{if } a_{t+1} = \leftarrow, \\ i_t & \text{if } a_{t+1} = \mathbf{WRITE}(\sigma). \end{cases}$$

Computation of next thoughts?

Suppose the latest thought is

$$\xi_t = (i_t, a_t, q_t).$$

To produce ξ_{t+1}, the transformer must know the current tape symbol at cell i_t. Call it σ_t.

Then it applies the finite transition table

$$\delta(\sigma_t, q_t) = (a_{t+1}, q_{t+1}).$$

The next head position is

$$i_{t+1} = \begin{cases} i_t + 1 & \text{if } a_{t+1} = \rightarrow, \\ i_t - 1 & \text{if } a_{t+1} = \leftarrow, \\ i_t & \text{if } a_{t+1} = \mathbf{WRITE}(\sigma). \end{cases}$$

Thus the only real memory problem is: what symbol is currently stored at tape cell i_t?

Computation of next thoughts?

Suppose the latest thought is

$$\xi_t = (i_t, a_t, q_t).$$

To produce ξ_{t+1}, the transformer must know the current tape symbol at cell i_t. Call it σ_t.

Then it applies the finite transition table

$$\delta(\sigma_t, q_t) = (a_{t+1}, q_{t+1}).$$

The next head position is

$$i_{t+1} = \begin{cases} i_t + 1 & \text{if } a_{t+1} = \rightarrow, \\ i_t - 1 & \text{if } a_{t+1} = \leftarrow, \\ i_t & \text{if } a_{t+1} = \mathbf{WRITE}(\sigma). \end{cases}$$

Thus the only real memory problem is: what symbol is currently stored at tape cell i_t?

A tape cell changes only when the machine performs a write action there.

Computation of next thoughts?

Suppose the latest thought is

$$\xi_t = (i_t, a_t, q_t).$$

To produce ξ_{t+1}, the transformer must know the current tape symbol at cell i_t. Call it σ_t.

Then it applies the finite transition table

$$\delta(\sigma_t, q_t) = (a_{t+1}, q_{t+1}).$$

The next head position is

$$i_{t+1} = \begin{cases} i_t + 1 & \text{if } a_{t+1} = \rightarrow, \\ i_t - 1 & \text{if } a_{t+1} = \leftarrow, \\ i_t & \text{if } a_{t+1} = \mathbf{WRITE}(\sigma). \end{cases}$$

Thus the only real memory problem is: what symbol is currently stored at tape cell i_t?

A tape cell changes only when the machine performs a write action there.

Define the latest previous write to the current cell:

$$\ell_t = \max\{s \leq t : i_s = i_t \text{ and } a_s = \mathbf{WRITE}(\sigma) \text{ for some } \sigma\}.$$

Computation of next thoughts?

Suppose the latest thought is

$$\xi_t = (i_t, a_t, q_t).$$

To produce ξ_{t+1}, the transformer must know the current tape symbol at cell i_t. Call it σ_t.

Then it applies the finite transition table

$$\delta(\sigma_t, q_t) = (a_{t+1}, q_{t+1}).$$

The next head position is

$$i_{t+1} = \begin{cases} i_t + 1 & \text{if } a_{t+1} = \rightarrow, \\ i_t - 1 & \text{if } a_{t+1} = \leftarrow, \\ i_t & \text{if } a_{t+1} = \mathbf{WRITE}(\sigma). \end{cases}$$

Thus the only real memory problem is: what symbol is currently stored at tape cell i_t?

A tape cell changes only when the machine performs a write action there.

Define the latest previous write to the current cell:

$$\ell_t = \max\{s \le t : i_s = i_t \text{ and } a_s = \mathbf{WRITE}(\sigma) \text{ for some } \sigma\}.$$

If ℓ_t exists and $a_{\ell_t} = \mathbf{WRITE}(\sigma)$, then

$$\sigma_t = \sigma.$$

If ℓ_t does not exist, the symbol is the initial input symbol at position i_t, or $\sqcup$ beyond the input separator.

Head 1: retrieve the latest write

let p_t be the current query position and p_s be the context position of a previous thought $\xi_s = (i_s, a_s, q_s)$.

Head 1: retrieve the latest write

let p_t be the current query position and p_s be the context position of a previous thought $\xi_s = (i_s, a_s, q_s)$.

Choose fixed query/key maps so that the score ξ_s of p_s for p_t is

$$\mathbf{score}_s = -(i_s - i_t)^2 + 1[a_s \text{writes}] + \frac{p_s + 1}{p_t + 1}.$$

Head 1: retrieve the latest write

let p_t be the current query position and p_s be the context position of a previous thought $\xi_s = (i_s, a_s, q_s)$.

Choose fixed query/key maps so that the score ξ_s of p_s for p_t is

$$\mathbf{score}_s = -(i_s - i_t)^2 + \mathbf{1}[a_s \text{writes}] + \frac{p_s + 1}{p_t + 1}.$$

This is a dot product because

$$-(i_s - i_t)^2 = 2 i_t i_s - i_s^2 - i_t^2,$$

and the final term uses the key coordinate $p_s + 1$ and query coordinate $1/(p_t + 1)$.

Head 1: retrieve the latest write

let p_t be the current query position and p_s be the context position of a previous thought $\xi_s = (i_s, a_s, q_s)$.

Choose fixed query/key maps so that the score ξ_s of p_s for p_t is

$$\mathbf{score}_s = -(i_s - i_t)^2 + 1[a_s \text{writes}] + \frac{p_s + 1}{p_t + 1}.$$

This is a dot product because

$$-(i_s - i_t)^2 = 2 i_t i_s - i_s^2 - i_t^2,$$

and the final term uses the key coordinate $p_s + 1$ and query coordinate $1/(p_t + 1)$.

Since tape positions are integers,

$$i_s \neq i_t \quad \Rightarrow \quad -(i_s - i_t)^2 \leq -1.$$

Also, because $p_s < p_t$,

$$0 < \frac{p_s + 1}{p_t + 1} < 1.$$

Head 1: retrieve the latest write

let p_t be the current query position and p_s be the context position of a previous thought $\xi_s = (i_s, a_s, q_s)$.

Choose fixed query/key maps so that the score ξ_s of p_s for p_t is

$$\mathbf{score}_s = -(i_s - i_t)^2 + 1[a_s \text{writes}] + \frac{p_s + 1}{p_t + 1}.$$

This is a dot product because

$$-(i_s - i_t)^2 = 2 i_t i_s - i_s^2 - i_t^2,$$

and the final term uses the key coordinate $p_s + 1$ and query coordinate $1/(p_t + 1)$.

Since tape positions are integers,

$$i_s \neq i_t \quad \Rightarrow \quad -(i_s - i_t)^2 \leq -1.$$

Also, because $p_s < p_t$,

$$0 < \frac{p_s + 1}{p_t + 1} < 1.$$

Therefore:

same cell and write	$1 + \frac{p_s + 1}{p_t + 1} \in (1, 2)$
same cell, not write	$\frac{p_s + 1}{p_t + 1} \in (0, 1)$
different cell	< 1

Head 1: retrieve the latest write

let p_t be the current query position and p_s be the context position of a previous thought $\xi_s = (i_s, a_s, q_s)$.

Choose fixed query/key maps so that the score ξ_s of p_s for p_t is

$$\mathbf{score}_s = -(i_s - i_t)^2 + 1[a_s \text{writes}] + \frac{p_s + 1}{p_t + 1}.$$

This is a dot product because

$$-(i_s - i_t)^2 = 2i_t i_s - i_s^2 - i_t^2,$$

and the final term uses the key coordinate $p_s + 1$ and query coordinate $1/(p_t + 1)$.

Since tape positions are integers,

$$i_s \neq i_t \quad \Rightarrow \quad -(i_s - i_t)^2 \leq -1.$$

Also, because $p_s < p_t$,

$$0 < \frac{p_s + 1}{p_t + 1} < 1.$$

Therefore:

same cell and write	$1 + \frac{p_s + 1}{p_t + 1} \in (1, 2)$
same cell, not write	$\frac{p_s + 1}{p_t + 1} \in (0, 1)$
different cell	< 1

Thus any same-cell write beats every other previous thought. Among same-cell writes, the score is larger exactly for the larger context position p_s.

Head 1: retrieve the latest write

let p_t be the current query position and p_s be the context position of a previous thought $\xi_s = (i_s, a_s, q_s)$.

Choose fixed query/key maps so that the score ξ_s of p_s for p_t is

$$\mathbf{score}_s = -(i_s - i_t)^2 + 1[a_s \text{writes}] + \frac{p_s + 1}{p_t + 1}.$$

This is a dot product because

$$-(i_s - i_t)^2 = 2 i_t i_s - i_s^2 - i_t^2,$$

and the final term uses the key coordinate $p_s + 1$ and query coordinate $1/(p_t + 1)$.

Since tape positions are integers,

$$i_s \neq i_t \quad \Rightarrow \quad -(i_s - i_t)^2 \leq -1.$$

Also, because $p_s < p_t$,

$$0 < \frac{p_s + 1}{p_t + 1} < 1.$$

Therefore:

same cell and write	$1 + \frac{p_s + 1}{p_t + 1} \in (1, 2)$
same cell, not write	$\frac{p_s + 1}{p_t + 1} \in (0, 1)$
different cell	< 1

Thus any same-cell write beats every other previous thought. Among same-cell writes, the score is larger exactly for the larger context position p_s.

Unique hard attention retrieves the latest write ξ_{ℓ_t}, if such a write exists. Its value vector contains i_{ℓ_t} and a_{ℓ_t}.

Head 2: recover the initial symbol

If Head 1 did not retrieve a matching write, then the tape position still contains its initial symbol.

Head 2: recover the initial symbol

If Head 1 did not retrieve a matching write, then the tape position still contains its initial symbol.

Choose fixed query/key maps so that the score ξ_s of p_s for p_t is

$$-(i_s - i_t)^2.$$

Head 2: recover the initial symbol

If Head 1 did not retrieve a matching write, then the tape position still contains its initial symbol.

Choose fixed query/key maps so that the score ξ_s of p_s for p_t is

$$-(i_s - i_t)^2.$$

This is maximized for $i_s = i_t$.
The value map uses $\mathbb{I}[i < n]$ so that

- If $0 < i_t \leq n$, then i_s is an actual input position, so the value map retrieves x_{i_t+1}.
- If $i_t > n$, no input position has address i_s so the value map retrieves $\sqcup$.

Head 2: recover the initial symbol

If Head 1 did not retrieve a matching write, then the tape position still contains its initial symbol.

Choose fixed query/key maps so that the score ξ_s of p_s for p_t is

$$-(i_s - i_t)^2.$$

This is maximized for $i_s = i_t$.
The value map uses $\mathbb{I}[i < n]$ so that

- If $0 < i_t \leq n$, then i_s is an actual input position, so the value map retrieves x_{i_t+1}.
- If $i_t > n$, no input position has address i_s so the value map retrieves $\sqcup$.

We combine the value of both heads with a FFN.

Head 2: recover the initial symbol

If Head 1 did not retrieve a matching write, then the tape position still contains its initial symbol.

Choose fixed query/key maps so that the score ξ_s of p_s for p_t is

$$-(i_s - i_t)^2.$$

This is maximized for $i_s = i_t$.
The value map uses $\mathbb{I}[i < n]$ so that

- If $0 < i_t \leq n$, then i_s is an actual input position, so the value map retrieves x_{i_t+1}.
- If $i_t > n$, no input position has address i_s so the value map retrieves $\sqcup$.

We combine the value of both heads with a FFN.

All the information needed has been computed, and can be extracted with a FFN.

Head 2: recover the initial symbol

If Head 1 did not retrieve a matching write, then the tape position still contains its initial symbol.

Choose fixed query/key maps so that the score ξ_s of p_s for p_t is

$$-(i_s - i_t)^2.$$

This is maximized for $i_s = i_t$.
The value map uses $\mathbb{I}[i < n]$ so that

- If $0 < i_t \leq n$, then i_s is an actual input position, so the value map retrieves x_{i_t+1}.
- If $i_t > n$, no input position has address i_s so the value map retrieves $\sqcup$.

We combine the value of both heads with a FFN.

All the information needed has been computed, and can be extracted with a FFN.

It checks whether Head 1 really found a write to the same cell. Over integer coordinates,

$$\mathbf{1}[i = i_t] = \mathbf{ReLU}\big(\mathbf{1} - |i - i_t|\big),$$

where $|z| = \mathbf{ReLU}(z) + \mathbf{ReLU}(-z)$.

Head 2: recover the initial symbol

If Head 1 did not retrieve a matching write, then the tape position still contains its initial symbol.

Choose fixed query/key maps so that the score ξ_s of p_s for p_t is

$$-(i_s - i_t)^2.$$

This is maximized for $i_s = i_t$.
The value map uses $\mathbb{I}[i < n]$ so that

- If $0 < i_t \leq n$, then i_s is an actual input position, so the value map retrieves x_{i_t+1}.
- If $i_t > n$, no input position has address i_s so the value map retrieves $\sqcup$.

We combine the value of both heads with a FFN.

All the information needed has been computed, and can be extracted with a FFN.

It checks whether Head 1 really found a write to the same cell. Over integer coordinates,

$$\mathbf{1}[i = i_t] = \mathbf{ReLU}\big(\mathbf{1} - |i - i_t|\big),$$

where $|z| = \mathbf{ReLU}(z) + \mathbf{ReLU}(-z)$.

Thus it reconstructs the current symbol σ_t, applies

$$\delta(\sigma_t, q_t) = (a_{t+1}, q_{t+1}),$$

and computes $i_{t+1} \in \{i_t - 1, i_t, i_t + 1\}$.

Head 2: recover the initial symbol

If Head 1 did not retrieve a matching write, then the tape position still contains its initial symbol.

Choose fixed query/key maps so that the score ξ_s of p_s for p_t is

$$-(i_s - i_t)^2.$$

This is maximized for $i_s = i_t$.
The value map uses $\mathbb{I}[i < n]$ so that

- If $0 < i_t \leq n$, then i_s is an actual input position, so the value map retrieves x_{i_t+1}.
- If $i_t > n$, no input position has address i_s so the value map retrieves $\sqcup$.

We combine the value of both heads with a FFN.

All the information needed has been computed, and can be extracted with a FFN.

It checks whether Head 1 really found a write to the same cell. Over integer coordinates,

$$\mathbf{1}[i = i_t] = \mathbf{ReLU}\big(1 - |i - i_t|\big),$$

where $|z| = \mathbf{ReLU}(z) + \mathbf{ReLU}(-z)$.

Thus it reconstructs the current symbol σ_t, applies

$$\delta(\sigma_t, q_t) = (a_{t+1}, q_{t+1}),$$

and computes $i_{t+1} \in \{i_t - 1, i_t, i_t + 1\}$.

The next output is exactly the record

$$\xi_{t+1} = (i_{t+1}, a_{t+1}, q_{t+1}).$$

The simulation invariant

Let

$$\xi_0, \xi_1, \xi_2, \ldots$$

be the transition sequence of M on input x.

The simulation invariant

Let

$$\xi_0, \xi_1, \xi_2, \dots$$

be the transition sequence of M on input x.

Invariant. After $t + 1$ generated thoughts, and before a terminating state is reached,

$$\gamma_0 \gamma_1 \cdots \gamma_t = \xi_0 \xi_1 \cdots \xi_t.$$

The simulation invariant

Let

$$\xi_0, \xi_1, \xi_2, \dots$$

be the transition sequence of M on input x.

Invariant. After $t+1$ generated thoughts, and before a terminating state is reached,

$$\gamma_0 \gamma_1 \cdots \gamma_t = \xi_0 \xi_1 \cdots \xi_t.$$

Base case: $\gamma_0 = \xi_0 = (0, \texttt{start}, s_0)$ is fixed.

The simulation invariant

Let

$$\xi_0, \xi_1, \xi_2, \dots$$

be the transition sequence of M on input x.

Invariant. After $t+1$ generated thoughts, and before a terminating state is reached,

$$\gamma_0 \gamma_1 \cdots \gamma_t = \xi_0 \xi_1 \cdots \xi_t.$$

Base case: $\gamma_0 = \xi_0 = (0, \texttt{start}, s_0)$ is fixed.

Assume

$$\gamma_0 \cdots \gamma_t = \xi_0 \cdots \xi_t.$$

The latest-thought information gives i_t, a_t, q_t.

The simulation invariant

Let

$$\xi_0, \xi_1, \xi_2, \dots$$

be the transition sequence of M on input x.

Invariant. After $t+1$ generated thoughts, and before a terminating state is reached,

$$\gamma_0 \gamma_1 \cdots \gamma_t = \xi_0 \xi_1 \cdots \xi_t.$$

Base case: $\gamma_0 = \xi_0 = (0, \texttt{start}, s_0)$ is fixed.

Assume

$$\gamma_0 \cdots \gamma_t = \xi_0 \cdots \xi_t.$$

The latest-thought information gives i_t, a_t, q_t.

Head 1 retrieves the latest write to cell i_t, if one exists. Head 2 retrieves the initial input symbol at cell i_t, if it exists.

The simulation invariant

Let

$$\xi_0, \xi_1, \xi_2, \dots$$

be the transition sequence of M on input x.

Invariant. After $t+1$ generated thoughts, and before a terminating state is reached,

$$\gamma_0 \gamma_1 \cdots \gamma_t = \xi_0 \xi_1 \cdots \xi_t.$$

Base case: $\gamma_0 = \xi_0 = (0, \texttt{start}, s_0)$ is fixed.

Assume

$$\gamma_0 \cdots \gamma_t = \xi_0 \cdots \xi_t.$$

The latest-thought information gives i_t, a_t, q_t.

Head 1 retrieves the latest write to cell i_t, if one exists. Head 2 retrieves the initial input symbol at cell i_t, if it exists.

The position-wise map reconstructs the true current symbol σ_t, applies δ, and outputs the true next record

$$\xi_{t+1} = (i_{t+1}, a_{t+1}, q_{t+1}).$$

The simulation invariant

Let

$$\xi_0, \xi_1, \xi_2, \dots$$

be the transition sequence of M on input x.

Invariant. After $t+1$ generated thoughts, and before a terminating state is reached,

$$\gamma_0 \gamma_1 \cdots \gamma_t = \xi_0 \xi_1 \cdots \xi_t.$$

Base case: $\gamma_0 = \xi_0 = (0, \texttt{start}, s_0)$ is fixed.

Assume

$$\gamma_0 \cdots \gamma_t = \xi_0 \cdots \xi_t.$$

The latest-thought information gives i_t, a_t, q_t.

Head 1 retrieves the latest write to cell i_t, if one exists. Head 2 retrieves the initial input symbol at cell i_t, if it exists.

The position-wise map reconstructs the true current symbol σ_t, applies δ, and outputs the true next record

$$\xi_{t+1} = (i_{t+1}, a_{t+1}, q_{t+1}).$$

Hence the invariant holds for $t+1$. By induction, the whole Chain-of-thought follows the run of M.

Other Turing-completeness results

Our proof used a **countable** Chain-of-thought alphabet:

$$\gamma_t = (i_t, a_t, q_t) \in \mathbb{Z} \times A \times Q.$$

This lets us keep one fixed transformer while storing the head position directly inside each thought.

Other Turing-completeness results

Our proof used a **countable** Chain-of-thought alphabet:

$$\gamma_t = (i_t, a_t, q_t) \in \mathbb{Z} \times A \times Q.$$

This lets us keep one fixed transformer while storing the head position directly inside each thought.

Other works obtain Turing completeness with a **finite** generated alphabet, but by using other resources.

Work	finite alphabets	extra resource
Amiri–Huang–Rofin–Hahn	no	UHAT with CoT
Pérez–Barceló–Marinković	yes	decoder transformers, arbitrary precision
Merrill–Sabharwal	yes	decoders, layer-norm hash but no PE
Jiang–Hahn–Zetzsche–Lin	yes	softmax CoT, relative PE

Conclusion

We have seen:

- UHAT are included in $\mathbf{AC^0}$
- Further restrictions of UHAT gives exactly the star-free languages
- SMAT can do MAJORITY and PARITY, but are restricted to $\mathbf{TC^0}$
- With a CoT mechanism, we can reach Turing-completeness

Conclusion

We have seen:

- UHAT are included in $\mathbf{AC^0}$
- Further restrictions of UHAT gives exactly the star-free languages
- SMAT can do MAJORITY and PARITY, but are restricted to $\mathbf{TC^0}$
- With a CoT mechanism, we can reach Turing-completeness

The expressivity vastly depends on the chosen model!

Conclusion

We have seen:

- UHAT are included in $\mathbf{AC^0}$
- Further restrictions of UHAT gives exactly the star-free languages
- SMAT can do MAJORITY and PARITY, but are restricted to $\mathbf{TC^0}$
- With a CoT mechanism, we can reach Turing-completeness

The expressivity vastly depends on the chosen model!

Next time: focus on SMAT, which seems closer to actual models.

References

- This chapter is based to a some extent on the lecture "Expressivity of Transformers: Logic, Circuits, and Formal Languages" given at ESSLI 2024

See also:

- David Chiang and Peter Cholak. Overcoming a theoretical limitation of self-attention. *CoRR*, abs/2202.12172, 2022
- Yiding Hao, Dana Angluin, and Robert Frank. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Trans. Assoc. Comput. Linguistics*, 10:800–810, 2022
- Andy Yang, David Chiang, and Dana Angluin. Masked hard-attention transformers recognize exactly the star-free languages. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc
- Jorge PÃ©rez, Pablo BarcelÃ³, and Javier Marinkovic. Attention is turing-complete. *Journal of Machine Learning Research*, 22(75):1–35, 2021

Chapter E.1:

Expressivity of transformers II

Why return to softmax?

Last time: many different models

Why return to softmax?

Last time: many different models
Today: focus on **softmax** attention
$\longrightarrow$ practically relevant

Why return to softmax?

Last time: many different models
Today: focus on **softmax** attention
$\longrightarrow$ practically relevant

What does **softmax aggregation add to expressivity**?

Why return to softmax?

Last time: many different models
Today: focus on **softmax** attention
$\longrightarrow$ practically relevant

What does **softmax aggregation add to expressivity**?

Contrary to UHAT, at position i, it does not select one value vector. Write the attention score and weight as

$$s_{i,j} = q_i^\top k_j, \qquad \alpha_{i,j} = \frac{e^{s_{i,j}}}{\sum_{\ell=1}^{n} e^{s_{i,\ell}}}.$$

Then

$$z_i = \sum_{j=1}^{n} \alpha_{i,j} v_j + y_i.$$

Why return to softmax?

Last time: many different models
Today: focus on **softmax** attention
→ practically relevant

What does **softmax aggregation add to expressivity**?

Contrary to UHAT, at position i, it does not select one value vector. Write the attention score and weight as

$$s_{i,j} = q_i^\top k_j, \qquad \alpha_{i,j} = \frac{e^{s_{i,j}}}{\sum_{\ell=1}^{n} e^{s_{i,\ell}}}.$$

Then

$$z_i = \sum_{j=1}^{n} \alpha_{i,j} v_j + y_i.$$

Sharp regime. If one score dominates,

$$s_{i,j^\star} \gg s_{i,j} \qquad (j \neq j^\star),$$

then

$$\alpha_{i,j^\star} \approx 1.$$

Softmax behaves almost like hard attention.

Why return to softmax?

Last time: many different models
Today: focus on **softmax** attention
→ practically relevant

What does **softmax aggregation add to expressivity**?

Contrary to UHAT, at position i, it does not select one value vector. Write the attention score and weight as

$$s_{i,j} = q_i^\top k_j, \qquad \alpha_{i,j} = \frac{e^{s_{i,j}}}{\sum_{\ell=1}^{n} e^{s_{i,\ell}}}.$$

Then

$$z_i = \sum_{j=1}^{n} \alpha_{i,j} v_j + y_i.$$

Sharp regime. If one score dominates,

$$s_{i,j^\star} \gg s_{i,j} \qquad (j \neq j^\star),$$

then

$$\alpha_{i,j^\star} \approx 1.$$

Softmax behaves almost like hard attention.

Flat regime. If all scores are equal,

$$s_{i,1} = \cdots = s_{i,n},$$

then

$$\alpha_{i,j} = \frac{1}{n}, \qquad z_i = \frac{1}{n} \sum_{j=1}^{n} v_j + y_i.$$

Softmax computes an average.

Why return to softmax?

Last time: many different models
Today: focus on **softmax** attention
→ practically relevant

What does **softmax aggregation add to expressivity**?

Contrary to UHAT, at position i, it does not select one value vector. Write the attention score and weight as

$$s_{i,j} = q_i^\top k_j, \qquad \alpha_{i,j} = \frac{e^{s_{i,j}}}{\sum_{\ell=1}^{n} e^{s_{i,\ell}}}.$$

Then

$$z_i = \sum_{j=1}^{n} \alpha_{i,j} v_j + y_i.$$

Sharp regime. If one score dominates,

$$s_{i,j^\star} \gg s_{i,j} \qquad (j \neq j^\star),$$

then

$$\alpha_{i,j^\star} \approx 1.$$

Softmax behaves almost like hard attention.

Flat regime. If all scores are equal,

$$s_{i,1} = \cdots = s_{i,n},$$

then

$$\alpha_{i,j} = \frac{1}{n}, \qquad z_i = \frac{1}{n} \sum_{j=1}^{n} v_j + y_i.$$

Softmax computes an average.

As a rule of thumb: **SMAT can retreive a position and count**.

Why return to softmax?

Last time: many different models
Today: focus on **softmax** attention
→ practically relevant

What does **softmax aggregation add to expressivity**?

Contrary to UHAT, at position i, it does not select one value vector. Write the attention score and weight as

$$s_{i,j} = q_i^\top k_j, \qquad \alpha_{i,j} = \frac{e^{s_{i,j}}}{\sum_{\ell=1}^{n} e^{s_{i,\ell}}}.$$

Then

$$z_i = \sum_{j=1}^{n} \alpha_{i,j} v_j + y_i.$$

Sharp regime. If one score dominates,

$$s_{i,j^\star} \gg s_{i,j} \qquad (j \neq j^\star),$$

then

$$\alpha_{i,j^\star} \approx 1.$$

Softmax behaves almost like hard attention.

Flat regime. If all scores are equal,

$$s_{i,1} = \cdots = s_{i,n},$$

then

$$\alpha_{i,j} = \frac{1}{n}, \qquad z_i = \frac{1}{n} \sum_{j=1}^{n} v_j + y_i.$$

Softmax computes an average.

As a rule of thumb: **SMAT can retreive a position and count**.
→ Both were used to compute **PARITY**

Roadmap

Today's plan:

Roadmap

Today's plan:

Lower bound. We first ask what a fixed **SMAT** can do in general.

SMAT $\subseteq$ a logic with basic arithmetic and counting.

Roadmap

Today's plan:

Lower bound. We first ask what a fixed **SMAT** can do in general.

SMAT $\subseteq$ a logic with basic arithmetic and counting.

Upper bound. We then introduce C-RASP, a small programming language for counting like transformers.

C-RASP programs $\leftrightarrow$ softmax transformers.

It gives clean constructions instead of hand-built matrices.

Roadmap

Today's plan:

Lower bound. We first ask what a fixed **SMAT** can do in general.

SMAT $\subseteq$ a logic with basic arithmetic and counting.

Upper bound. We then introduce C-RASP, a small programming language for counting like transformers.

C-RASP programs $\leftrightarrow$ softmax transformers.

It gives clean constructions instead of hand-built matrices.

Depth hierarchy. If a task can be done by transformers, how many layers are needed?
We will obtain a logical characterization of the languages recognizable by a transformer with L layers.

Roadmap

Today's plan:

Lower bound. We first ask what a fixed **SMAT** can do in general.

SMAT $\subseteq$ a logic with basic arithmetic and counting.

Upper bound. We then introduce C-RASP, a small programming language for counting like transformers.

C-RASP programs $\leftrightarrow$ softmax transformers.

It gives clean constructions instead of hand-built matrices.

Depth hierarchy. If a task can be done by transformers, how many layers are needed?
We will obtain a logical characterization of the languages recognizable by a transformer with L layers.

Length-generalizability. Some insights on how to go from expressibility to learnability.

Logical limitations

The logic $\mathbf{FOC}[\mathbf{MOD}, +]$

We introduce a logic with a counting mechanism.

$\mathbf{FOC}[\mathbf{MOD}, +]$

A formula is interpreted over a word $w = w_1 \cdots w_n$.
It has two sorts of variables:

- **positions** $p, q, \ldots \in \{1, \ldots, n\}$;
- **counts** $x, y, \ldots \in \mathbb{R}$.

It is built using:

- letter predicates $Q_a(p)$, meaning $w_p = a$;
- modular predicates $\mathbf{MOD}_m^r(p)$, meaning $p \equiv r \pmod m$;
- linear count terms $c_0 + c_1 x_1 + \cdots + c_k x_k$;
- comparisons of count terms: $t = t'$, $t < t'$;
- Boolean connectives $\neg, \wedge, \vee$;
- quantifiers over counts: $\exists x, \forall x$;
- counting quantifiers $\exists^{=x} p\, \varphi(p)$.

The logic $\mathbf{FOC}[\mathbf{MOD}, +]$

We introduce a logic with a counting mechanism.

$\mathbf{FOC}[\mathbf{MOD}, +]$

A formula is interpreted over a word $w = w_1 \cdots w_n$.
It has two sorts of variables:

- **positions** $p, q, \ldots \in \{1, \ldots, n\}$;
- **counts** $x, y, \ldots \in \mathbb{R}$.

It is built using:

- letter predicates $Q_a(p)$, meaning $w_p = a$;
- modular predicates $\mathbf{MOD}^r_m(p)$, meaning $p \equiv r \pmod m$;
- linear count terms $c_0 + c_1 x_1 + \cdots + c_k x_k$;
- comparisons of count terms: $t = t'$, $t < t'$;
- Boolean connectives $\neg, \wedge, \vee$;
- quantifiers over counts: $\exists x, \forall x$;
- counting quantifiers $\exists^{=x} p\, \varphi(p)$.

The counting quantifier connects positions and numbers:

$\exists^{=x} p\, \varphi(p)$

means: exactly x positions p satisfy $\varphi(p)$.

The logic $\mathbf{FOC}[\mathbf{MOD}, +]$

We introduce a logic with a counting mechanism.

$\mathbf{FOC}[\mathbf{MOD}, +]$

A formula is interpreted over a word $w = w_1 \cdots w_n$.
It has two sorts of variables:

- **positions** $p, q, \ldots \in \{1, \ldots, n\}$;
- **counts** $x, y, \ldots \in \mathbb{R}$.

It is built using:

- letter predicates $Q_a(p)$, meaning $w_p = a$;
- modular predicates $\mathbf{MOD}^r_m(p)$, meaning $p \equiv r \pmod m$;
- linear count terms $c_0 + c_1 x_1 + \cdots + c_k x_k$;
- comparisons of count terms: $t = t'$, $t < t'$;
- Boolean connectives $\neg, \wedge, \vee$;
- quantifiers over counts: $\exists x, \forall x$;
- counting quantifiers $\exists^{=x} p\, \varphi(p)$.

The counting quantifier connects positions and numbers:

$\exists^{=x} p\, \varphi(p)$

means: exactly x positions p satisfy $\varphi(p)$.

Example

Over $\Sigma = \{0, 1\}$:

- Equal number of 0's and 1's:

$$\exists x \left(\exists^{=x} p\, Q_0(p) \wedge \exists^{=x} p\, Q_1(p) \right).$$

The logic $\mathbf{FOC}[\mathbf{MOD}, +]$

We introduce a logic with a counting mechanism.

$\mathbf{FOC}[\mathbf{MOD}, +]$

A formula is interpreted over a word $w = w_1 \cdots w_n$.
It has two sorts of variables:

- **positions** $p, q, \ldots \in \{1, \ldots, n\}$;
- **counts** $x, y, \ldots \in \mathbb{R}$.

It is built using:

- letter predicates $Q_a(p)$, meaning $w_p = a$;
- modular predicates $\mathbf{MOD}^r_m(p)$, meaning $p \equiv r \pmod m$;
- linear count terms $c_0 + c_1 x_1 + \cdots + c_k x_k$;
- comparisons of count terms: $t = t'$, $t < t'$;
- Boolean connectives $\neg, \wedge, \vee$;
- quantifiers over counts: $\exists x, \forall x$;
- counting quantifiers $\exists^{=x} p\, \varphi(p)$.

The counting quantifier connects positions and numbers:

$\exists^{=x} p\, \varphi(p)$

means: exactly x positions p satisfy $\varphi(p)$.

Example

Over $\Sigma = \{0, 1\}$:

- Equal number of 0's and 1's:

$$\exists x \left(\exists^{=x} p\, Q_0(p) \wedge \exists^{=x} p\, Q_1(p) \right).$$

- Twice as many 1's as 0's:

$$\exists x\, \exists y \left(\exists^{=x} p\, Q_0(p) \wedge \exists^{=y} p\, Q_1(p) \wedge y = 2x \right).$$

The logic $\mathbf{FOC}[\mathbf{MOD}, +]$

We introduce a logic with a counting mechanism.

$\mathbf{FOC}[\mathbf{MOD}, +]$

A formula is interpreted over a word $w = w_1 \cdots w_n$.
It has two sorts of variables:

- **positions** $p, q, \ldots \in \{1, \ldots, n\}$;
- **counts** $x, y, \ldots \in \mathbb{R}$.

It is built using:

- letter predicates $Q_a(p)$, meaning $w_p = a$;
- modular predicates $\mathbf{MOD}^r_m(p)$, meaning $p \equiv r \pmod m$;
- linear count terms $c_0 + c_1 x_1 + \cdots + c_k x_k$;
- comparisons of count terms: $t = t'$, $t < t'$;
- Boolean connectives $\neg, \wedge, \vee$;
- quantifiers over counts: $\exists x, \forall x$;
- counting quantifiers $\exists^{=x} p\, \varphi(p)$.

The counting quantifier connects positions and numbers:

$\exists^{=x} p\, \varphi(p)$

means: exactly x positions p satisfy $\varphi(p)$.

Example

Over $\Sigma = \{0, 1\}$:

- Equal number of 0's and 1's:

$$\exists x \left(\exists^{=x} p\, Q_0(p) \wedge \exists^{=x} p\, Q_1(p) \right).$$

- Twice as many 1's as 0's:

$$\exists x\, \exists y \left(\exists^{=x} p\, Q_0(p) \wedge \exists^{=y} p\, Q_1(p) \wedge y = 2x \right).$$

- Alternating word $0101\cdots$:

$$\forall p \left(\mathbf{MOD}^1_2(p) \to Q_0(p) \right) \wedge \forall p \left(\mathbf{MOD}^0_2(p) \to Q_1(p) \right).$$

A sharper upper bound for SMAT

Since **SMAT** can average over all positions, it is natural to compare it with threshold circuits:

$$\textbf{SMAT} \subseteq \mathbf{TC^0}.$$

But this is not the most informative upper bound.

A sharper upper bound for SMAT

Since **SMAT** can average over all positions, it is natural to compare it with threshold circuits:

$$\textbf{SMAT} \subseteq \mathbf{TC^0}.$$

But this is not the most informative upper bound.

The logic $\mathbf{FOC}[\mathbf{MOD}, +]$ is more structured:

- it can count positions satisfying definable properties;
- it can use modular information about positions;
- it can combine counts by linear arithmetic.

So it is tailored to the way softmax attention aggregates information.

A sharper upper bound for SMAT

Since **SMAT** can average over all positions, it is natural to compare it with threshold circuits:

$$\textbf{SMAT} \subseteq \mathbf{TC^0}.$$

But this is not the most informative upper bound.

The logic $\mathbf{FOC}[\mathbf{MOD}, +]$ is more structured:

- it can count positions satisfying definable properties;
- it can use modular information about positions;
- it can combine counts by linear arithmetic.

So it is tailored to the way softmax attention aggregates information.

Theorem E.1.1 (Chiang–Cholak–Pillay)

Every language recognized by a **SMAT** with **rational sinusoidals PE** and **finite precision** are definable in

$$\mathbf{FOC}[\mathbf{MOD}, +].$$

A sharper upper bound for SMAT

Since **SMAT** can average over all positions, it is natural to compare it with threshold circuits:

$$\textbf{SMAT} \subseteq \mathbf{TC^0}.$$

But this is not the most informative upper bound.

The logic $\mathbf{FOC[MOD, +]}$ is more structured:

- it can count positions satisfying definable properties;
- it can use modular information about positions;
- it can combine counts by linear arithmetic.

So it is tailored to the way softmax attention aggregates information.

Theorem E.1.1 (Chiang–Cholak–Pillay)

Every language recognized by a **SMAT** with **rational sinusoidals PE** and **finite precision** are definable in

$$\mathbf{FOC[MOD, +]}.$$

All PEs are of the form $\boldsymbol{i \mapsto \sin(2\pi q i)}$ for $\boldsymbol{q \in \mathbb{Q}}$

A sharper upper bound for SMAT

Since **SMAT** can average over all positions, it is natural to compare it with threshold circuits:

$$\textbf{SMAT} \subseteq \mathbf{TC^0}.$$

But this is not the most informative upper bound.

The logic $\mathbf{FOC[MOD, +]}$ is more structured:

- it can count positions satisfying definable properties;
- it can use modular information about positions;
- it can combine counts by linear arithmetic.

So it is tailored to the way softmax attention aggregates information.

Theorem E.1.1 (Chiang–Cholak–Pillay)

Every language recognized by a **SMAT** with **rational sinusoidals PE** and **finite precision** are definable in

$$\mathbf{FOC[MOD, +]}.$$

All PEs are of the form $i \mapsto \sin(2\pi q i)$ for $q \in \mathbb{Q}$

This is a **better upper bound** than $\mathbf{TC^0}$:

$$\mathbf{FOC[MOD, +]} \subsetneq \mathbf{TC^0}.$$

A sharper upper bound for SMAT

Since **SMAT** can average over all positions, it is natural to compare it with threshold circuits:

$$\textbf{SMAT} \subseteq \mathbf{TC^0}.$$

But this is not the most informative upper bound.

The logic $\mathbf{FOC[MOD, +]}$ is more structured:

- it can count positions satisfying definable properties;
- it can use modular information about positions;
- it can combine counts by linear arithmetic.

So it is tailored to the way softmax attention aggregates information.

Theorem E.1.1 (Chiang–Cholak–Pillay)

Every language recognized by a **SMAT** with **rational sinusoidals PE** and **finite precision** are definable in

$$\mathbf{FOC[MOD, +]}.$$

All PEs are of the form $i \mapsto \sin(2\pi q i)$ for $q \in \mathbb{Q}$

This is a **better upper bound** than $\mathbf{TC^0}$:

$$\mathbf{FOC[MOD, +]} \subsetneq \mathbf{TC^0}.$$

This is witnessed by $\{0^n 1^n \mid n \in \mathbb{N}\}$.

Proof idea: names again

Fix a fixed-precision **SMAT** transformer T: all coordinates lie in a finite set F.

Proof idea: names again

Fix a fixed-precision **SMAT** transformer T: all coordinates lie in a finite set F.

As for **UHAT** $\subseteq$ $\mathbf{AC^0}$, we do not manipulate vectors directly: we give them **names**.

Proof idea: names again

Fix a fixed-precision **SMAT** transformer T: all coordinates lie in a finite set F.

As for **UHAT** $\subseteq$ $\mathbf{AC^0}$, we do not manipulate vectors directly: we give them **names**.

For each layer ℓ, fix a finite set N_ℓ and an interpretation map

$$\tau_\ell : N_\ell \to F^d.$$

A name $\eta \in N_\ell$ denotes the vector $\tau_\ell(\eta)$.

Proof idea: names again

Fix a fixed-precision **SMAT** transformer T: all coordinates lie in a finite set F.

As for **UHAT** $\subseteq$ $\mathbf{AC^0}$, we do not manipulate vectors directly: we give them **names**.

For each layer ℓ, fix a finite set N_ℓ and an interpretation map

$$\tau_\ell : N_\ell \to F^d.$$

A name $\eta \in N_\ell$ denotes the vector $\tau_\ell(\eta)$.

Induction invariant:

$$\mathbf{Name}_\ell^\eta(p) \iff y_p^{(\ell)} = \tau_\ell(\eta).$$

Each $\mathbf{Name}_\ell^\eta(p)$ is a formula of $\mathbf{FOC}[\mathbf{MOD}, +]$.

Proof idea: names again

Fix a fixed-precision **SMAT** transformer T: all coordinates lie in a finite set F.

As for **UHAT** $\subseteq$ **AC**^{0}, we do not manipulate vectors directly: we give them **names**.

For each layer ℓ, fix a finite set N_ℓ and an interpretation map

$$\tau_\ell : N_\ell \to F^d.$$

A name $\eta \in N_\ell$ denotes the vector $\tau_\ell(\eta)$.

Induction invariant:

$$\mathbf{Name}_\ell^\eta(p) \iff y_p^{(\ell)} = \tau_\ell(\eta).$$

Each $\mathbf{Name}_\ell^\eta(p)$ is a formula of $\mathbf{FOC}[\mathbf{MOD}, +]$.

At layer 0,

$$y_p^{(0)} = \mathbf{we}(x_p) + \mathbf{pe}(p).$$

The PE is periodic in p for some period m

Proof idea: names again

Fix a fixed-precision **SMAT** transformer T: all coordinates lie in a finite set F.

As for **UHAT** $\subseteq$ $\mathbf{AC^0}$, we do not manipulate vectors directly: we give them **names**.

For each layer ℓ, fix a finite set N_ℓ and an interpretation map

$$\tau_\ell : N_\ell \to F^d.$$

A name $\eta \in N_\ell$ denotes the vector $\tau_\ell(\eta)$.

Induction invariant:

$$\mathbf{Name}_\ell^\eta(p) \iff y_p^{(\ell)} = \tau_\ell(\eta).$$

Each $\mathbf{Name}_\ell^\eta(p)$ is a formula of $\mathbf{FOC}[\mathbf{MOD}, +]$.

At layer $\mathbf{0}$,

$$y_p^{(0)} = \mathbf{we}(x_p) + \mathbf{pe}(p).$$

The PE is periodic in p for some period m

Use names

$$\eta = (a, r),$$

$$a \in \Sigma, \qquad r \in \{0, \dots, m-1\}.$$

Define

$$\tau_0(a, r) = \mathbf{we}(a) + \mathbf{pe}(p) \quad (p \equiv r \bmod m).$$

Proof idea: names again

Fix a fixed-precision **SMAT** transformer T: all coordinates lie in a finite set F.

As for **UHAT** $\subseteq$ $\mathbf{AC^0}$, we do not manipulate vectors directly: we give them **names**.

For each layer ℓ, fix a finite set N_ℓ and an interpretation map

$$\tau_\ell : N_\ell \to F^d.$$

A name $\eta \in N_\ell$ denotes the vector $\tau_\ell(\eta)$.

Induction invariant:

$$\mathbf{Name}_\ell^\eta(p) \iff y_p^{(\ell)} = \tau_\ell(\eta).$$

Each $\mathbf{Name}_\ell^\eta(p)$ is a formula of $\mathbf{FOC}[\mathbf{MOD}, +]$.

At layer $\mathbf{0}$,

$$y_p^{(0)} = \mathbf{we}(x_p) + \mathbf{pe}(p).$$

The PE is periodic in p for some period m

Use names

$$\eta = (a, r),$$
$$a \in \Sigma, \qquad r \in \{0, \dots, m-1\}.$$

Define

$$\tau_0(a, r) = \mathbf{we}(a) + \mathbf{pe}(p) \quad (p \equiv r \bmod m).$$

The formula is

$$\mathbf{Name}_0^{(a,r)}(p) := Q_a(p) \wedge \mathbf{MOD}_m^r(p).$$

Proof idea: names again

Fix a fixed-precision **SMAT** transformer T: all coordinates lie in a finite set F.

As for **UHAT** $\subseteq$ $\mathbf{AC^0}$, we do not manipulate vectors directly: we give them **names**.

For each layer ℓ, fix a finite set N_ℓ and an interpretation map

$$\tau_\ell : N_\ell \to F^d.$$

A name $\eta \in N_\ell$ denotes the vector $\tau_\ell(\eta)$.

Induction invariant:

$$\mathbf{Name}_\ell^\eta(p) \iff y_p^{(\ell)} = \tau_\ell(\eta).$$

Each $\mathbf{Name}_\ell^\eta(p)$ is a formula of $\mathbf{FOC}[\mathbf{MOD}, +]$.

At layer $\mathbf{0}$,

$$y_p^{(0)} = \mathbf{we}(x_p) + \mathbf{pe}(p).$$

The PE is periodic in p for some period m

Use names

$$\eta = (a, r),$$
$$a \in \Sigma, \qquad r \in \{0, \dots, m-1\}.$$

Define

$$\tau_0(a, r) = \mathbf{we}(a) + \mathbf{pe}(p) \quad (p \equiv r \bmod m).$$

The formula is

$$\mathbf{Name}_0^{(a,r)}(p) := Q_a(p) \wedge \mathbf{MOD}_m^r(p).$$

Hence

$$\mathbf{Name}_0^\eta(p) \iff y_p^{(0)} = \tau_0(\eta).$$

Attention layer

Assume the invariant at layer ℓ. If $\mathbf{Name}_\ell^\eta(p)$, then

$$q(\eta) = Q^{(\ell)}\tau_\ell(\eta),$$
$$k(\eta) = K^{(\ell)}\tau_\ell(\eta),$$
$$v(\eta) = V^{(\ell)}\tau_\ell(\eta).$$

Attention layer

Assume the invariant at layer ℓ. If $\mathbf{Name}_\ell^\eta(p)$, then

$$q(\eta) = Q^{(\ell)}\tau_\ell(\eta),$$
$$k(\eta) = K^{(\ell)}\tau_\ell(\eta),$$
$$v(\eta) = V^{(\ell)}\tau_\ell(\eta).$$

Scores depend only on two names:

$$s^{(\ell)}(\eta, \theta) = q(\eta)^\top k(\theta).$$

Attention layer

Assume the invariant at layer ℓ. If $\mathbf{Name}_\ell^\eta(p)$, then

$$q(\eta) = Q^{(\ell)}\tau_\ell(\eta),$$
$$k(\eta) = K^{(\ell)}\tau_\ell(\eta),$$
$$v(\eta) = V^{(\ell)}\tau_\ell(\eta).$$

Scores depend only on two names:

$$s^{(\ell)}(\eta, \theta) = q(\eta)^\top k(\theta).$$

All position-wise maps are finite lookup tables:

$$\eta \mapsto Q\tau_\ell(\eta), K\tau_\ell(\eta), V\tau_\ell(\eta).$$

Attention layer

Assume the invariant at layer ℓ. If $\mathbf{Name}_\ell^\eta(p)$, then

$$q(\eta) = Q^{(\ell)}\tau_\ell(\eta),$$
$$k(\eta) = K^{(\ell)}\tau_\ell(\eta),$$
$$v(\eta) = V^{(\ell)}\tau_\ell(\eta).$$

Scores depend only on two names:

$$s^{(\ell)}(\eta,\theta) = q(\eta)^\top k(\theta).$$

All position-wise maps are finite lookup tables:

$$\eta \mapsto Q\tau_\ell(\eta), K\tau_\ell(\eta), V\tau_\ell(\eta).$$

Fix a query position i with name η:

$$\mathbf{Name}_\ell^\eta(i).$$

Then every key position p has some name $\theta \in N_\ell$.

Attention layer

Assume the invariant at layer ℓ. If $\mathbf{Name}_\ell^\eta(p)$, then

$$q(\eta) = Q^{(\ell)}\tau_\ell(\eta),$$
$$k(\eta) = K^{(\ell)}\tau_\ell(\eta),$$
$$v(\eta) = V^{(\ell)}\tau_\ell(\eta).$$

Scores depend only on two names:

$$s^{(\ell)}(\eta,\theta) = q(\eta)^\top k(\theta).$$

All position-wise maps are finite lookup tables:

$$\eta \mapsto Q\tau_\ell(\eta), K\tau_\ell(\eta), V\tau_\ell(\eta).$$

Fix a query position i with name η:

$$\mathbf{Name}_\ell^\eta(i).$$

Then every key position p has some name $\theta \in N_\ell$.

For each key name θ, introduce a count variable x_θ:

$$\exists^{=x_\theta} p\ \mathbf{Name}_\ell^\theta(p).$$

So x_θ is the number of positions whose layer-ℓ vector is $\tau_\ell(\theta)$.

Attention layer

Assume the invariant at layer ℓ. If $\mathbf{Name}_\ell^\eta(p)$, then

$$q(\eta) = Q^{(\ell)}\tau_\ell(\eta),$$
$$k(\eta) = K^{(\ell)}\tau_\ell(\eta),$$
$$v(\eta) = V^{(\ell)}\tau_\ell(\eta).$$

Scores depend only on two names:

$$s^{(\ell)}(\eta,\theta) = q(\eta)^\top k(\theta).$$

All position-wise maps are finite lookup tables:

$$\eta \mapsto Q\tau_\ell(\eta), K\tau_\ell(\eta), V\tau_\ell(\eta).$$

Fix a query position i with name η:

$$\mathbf{Name}_\ell^\eta(i).$$

Then every key position p has some name $\theta \in N_\ell$.

For each key name θ, introduce a count variable x_θ:

$$\exists^{=x_\theta} p\ \mathbf{Name}_\ell^\theta(p).$$

So x_θ is the number of positions whose layer-ℓ vector is $\tau_\ell(\theta)$.

The softmax numerator and denominator become grouped sums:

$$A(\eta,\bar{x}) = \sum_{\theta\in N_\ell} x_\theta e^{s_{\eta,\theta}} v_\theta,$$

$$B(\eta,\bar{x}) = \sum_{\theta\in N_\ell} x_\theta e^{s_{\eta,\theta}}.$$

Hence

$$\mathbf{Att}_\ell(\eta,\bar{x}) = \frac{A(\eta,\bar{x})}{B(\eta,\bar{x})}.$$

Attention layer

Assume the invariant at layer ℓ. If $\mathbf{Name}_\ell^\eta(p)$, then

$$q(\eta) = Q^{(\ell)}\tau_\ell(\eta),$$
$$k(\eta) = K^{(\ell)}\tau_\ell(\eta),$$
$$v(\eta) = V^{(\ell)}\tau_\ell(\eta).$$

Scores depend only on two names:

$$s^{(\ell)}(\eta,\theta) = q(\eta)^\top k(\theta).$$

All position-wise maps are finite lookup tables:

$$\eta \mapsto Q\tau_\ell(\eta), K\tau_\ell(\eta), V\tau_\ell(\eta).$$

Fix a query position i with name η:

$$\mathbf{Name}_\ell^\eta(i).$$

Then every key position p has some name $\theta \in N_\ell$.

For each key name θ, introduce a count variable x_θ:

$$\exists^{=x_\theta} p\ \mathbf{Name}_\ell^\theta(p).$$

So x_θ is the number of positions whose layer-ℓ vector is $\tau_\ell(\theta)$.

The softmax numerator and denominator become grouped sums:

$$A(\eta,\bar{x}) = \sum_{\theta\in N_\ell} x_\theta e^{s_{\eta,\theta}} v_\theta,$$

$$B(\eta,\bar{x}) = \sum_{\theta\in N_\ell} x_\theta e^{s_{\eta,\theta}}.$$

Hence

$$\mathbf{Att}_\ell(\eta,\bar{x}) = \frac{A(\eta,\bar{x})}{B(\eta,\bar{x})}.$$

For $u \in F$, we can test equality of the attention score with u with formulas of $\mathbf{FOC}[\mathbf{MOD},+]$.

Defining the next name

Let $\boldsymbol{u}$ be the attention output. Set

$$\boldsymbol{z = u + \tau_\ell(\eta).}$$

The rest is pointwise:

$$\mathbf{Upd}_\ell(\eta, u) = \boldsymbol{W_2}\mathbf{ReLU}(\boldsymbol{W_1 z + b_1}) + \boldsymbol{b_2} + \boldsymbol{z}.$$

Defining the next name

Let $\boldsymbol{u}$ be the attention output. Set

$$\boldsymbol{z = u + \tau_\ell(\eta).}$$

The rest is pointwise:

$$\mathbf{Upd}_\ell(\eta, u) = \boldsymbol{W_2}\mathbf{ReLU}(\boldsymbol{W_1 z + b_1}) + \boldsymbol{b_2} + \boldsymbol{z}.$$

Let $\boldsymbol{N_{\ell+1}}$ contain one name per possible value of $\mathbf{Upd}_\ell(\eta, u)$, with

$$\boldsymbol{\tau_{\ell+1}(\mu) = \mathbf{Upd}_\ell(\eta, u).}$$

Defining the next name

Let $\boldsymbol{u}$ be the attention output. Set

$$\boldsymbol{z = u + \tau_\ell(\eta).}$$

The rest is pointwise:

$$\mathbf{Upd}_\ell(\eta, u) = \boldsymbol{W_2}\mathbf{ReLU}(\boldsymbol{W_1 z + b_1}) + \boldsymbol{b_2} + \boldsymbol{z}.$$

Let $\boldsymbol{N_{\ell+1}}$ contain one name per possible value of $\mathbf{Upd}_\ell(\eta, u)$, with

$$\boldsymbol{\tau_{\ell+1}(\mu) = \mathbf{Upd}_\ell(\eta, u).}$$

Define $\mathbf{Name}^{\mu}_{\ell+1}(i)$ by choosing $\eta, u, \bar{x}$ such that:

- $\mathbf{Name}^{\eta}_{\ell}(i)$;
- each x_θ counts $\mathbf{Name}^{\theta}_{\ell}$-positions;
- $\mathbf{Att}_\ell(\eta, \bar{x}) = u$;
- $\mathbf{Upd}_\ell(\eta, u) = \tau_{\ell+1}(\mu)$.

Defining the next name

Let $\boldsymbol{u}$ be the attention output. Set

$$\boldsymbol{z = u + \tau_\ell(\eta).}$$

The rest is pointwise:

$$\mathbf{Upd}_\ell(\eta, u) = \boldsymbol{W_2}\mathbf{ReLU}(\boldsymbol{W_1 z + b_1}) + \boldsymbol{b_2} + \boldsymbol{z}.$$

Let $\boldsymbol{N_{\ell+1}}$ contain one name per possible value of $\mathbf{Upd}_\ell(\eta, \boldsymbol{u})$, with

$$\boldsymbol{\tau_{\ell+1}(\mu) = \mathbf{Upd}_\ell(\eta, u).}$$

Define $\mathbf{Name}^{\mu}_{\ell+1}(i)$ by choosing $\eta, \boldsymbol{u}, \bar{\boldsymbol{x}}$ such that:

- $\mathbf{Name}^{\eta}_{\ell}(i)$;
- each x_θ counts $\mathbf{Name}^{\theta}_{\ell}$-positions;
- $\mathbf{Att}_\ell(\eta, \bar{\boldsymbol{x}}) = \boldsymbol{u}$;
- $\mathbf{Upd}_\ell(\eta, \boldsymbol{u}) = \boldsymbol{\tau_{\ell+1}(\mu)}$.

This is an $\mathbf{FOC}[\mathbf{MOD}, +]$-formula: only the names are counted.

Defining the next name

Let $\boldsymbol{u}$ be the attention output. Set

$$z = u + \tau_\ell(\eta).$$

The rest is pointwise:

$$\mathbf{Upd}_\ell(\eta, u) = W_2\mathbf{ReLU}(W_1 z + b_1) + b_2 + z.$$

Let $N_{\ell+1}$ contain one name per possible value of $\mathbf{Upd}_\ell(\eta, u)$, with

$$\tau_{\ell+1}(\mu) = \mathbf{Upd}_\ell(\eta, u).$$

Define $\mathbf{Name}^{\mu}_{\ell+1}(i)$ by choosing $\eta, u, \bar{x}$ such that:

- $\mathbf{Name}^{\eta}_{\ell}(i)$;
- each x_θ counts $\mathbf{Name}^{\theta}_{\ell}$-positions;
- $\mathbf{Att}_\ell(\eta, \bar{x}) = u$;
- $\mathbf{Upd}_\ell(\eta, u) = \tau_{\ell+1}(\mu)$.

This is an $\mathbf{FOC}[\mathbf{MOD}, +]$-formula: only the names are counted.

If $\mathbf{Name}^{\mu}_{\ell+1}(i)$ holds, the chosen counts give the true softmax average at i.
Hence

$$y_i^{(\ell+1)} = \tau_{\ell+1}(\mu).$$

Defining the next name

Let $\boldsymbol{u}$ be the attention output. Set

$$\boldsymbol{z = u + \tau_\ell(\eta).}$$

The rest is pointwise:

$$\mathbf{Upd}_\ell(\eta, u) = \boldsymbol{W_2}\mathbf{ReLU}(\boldsymbol{W_1 z + b_1}) + \boldsymbol{b_2} + \boldsymbol{z}.$$

Let $\boldsymbol{N_{\ell+1}}$ contain one name per possible value of $\mathbf{Upd}_\ell(\eta, \boldsymbol{u})$, with

$$\boldsymbol{\tau_{\ell+1}(\mu) = \mathbf{Upd}_\ell(\eta, u).}$$

Define $\mathbf{Name}^{\mu}_{\ell+1}(i)$ by choosing $\eta, \boldsymbol{u}, \bar{\boldsymbol{x}}$ such that:

- $\mathbf{Name}^{\eta}_{\ell}(i)$;
- each $\boldsymbol{x_\theta}$ counts $\mathbf{Name}^{\theta}_{\ell}$-positions;
- $\mathbf{Att}_\ell(\eta, \bar{\boldsymbol{x}}) = \boldsymbol{u}$;
- $\mathbf{Upd}_\ell(\eta, \boldsymbol{u}) = \boldsymbol{\tau_{\ell+1}(\mu)}$.

This is an $\mathbf{FOC}[\mathbf{MOD}, +]$-formula: only the names are counted.

If $\mathbf{Name}^{\mu}_{\ell+1}(i)$ holds, the chosen counts give the true softmax average at $\boldsymbol{i}$.
Hence

$$\boldsymbol{y_i^{(\ell+1)} = \tau_{\ell+1}(\mu).}$$

Conversely, the real computation has:

- an old name η,
- true counts $(\boldsymbol{x_\theta})_\theta$,
- a rounded attention output $\boldsymbol{u}$.

Thus some $\mathbf{Name}^{\mu}_{\ell+1}(i)$ holds.

Defining the next name

Let $\boldsymbol{u}$ be the attention output. Set

$$\boldsymbol{z = u + \tau_\ell(\eta).}$$

The rest is pointwise:

$$\mathbf{Upd}_\ell(\eta, u) = \boldsymbol{W_2}\mathbf{ReLU}(\boldsymbol{W_1 z + b_1}) + \boldsymbol{b_2} + \boldsymbol{z}.$$

Let $\boldsymbol{N_{\ell+1}}$ contain one name per possible value of $\mathbf{Upd}_\ell(\eta, \boldsymbol{u})$, with

$$\boldsymbol{\tau_{\ell+1}(\mu) = \mathbf{Upd}_\ell(\eta, u).}$$

Define $\mathbf{Name}^{\mu}_{\ell+1}(i)$ by choosing $\eta, \boldsymbol{u}, \bar{\boldsymbol{x}}$ such that:

- $\mathbf{Name}^{\eta}_{\ell}(i)$;
- each $\boldsymbol{x_\theta}$ counts $\mathbf{Name}^{\theta}_{\ell}$-positions;
- $\mathbf{Att}_\ell(\eta, \bar{\boldsymbol{x}}) = \boldsymbol{u}$;
- $\mathbf{Upd}_\ell(\eta, \boldsymbol{u}) = \boldsymbol{\tau_{\ell+1}(\mu)}$.

This is an $\mathbf{FOC}[\mathbf{MOD}, +]$-formula: only the names are counted.

If $\mathbf{Name}^{\mu}_{\ell+1}(i)$ holds, the chosen counts give the true softmax average at $\boldsymbol{i}$.
Hence

$$\boldsymbol{y_i^{(\ell+1)} = \tau_{\ell+1}(\mu).}$$

Conversely, the real computation has:

- an old name η,
- true counts $(\boldsymbol{x_\theta})_\theta$,
- a rounded attention output $\boldsymbol{u}$.

Thus some $\mathbf{Name}^{\mu}_{\ell+1}(i)$ holds.

Therefore

$$\mathbf{Name}^{\mu}_{\ell+1}(i) \iff \boldsymbol{y_i^{(\ell+1)} = \tau_{\ell+1}(\mu).}$$

Defining the next name

Let $\boldsymbol{u}$ be the attention output. Set

$$\boldsymbol{z = u + \tau_\ell(\eta).}$$

The rest is pointwise:

$$\mathbf{Upd}_\ell(\eta, u) = \boldsymbol{W_2}\mathbf{ReLU}(\boldsymbol{W_1 z + b_1}) + \boldsymbol{b_2} + \boldsymbol{z}.$$

Let $\boldsymbol{N_{\ell+1}}$ contain one name per possible value of $\mathbf{Upd}_\ell(\eta, \boldsymbol{u})$, with

$$\boldsymbol{\tau_{\ell+1}(\mu) = \mathbf{Upd}_\ell(\eta, u).}$$

Define $\mathbf{Name}^{\mu}_{\ell+1}(i)$ by choosing $\eta, \boldsymbol{u}, \bar{\boldsymbol{x}}$ such that:

- $\mathbf{Name}^{\eta}_{\ell}(i)$;
- each $\boldsymbol{x_\theta}$ counts $\mathbf{Name}^{\theta}_{\ell}$-positions;
- $\mathbf{Att}_\ell(\eta, \bar{\boldsymbol{x}}) = \boldsymbol{u}$;
- $\mathbf{Upd}_\ell(\eta, \boldsymbol{u}) = \boldsymbol{\tau_{\ell+1}(\mu)}$.

This is an $\mathbf{FOC}[\mathbf{MOD}, +]$-formula: only the names are counted.

If $\mathbf{Name}^{\mu}_{\ell+1}(i)$ holds, the chosen counts give the true softmax average at $\boldsymbol{i}$.
Hence

$$\boldsymbol{y_i^{(\ell+1)} = \tau_{\ell+1}(\mu).}$$

Conversely, the real computation has:

- an old name η,
- true counts $(\boldsymbol{x_\theta})_\theta$,
- a rounded attention output $\boldsymbol{u}$.

Thus some $\mathbf{Name}^{\mu}_{\ell+1}(i)$ holds.

Therefore

$$\mathbf{Name}^{\mu}_{\ell+1}(i) \iff \boldsymbol{y_i^{(\ell+1)} = \tau_{\ell+1}(\mu).}$$

The induction step is complete.

Output

After L layers,

$$\mathbf{Name}_L^{\eta}(p) \iff y_p^{(L)} = \tau_L(\eta).$$

Output

After $\boldsymbol{L}$ layers,

$$\mathbf{Name}_L^{\eta}(p) \iff y_p^{(L)} = \tau_L(\eta).$$

Acceptance is a finite disjunction:

$$\bigvee_{\eta \in N_L^{\mathbf{acc}}} \mathbf{Name}_L^{\eta}(p_{\mathbf{out}}).$$

Output

After L layers,

$$\mathbf{Name}_L^{\eta}(p) \iff y_p^{(L)} = \tau_L(\eta).$$

Acceptance is a finite disjunction:

$$\bigvee_{\eta \in N_L^{\mathbf{acc}}} \mathbf{Name}_L^{\eta}(p_{\mathbf{out}}).$$

Thus the recognized language is definable in $\mathbf{FOC}[\mathbf{MOD}, +]$.

Output

After L layers,

$$\mathbf{Name}_L^{\eta}(p) \iff y_p^{(L)} = \tau_L(\eta).$$

Acceptance is a finite disjunction:

$$\bigvee_{\eta \in N_L^{\mathbf{acc}}} \mathbf{Name}_L^{\eta}(p_{\mathbf{out}}).$$

Thus the recognized language is definable in $\mathbf{FOC}[\mathbf{MOD}, +]$.

Theorem E.1.2

With finite precision:

$\mathbf{SMAT} \subseteq \mathbf{FOC}[\mathbf{MOD}, +]$.

A convenient logic

The temporal counting logic $K_t[\#]$

We introduce a one-variable temporal logic with a counting operator. A formula is evaluated at a position of a word:

$$w = w_1 \cdots w_n, \qquad i \in \{1, \ldots, n\}.$$

The temporal counting logic $K_t[\#]$

We introduce a one-variable temporal logic with a counting operator. A formula is evaluated at a position of a word:

$$w = w_1 \cdots w_n, \qquad i \in \{1, \ldots, n\}.$$

$K_t[\#]$

For a finite alphabet Σ, formulas F and count terms C are:

$$F ::= Q_a \mid \neg F \mid F \wedge F \mid C \leq C \mid \top,$$
$$C ::= \#[F] \mid C + C \mid C - C \mid 1,$$

where $a \in \Sigma$.

The temporal counting logic $K_t[\#]$

We introduce a one-variable temporal logic with a counting operator. A formula is evaluated at a position of a word:

$$w = w_1 \cdots w_n, \qquad i \in \{1, \ldots, n\}.$$

$K_t[\#]$

For a finite alphabet Σ, formulas F and count terms C are:

$$F ::= Q_a \mid \neg F \mid F \wedge F \mid C \leq C \mid \top,$$
$$C ::= \#[F] \mid C + C \mid C - C \mid 1,$$

where $a \in \Sigma$.

The operator $\#[F]$ counts how many earlier positions satisfy F:

$$\#[F]^{w,i} = \big|\{j \leq i : w, j \models F\}\big|.$$

Thus every count is a **prefix count**.

The temporal counting logic $K_t[\#]$

We introduce a one-variable temporal logic with a counting operator. A formula is evaluated at a position of a word:

$$w = w_1 \cdots w_n, \qquad i \in \{1, \ldots, n\}.$$

$K_t[\#]$

For a finite alphabet Σ, formulas F and count terms C are:

$$F ::= Q_a \mid \neg F \mid F \wedge F \mid C \leq C \mid \top,$$
$$C ::= \#[F] \mid C + C \mid C - C \mid 1,$$

where $a \in \Sigma$.

The operator $\#[F]$ counts how many earlier positions satisfy F:

$$\#[F]^{w,i} = \big|\{j \leq i : w, j \models F\}\big|.$$

Thus every count is a **prefix count**.

A sentence defines a language by **end-satisfaction**:

$$w \in L(F) \iff w, n \models F.$$

So all examples are read at the last position!

Reading count terms

On the word $w = \texttt{abbab}$, the basic prefix counts are:

i	$\mathbf{1}$	$\mathbf{2}$	$\mathbf{3}$	$\mathbf{4}$	$\mathbf{5}$
w_i	a	b	b	a	b
$\#[Q_a](i)$	$\mathbf{1}$	$\mathbf{1}$	$\mathbf{1}$	$\mathbf{2}$	$\mathbf{2}$
$\#[Q_b](i)$	$\mathbf{0}$	$\mathbf{1}$	$\mathbf{2}$	$\mathbf{2}$	$\mathbf{3}$
$\#[\top](i)$	$\mathbf{1}$	$\mathbf{2}$	$\mathbf{3}$	$\mathbf{4}$	$\mathbf{5}$

Reading count terms

On the word $w = \texttt{abbab}$, the basic prefix counts are:

i	$\mathbf{1}$	$\mathbf{2}$	$\mathbf{3}$	$\mathbf{4}$	$\mathbf{5}$
w_i	a	b	b	a	b
$\#[Q_a](i)$	$\mathbf{1}$	$\mathbf{1}$	$\mathbf{1}$	$\mathbf{2}$	$\mathbf{2}$
$\#[Q_b](i)$	$\mathbf{0}$	$\mathbf{1}$	$\mathbf{2}$	$\mathbf{2}$	$\mathbf{3}$
$\#[\top](i)$	$\mathbf{1}$	$\mathbf{2}$	$\mathbf{3}$	$\mathbf{4}$	$\mathbf{5}$

Global counts

At the last position:

$$\#[Q_a] = \mathbf{2}, \qquad \#[Q_b] = \mathbf{3}, \qquad \#[\top] = \mathbf{5}.$$

Therefore

$$\#[Q_b] = \#[Q_a] + \mathbf{1}$$

is true on abbab.

Reading count terms

On the word $w = \texttt{abbab}$, the basic prefix counts are:

i	$\mathbf{1}$	$\mathbf{2}$	$\mathbf{3}$	$\mathbf{4}$	$\mathbf{5}$
w_i	a	b	b	a	b
$\#[Q_a](i)$	$\mathbf{1}$	$\mathbf{1}$	$\mathbf{1}$	$\mathbf{2}$	$\mathbf{2}$
$\#[Q_b](i)$	$\mathbf{0}$	$\mathbf{1}$	$\mathbf{2}$	$\mathbf{2}$	$\mathbf{3}$
$\#[\top](i)$	$\mathbf{1}$	$\mathbf{2}$	$\mathbf{3}$	$\mathbf{4}$	$\mathbf{5}$

Global counts

At the last position:

$$\#[Q_a] = \mathbf{2}, \qquad \#[Q_b] = \mathbf{3}, \qquad \#[\top] = \mathbf{5}.$$

Therefore

$$\#[Q_b] = \#[Q_a] + \mathbf{1}$$

is true on abbab.

A nested count

The term

$$\#[Q_b \wedge \#[Q_a] = \mathbf{1}]$$

counts the b-positions whose prefix contains exactly one a.

On abbab, this value is $\mathbf{2}$: the two middle b's are counted.

Reading count terms

On the word $w = \texttt{abbab}$, the basic prefix counts are:

i	$\mathbf{1}$	$\mathbf{2}$	$\mathbf{3}$	$\mathbf{4}$	$\mathbf{5}$
w_i	a	b	b	a	b
$\#[Q_a](i)$	$\mathbf{1}$	$\mathbf{1}$	$\mathbf{1}$	$\mathbf{2}$	$\mathbf{2}$
$\#[Q_b](i)$	$\mathbf{0}$	$\mathbf{1}$	$\mathbf{2}$	$\mathbf{2}$	$\mathbf{3}$
$\#[\top](i)$	$\mathbf{1}$	$\mathbf{2}$	$\mathbf{3}$	$\mathbf{4}$	$\mathbf{5}$

Global counts

At the last position:

$$\#[Q_a] = \mathbf{2}, \qquad \#[Q_b] = \mathbf{3}, \qquad \#[\top] = \mathbf{5}.$$

Therefore

$$\#[Q_b] = \#[Q_a] + \mathbf{1}$$

is true on abbab.

A nested count

The term

$$\#[Q_b \wedge \#[Q_a] = \mathbf{1}]$$

counts the b-positions whose prefix contains exactly one a.
On abbab, this value is $\mathbf{2}$: the two middle b's are counted.

The inner count is evaluated at each candidate position.
In

$$\#[Q_b \wedge \#[Q_a] = \mathbf{1}],$$

the test $\#[Q_a] = \mathbf{1}$ is evaluated at the candidate b-position, not at the final position.

Basic examples in $K_t[\#]$

Over $\Sigma = \{a, b\}$, the following are direct prefix-count tests.

Basic examples in $K_t[\#]$

Over $\Sigma = \{a, b\}$, the following are direct prefix-count tests.

Example

- Last symbol is a:

Basic examples in $K_t[\#]$

Over $\Sigma = \{a, b\}$, the following are direct prefix-count tests.

Example

- Last symbol is a:

  Q_a.

Basic examples in $K_t[\#]$

Over $\Sigma = \{a, b\}$, the following are direct prefix-count tests.

Example

- Last symbol is a:

  Q_a.
- The word contains an a:

Basic examples in $K_t[\#]$

Over $\Sigma = \{a, b\}$, the following are direct prefix-count tests.

Example

- Last symbol is a:

  Q_a.

- The word contains an a:

  $\#[Q_a] \geq 1$.

Basic examples in $K_t[\#]$

Over $\Sigma = \{a, b\}$, the following are direct prefix-count tests.

Example

- Last symbol is a:

  Q_a.
- The word contains an a:

  $\#[Q_a] \geq 1$.
- The length is exactly 5:

Basic examples in $K_t[\#]$

Over $\Sigma = \{a, b\}$, the following are direct prefix-count tests.

Example

- Last symbol is a:
  Q_a.
- The word contains an a:
  $\#[Q_a] \geq 1$.
- The length is exactly 5:
  $\#[\top] = 5$.

Basic examples in $K_t[\#]$

Over $\Sigma = \{a, b\}$, the following are direct prefix-count tests.

Example

- Last symbol is a:
  Q_a.
- The word contains an a:
  $\#[Q_a] \geq 1$.
- The length is exactly 5:
  $\#[\top] = 5$.
- There are exactly three a's:

Basic examples in $K_t[\#]$

Over $\Sigma = \{a, b\}$, the following are direct prefix-count tests.

Example

- Last symbol is a:
  Q_a.
- The word contains an a:
  $\#[Q_a] \geq 1$.
- The length is exactly 5:
  $\#[\top] = 5$.
- There are exactly three a's:
  $\#[Q_a] = 3$.

Basic examples in $K_t[\#]$

Over $\Sigma = \{a, b\}$, the following are direct prefix-count tests.

Example

- Last symbol is a:
  Q_a.
- The word contains an a:
  $\#[Q_a] \geq 1$.
- The length is exactly 5:
  $\#[\top] = 5$.
- There are exactly three a's:
  $\#[Q_a] = 3$.

Example

- Equal number of a's and b's:

Basic examples in $K_t[\#]$

Over $\Sigma = \{a, b\}$, the following are direct prefix-count tests.

Example

- Last symbol is a:
  Q_a.
- The word contains an a:
  $\#[Q_a] \geq 1$.
- The length is exactly 5:
  $\#[\top] = 5$.
- There are exactly three a's:
  $\#[Q_a] = 3$.

Example

- Equal number of a's and b's:
  $\#[Q_a] = \#[Q_b]$.

Basic examples in $K_t[\#]$

Over $\Sigma = \{a, b\}$, the following are direct prefix-count tests.

Example

- Last symbol is a:
  $$Q_a.$$
- The word contains an a:
  $$\#[Q_a] \geq 1.$$
- The length is exactly 5:
  $$\#[\top] = 5.$$
- There are exactly three a's:
  $$\#[Q_a] = 3.$$

Example

- Equal number of a's and b's:
  $$\#[Q_a] = \#[Q_b].$$
- Twice as many a's as b's:

Basic examples in $K_t[\#]$

Over $\Sigma = \{a, b\}$, the following are direct prefix-count tests.

Example

- Last symbol is a:
  $$Q_a.$$
- The word contains an a:
  $$\#[Q_a] \geq 1.$$
- The length is exactly 5:
  $$\#[\top] = 5.$$
- There are exactly three a's:
  $$\#[Q_a] = 3.$$

Example

- Equal number of a's and b's:
  $$\#[Q_a] = \#[Q_b].$$
- Twice as many a's as b's:
  $$\#[Q_a] = \#[Q_b] + \#[Q_b].$$

Basic examples in $K_t[\#]$

Over $\Sigma = \{a, b\}$, the following are direct prefix-count tests.

Example

- Last symbol is a:
  $$Q_a.$$
- The word contains an a:
  $$\#[Q_a] \geq 1.$$
- The length is exactly 5:
  $$\#[\top] = 5.$$
- There are exactly three a's:
  $$\#[Q_a] = 3.$$

Example

- Equal number of a's and b's:
  $$\#[Q_a] = \#[Q_b].$$
- Twice as many a's as b's:
  $$\#[Q_a] = \#[Q_b] + \#[Q_b].$$
- At least as many a's as b's:

Basic examples in $K_t[\#]$

Over $\Sigma = \{a, b\}$, the following are direct prefix-count tests.

Example

- Last symbol is a:
  $$Q_a.$$
- The word contains an a:
  $$\#[Q_a] \geq 1.$$
- The length is exactly 5:
  $$\#[\top] = 5.$$
- There are exactly three a's:
  $$\#[Q_a] = 3.$$

Example

- Equal number of a's and b's:
  $$\#[Q_a] = \#[Q_b].$$
- Twice as many a's as b's:
  $$\#[Q_a] = \#[Q_b] + \#[Q_b].$$
- At least as many a's as b's:
  $$\#[Q_b] \leq \#[Q_a].$$

Basic examples in $K_t[\#]$

Over $\Sigma = \{a, b\}$, the following are direct prefix-count tests.

Example

- Last symbol is a:
  $$Q_a.$$
- The word contains an a:
  $$\#[Q_a] \geq 1.$$
- The length is exactly 5:
  $$\#[\top] = 5.$$
- There are exactly three a's:
  $$\#[Q_a] = 3.$$

Example

- Equal number of a's and b's:
  $$\#[Q_a] = \#[Q_b].$$
- Twice as many a's as b's:
  $$\#[Q_a] = \#[Q_b] + \#[Q_b].$$
- At least as many a's as b's:
  $$\#[Q_b] \leq \#[Q_a].$$

We used some sugar:

$$C = D \equiv (C \leq D) \wedge (D \leq C),$$

and

$$C \geq D \equiv D \leq C.$$

Order-sensitive examples

Prefix counts can express that some bad pattern never appears.

Order-sensitive examples

Prefix counts can express that some bad pattern never appears.

The language a^*b^*

No a occurs after a b has appeared:

$$\#[Q_a \wedge \#[Q_b] \geq 1] = 0.$$

Order-sensitive examples

Prefix counts can express that some bad pattern never appears.

The language a^*b^*

No a occurs after a b has appeared:

$$\#[Q_a \wedge \#[Q_b] \geq 1] = 0.$$

The language a^+b^+

Add non-emptiness:

$$\begin{aligned} &\#[Q_a \wedge \#[Q_b] \geq 1] = 0 \\ &\quad \wedge\ \#[Q_a] \geq 1 \\ &\quad \wedge\ \#[Q_b] \geq 1. \end{aligned}$$

Order-sensitive examples

Prefix counts can express that some bad pattern never appears.

The language a^*b^*

No a occurs after a b has appeared:

$$\#[Q_a \wedge \#[Q_b] \geq 1] = 0.$$

The language a^+b^+

Add non-emptiness:

$$\#[Q_a \wedge \#[Q_b] \geq 1] = 0 \\ \wedge\ \#[Q_a] \geq 1 \\ \wedge\ \#[Q_b] \geq 1.$$

Every prefix has enough a's

Every prefix has at least as many a's as b's:

$$\#[\#[Q_b] > \#[Q_a]] = 0.$$

The outer count counts the violating prefixes.

Order-sensitive examples

Prefix counts can express that some bad pattern never appears.

The language a^*b^*

No a occurs after a b has appeared:

$$\#[Q_a \wedge \#[Q_b] \geq 1] = 0.$$

The language a^+b^+

Add non-emptiness:

$$\#[Q_a \wedge \#[Q_b] \geq 1] = 0 \\ \wedge\ \#[Q_a] \geq 1 \\ \wedge\ \#[Q_b] \geq 1.$$

Every prefix has enough a's

Every prefix has at least as many a's as b's:

$$\#[\#[Q_b] > \#[Q_a]] = 0.$$

The outer count counts the violating prefixes.

Dyck-1 over parentheses

Balanced parentheses are definable by:

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

Subsequences via nested counts

Nested #'s express subsequence witnesses.
For instance, define count terms:

$$\tau_a := \#[Q_a],$$
$$\tau_{ab} := \#[Q_b \wedge \tau_a \geq 1],$$
$$\tau_{aba} := \#[Q_a \wedge \tau_{ab} \geq 1].$$

Subsequences via nested counts

Nested #'s express subsequence witnesses. For instance, define count terms:

$$\tau_a := \#[Q_a],$$
$$\tau_{ab} := \#[Q_b \wedge \tau_a \geq 1],$$
$$\tau_{aba} := \#[Q_a \wedge \tau_{ab} \geq 1].$$

Subsequence aba

The word contains aba as a subsequence iff $\tau_{aba} \geq 1$.
Each nesting level extends the matched subsequence by one symbol.

Subsequences via nested counts

Nested #'s express subsequence witnesses. For instance, define count terms:

$$\tau_a := \#[Q_a],$$
$$\tau_{ab} := \#[Q_b \wedge \tau_a \geq 1],$$
$$\tau_{aba} := \#[Q_a \wedge \tau_{ab} \geq 1].$$

Subsequence aba

The word contains aba as a subsequence iff

$$\tau_{aba} \geq 1.$$

Each nesting level extends the matched subsequence by one symbol.

More generally, for

$$s = s_1 \cdots s_k,$$

define

$$\tau_1 := \#[Q_{s_1}],$$
$$\tau_{r+1} := \#[Q_{s_{r+1}} \wedge \tau_r \geq 1].$$

Then s occurs as a subsequence iff

$$\tau_k \geq 1.$$

Subsequences via nested counts

Nested #'s express subsequence witnesses. For instance, define count terms:

$$\tau_a := \#[Q_a],$$
$$\tau_{ab} := \#[Q_b \wedge \tau_a \geq 1],$$
$$\tau_{aba} := \#[Q_a \wedge \tau_{ab} \geq 1].$$

Subsequence aba

The word contains aba as a subsequence iff

$$\tau_{aba} \geq 1.$$

Each nesting level extends the matched subsequence by one symbol.

More generally, for

$$s = s_1 \cdots s_k,$$

define

$$\tau_1 := \#[Q_{s_1}],$$
$$\tau_{r+1} := \#[Q_{s_{r+1}} \wedge \tau_r \geq 1].$$

Then s occurs as a subsequence iff

$$\tau_k \geq 1.$$

Exact word `hello`

Length and last-symbol tests turn a subsequence pattern into an exact-word test:

$$\#[\top] = 5 \wedge Q_o \wedge \#[Q_l \wedge \#[Q_e \wedge \#[Q_h] = 1] = 1] = 2.$$

Non-regular and context-sensitive examples

The language $a^n b^n$

First force block order, then compare counts:

$$\#[Q_a \wedge \#[Q_b] \geq 1] = 0$$
$$\wedge \ \#[Q_a] = \#[Q_b].$$

Non-regular and context-sensitive examples

The language $a^n b^n$

First force block order, then compare counts:

$$\#[Q_a \wedge \#[Q_b] \geq 1] = 0$$
$$\wedge\ \#[Q_a] = \#[Q_b].$$

The language $a^n b^n c^n$

A compact way to enforce $a^* b^* c^*$ is to forbid ba, ca, and cb inversions:

$$\#[Q_a \wedge \#[Q_b \vee Q_c] \geq 1] = 0$$
$$\wedge\ \#[Q_b \wedge \#[Q_c] \geq 1] = 0$$
$$\wedge\ \#[Q_a] = \#[Q_b]$$
$$\wedge\ \#[Q_b] = \#[Q_c].$$

Non-regular and context-sensitive examples

The language $a^n b^n$

First force block order, then compare counts:

$$\#[Q_a \wedge \#[Q_b] \geq 1] = 0$$
$$\wedge\ \#[Q_a] = \#[Q_b].$$

The language $a^n b^n c^n$

A compact way to enforce $a^* b^* c^*$ is to forbid ba, ca, and cb inversions:

$$\#[Q_a \wedge \#[Q_b \vee Q_c] \geq 1] = 0$$
$$\wedge\ \#[Q_b \wedge \#[Q_c] \geq 1] = 0$$
$$\wedge\ \#[Q_a] = \#[Q_b]$$
$$\wedge\ \#[Q_b] = \#[Q_c].$$

These examples illustrate the two basic moves:

- **global arithmetic**: compare final prefix counts;
- **prefix safety**: count the positions where a bad prefix property holds.

Non-regular and context-sensitive examples

The language $a^n b^n$

First force block order, then compare counts:

$$\#[Q_a \wedge \#[Q_b] \geq 1] = 0 \\ \wedge\ \#[Q_a] = \#[Q_b].$$

The language $a^n b^n c^n$

A compact way to enforce $a^* b^* c^*$ is to forbid ba, ca, and cb inversions:

$$\#[Q_a \wedge \#[Q_b \vee Q_c] \geq 1] = 0 \\ \wedge\ \#[Q_b \wedge \#[Q_c] \geq 1] = 0 \\ \wedge\ \#[Q_a] = \#[Q_b] \\ \wedge\ \#[Q_b] = \#[Q_c].$$

These examples illustrate the two basic moves:

- **global arithmetic**: compare final prefix counts;
- **prefix safety**: count the positions where a bad prefix property holds.

Exactly one a before the first b

$$\#[Q_b] \geq 1 \\ \wedge\ \#[Q_a \wedge \#[Q_b] = 0] = 1.$$

The counted a-positions are precisely those before any b has appeared.

Link with transformers

The logic $\boldsymbol{K}_y[\#]$ is usefull to specify a language, and then be compiled to a transformer.

Theorem E.1. 3(Yang–Chiang)

Every $\boldsymbol{K}_t[\#]$ formula $\boldsymbol{F}$ can be simulated by a masked soft-attention transformer using logarithmic precision and no positional encodings.

Link with transformers

The logic $\boldsymbol{K}_y[\#]$ is usefull to specify a language, and then be compiled to a transformer.

Theorem E.1. 3(Yang–Chiang)

Every $\boldsymbol{K}_t[\#]$ formula $\boldsymbol{F}$ can be simulated by a masked soft-attention transformer using logarithmic precision and no positional encodings.

For finite precision, the reverse simulation holds.

Link with transformers

The logic $\boldsymbol{K}_y[\#]$ is usefull to specify a language, and then be compiled to a transformer.

Theorem E.1. 3(Yang–Chiang)

Every $\boldsymbol{K}_t[\#]$ formula $\boldsymbol{F}$ can be simulated by a masked soft-attention transformer using logarithmic precision and no positional encodings.

For finite precision, the reverse simulation holds.

Theorem E.1. 4(Yang–Chiang)

Fixed-precision future-masked soft-attention transformers **without** positional encodings are simulable in

$\boldsymbol{K}_t[\#]$.

Link with transformers

The logic $\boldsymbol{K_y}[\#]$ is usefull to specify a language, and then be compiled to a transformer.

Theorem E.1. 3(Yang–Chiang)

Every $\boldsymbol{K_t}[\#]$ formula $\boldsymbol{F}$ can be simulated by a masked soft-attention transformer using logarithmic precision and no positional encodings.

For finite precision, the reverse simulation holds.

Theorem E.1. 4(Yang–Chiang)

Fixed-precision future-masked soft-attention transformers **without** positional encodings are simulable in

$\boldsymbol{K_t}[\#]$.

Theorem E.1. 5(With rational sinusoidal PE)

Suppose the positional encodings are rational sinusoidals, built from finitely many coordinates of the form

$$i \longmapsto \sin(2\pi q i), \qquad i \longmapsto \cos(2\pi q i), \qquad q \in \mathbb{Q}.$$

Then the simulation uses

$$\boldsymbol{K_t}[\#; \mathbf{MOD}], \text{ where } w, i \models \mathbf{MOD}_m^k \iff i \equiv k \pmod{m}.$$

A depth hierarchy

The model and the depth measure

The model is **future-masked** and has **no positional encodings**.
For equivalence with $\boldsymbol{K}_t[\#]$, we need a middle-ground: **mixed precision**.
Most operations: finite precision.
Attention: the numerator and denominator of the softmax weighted average are arbitrary precision; final output is rounded.

The model and the depth measure

The model is **future-masked** and has **no positional encodings**.
For equivalence with $\boldsymbol{K_t}[\#]$, we need a middle-ground: **mixed precision**.
Most operations: finite precision.
Attention: the numerator and denominator of the softmax weighted average are arbitrary precision; final output is rounded.

Modal depth

Modal depth is the maximum nesting depth of $\#$:

$$\mathbf{md}(Q_a) = 0, \qquad \mathbf{md}(1) = 0,$$
$$\mathbf{md}(\neg F) = \mathbf{md}(F),$$
$$\mathbf{md}(\#[F]) = 1 + \mathbf{md}(F),$$
$$\mathbf{md}(F \wedge G) = \max\big(\mathbf{md}(F), \mathbf{md}(G)\big),$$
$$\mathbf{md}(C + D) = \max\big(\mathbf{md}(C), \mathbf{md}(D)\big).$$

The model and the depth measure

The model is **future-masked** and has **no positional encodings**.
For equivalence with $\boldsymbol{K_t[\#]}$, we need a middle-ground: **mixed precision**.
Most operations: finite precision.
Attention: the numerator and denominator of the softmax weighted average are arbitrary precision; final output is rounded.

Modal depth

Modal depth is the maximum nesting depth of $\#$:

$$\mathbf{md}(Q_a) = 0, \qquad \mathbf{md}(1) = 0,$$
$$\mathbf{md}(\neg F) = \mathbf{md}(F),$$
$$\mathbf{md}(\#[F]) = 1 + \mathbf{md}(F),$$
$$\mathbf{md}(F \wedge G) = \max\big(\mathbf{md}(F), \mathbf{md}(G)\big),$$
$$\mathbf{md}(C + D) = \max\big(\mathbf{md}(C), \mathbf{md}(D)\big).$$

Example

$\overleftarrow{\#}[Q_a] = \overleftarrow{\#}[Q_b]$
has depth $\mathbf{1}$.
$\overleftarrow{\#}[Q_b \wedge \overleftarrow{\#}[Q_a] \geq 1] \geq 1$
has depth $\mathbf{2}$.

Equivalence and strict hierarchy

Theorem E.1.6(Depth-preserving equivalence)

For every k, a language L is definable by a $\mathbf{TL}[\#]$ **formula of modal depth** k if and only if L is recognized by a **depth-**k **mixed-precision transformer**.

→ Admitted

Equivalence and strict hierarchy

Theorem E.1.6(Depth-preserving equivalence)

For every k, a language L is definable by a $\mathbf{TL}[\#]$ **formula of modal depth** k if and only if L is recognized by a **depth-k mixed-precision transformer**.

→ Admitted

Theorem E.1.7(Strict hierarchy)

For every $k > 0$, there is a language L_{k+1} such that:

- $L_{k+1} \in \mathbf{TL}[\#]_{k+1}$
- $L_{k+1} \notin \mathbf{TL}[\#]_k$

Hence depth-$(k+1)$ transformers can recognize L_{k+1}, but depth-k transformers cannot.

Equivalence and strict hierarchy

Theorem E.1.6(Depth-preserving equivalence)

For every k, a language L is definable by a $\mathbf{TL}[\#]$ **formula of modal depth** k if and only if L is recognized by a **depth-k mixed-precision transformer**.

→ Admitted

Theorem E.1.7(Strict hierarchy)

For every $k > 0$, there is a language L_{k+1} such that:

- $L_{k+1} \in \mathbf{TL}[\#]_{k+1}$
- $L_{k+1} \notin \mathbf{TL}[\#]_k$

Hence depth-$(k+1)$ transformers can recognize L_{k+1}, but depth-k transformers cannot.

Witness languages

Over $\Sigma = \{a, b\}$, let L_k be the language of exactly k nonempty alternating runs, starting with a:

$$L_k = \begin{cases} (a^+b^+)^{k/2}, & k \text{ even}, \\ (a^+b^+)^{(k-1)/2}a^+, & k \text{ odd}. \end{cases}$$

Equivalence and strict hierarchy

Theorem E.1.6(Depth-preserving equivalence)

For every k, a language L is definable by a $\mathbf{TL}[\#]$ **formula of modal depth** k if and only if L is recognized by a **depth-k mixed-precision transformer**.

→ Admitted

Theorem E.1.7(Strict hierarchy)

For every $k > 0$, there is a language L_{k+1} such that:

- $L_{k+1} \in \mathbf{TL}[\#]_{k+1}$
- $L_{k+1} \notin \mathbf{TL}[\#]_k$

Hence depth-$(k+1)$ transformers can recognize L_{k+1}, but depth-k transformers cannot.

Witness languages

Over $\Sigma = \{a, b\}$, let L_k be the language of exactly k nonempty alternating runs, starting with a:

$$L_k = \begin{cases} (a^+b^+)^{k/2}, & k \text{ even}, \\ (a^+b^+)^{(k-1)/2}a^+, & k \text{ odd}. \end{cases}$$

Example

$$\begin{aligned} L_1 &= a^+, \\ L_2 &= a^+b^+, \\ L_3 &= a^+b^+a^+, \\ L_4 &= a^+b^+a^+b^+. \end{aligned}$$

Each extra layer buys one more alternation of runs.

Upper bound I: piecewise-testable languages

J-expression

A r-subsequence pattern is

$$\Sigma^* \sigma_1 \Sigma^* \sigma_2 \Sigma^* \cdots \Sigma^* \sigma_r \Sigma^*.$$

A language is r-piecewise testable if it is a Boolean combination of r-subsequence patterns.

Upper bound I: piecewise-testable languages

J-expression

A r-subsequence pattern is

$$\Sigma^* \sigma_1 \Sigma^* \sigma_2 \Sigma^* \cdots \Sigma^* \sigma_r \Sigma^*.$$

A language is r-piecewise testable if it is a Boolean combination of r-subsequence patterns.

Historically: without dependency on r, the set of piecewise testable languages corresponds to the set of FO formulas with alternation. That is:

- There are no $\forall$ in the scope of an $\exists$,
- There are no $\exists$ in the scope of an $\forall$.

Upper bound I: piecewise-testable languages

J-expression

A r-subsequence pattern is

$$\Sigma^* \sigma_1 \Sigma^* \sigma_2 \Sigma^* \cdots \Sigma^* \sigma_r \Sigma^*.$$

A language is r-piecewise testable if it is a Boolean combination of r-subsequence patterns.

Historically: without dependency on r, the set of piecewise testable languages corresponds to the set of FO formulas with alternation. That is:

- There are no $\forall$ in the scope of an $\exists$,
- There are no $\exists$ in the scope of an $\forall$.

Formulas

Define terms $\tau_{\sigma_1 \cdots \sigma_j}$ by

$$\tau_{\sigma_1} := \overleftarrow{\#}[Q_{\sigma_1}],$$

$$\tau_{\sigma_1 \cdots \sigma_j} := \overleftarrow{\#}[Q_{\sigma_j} \wedge \tau_{\sigma_1 \cdots \sigma_{j-1}} \geq 1]$$

Upper bound I: piecewise-testable languages

J-expression

A r-subsequence pattern is

$$\Sigma^* \sigma_1 \Sigma^* \sigma_2 \Sigma^* \cdots \Sigma^* \sigma_r \Sigma^*.$$

A language is r-piecewise testable if it is a Boolean combination of r-subsequence patterns.

Historically: without dependency on r, the set of piecewise testable languages corresponds to the set of FO formulas with alternation. That is:

- There are no $\forall$ in the scope of an $\exists$,
- There are no $\exists$ in the scope of an $\forall$.

Formulas

Define terms $\tau_{\sigma_1 \cdots \sigma_j}$ by

$$\tau_{\sigma_1} := \overleftarrow{\#}[Q_{\sigma_1}],$$

$$\tau_{\sigma_1 \cdots \sigma_j} := \overleftarrow{\#}[Q_{\sigma_j} \wedge \tau_{\sigma_1 \cdots \sigma_{j-1}} \geq 1]$$

The formula $\tau_{\sigma_1 \cdots \sigma_j}$ defines the subsequence pattern associated to $\sigma_1 \cdots \sigma_j$.

Upper bound I: piecewise-testable languages

J-expression

A r-subsequence pattern is

$$\Sigma^* \sigma_1 \Sigma^* \sigma_2 \Sigma^* \cdots \Sigma^* \sigma_r \Sigma^*.$$

A language is r-piecewise testable if it is a Boolean combination of r-subsequence patterns.

Historically: without dependency on r, the set of piecewise testable languages corresponds to the set of FO formulas with alternation. That is:

- There are no $\forall$ in the scope of an $\exists$,
- There are no $\exists$ in the scope of an $\forall$.

Formulas

Define terms $\tau_{\sigma_1 \cdots \sigma_j}$ by

$$\tau_{\sigma_1} := \overleftarrow{\#}[Q_{\sigma_1}],$$

$$\tau_{\sigma_1 \cdots \sigma_j} := \overleftarrow{\#}[Q_{\sigma_j} \wedge \tau_{\sigma_1 \cdots \sigma_{j-1}} \geq 1]$$

The formula $\tau_{\sigma_1 \cdots \sigma_j}$ defines the subsequence pattern associated to $\sigma_1 \cdots \sigma_j$.

Example

The J-expression

$$\Sigma^* a \Sigma^* b \Sigma^* a \Sigma^*$$

is defined by

$$\overleftarrow{\#}[Q_a \wedge \overleftarrow{\#}[Q_b \wedge \overleftarrow{\#}[Q_a] \geq 1] \geq 1] \geq 1.$$

This has modal depth 3.

Upper bound II: the witnesses are piecewise testable

Definition

$$A_k = \begin{cases} \Sigma^*(a\Sigma^* b\Sigma^*)^{k/2}, & k \text{ even}, \\ \Sigma^*(a\Sigma^* b\Sigma^*)^{(k-1)/2} a\Sigma^*, & k \text{ odd}, \end{cases}$$

$$B_k = \begin{cases} \Sigma^*(b\Sigma^* a\Sigma^*)^{k/2}, & k \text{ even}, \\ \Sigma^*(b\Sigma^* a\Sigma^*)^{(k-1)/2} b\Sigma^*, & k \text{ odd}. \end{cases}$$

Upper bound II: the witnesses are piecewise testable

Definition

$$A_k = \begin{cases} \Sigma^*(a\Sigma^* b\Sigma^*)^{k/2}, & k \text{ even}, \\ \Sigma^*(a\Sigma^* b\Sigma^*)^{(k-1)/2} a\Sigma^*, & k \text{ odd}, \end{cases}$$

$$B_k = \begin{cases} \Sigma^*(b\Sigma^* a\Sigma^*)^{k/2}, & k \text{ even}, \\ \Sigma^*(b\Sigma^* a\Sigma^*)^{(k-1)/2} b\Sigma^*, & k \text{ odd}. \end{cases}$$

A_k requires the alternating subsequence that starts with a;
B_k requires the opposite alternating subsequence that starts with b.

Upper bound II: the witnesses are piecewise testable

Definition

$$A_k = \begin{cases} \Sigma^*(a\Sigma^* b\Sigma^*)^{k/2}, & k \text{ even}, \\ \Sigma^*(a\Sigma^* b\Sigma^*)^{(k-1)/2} a\Sigma^*, & k \text{ odd}, \end{cases}$$

$$B_k = \begin{cases} \Sigma^*(b\Sigma^* a\Sigma^*)^{k/2}, & k \text{ even}, \\ \Sigma^*(b\Sigma^* a\Sigma^*)^{(k-1)/2} b\Sigma^*, & k \text{ odd}. \end{cases}$$

A_k requires the alternating subsequence that starts with a;
B_k requires the opposite alternating subsequence that starts with b.

Theorem E.1.8(Upper bound)

$L_k = A_k \cap \overline{B_k}$.

Therefore L_k is k-piecewise testable, hence $L_k \in \mathbf{TL}[\#]_k$.

Upper bound II: the witnesses are piecewise testable

Definition

$$A_k = \begin{cases} \Sigma^*(a\Sigma^*b\Sigma^*)^{k/2}, & k \text{ even}, \\ \Sigma^*(a\Sigma^*b\Sigma^*)^{(k-1)/2}a\Sigma^*, & k \text{ odd}, \end{cases}$$

$$B_k = \begin{cases} \Sigma^*(b\Sigma^*a\Sigma^*)^{k/2}, & k \text{ even}, \\ \Sigma^*(b\Sigma^*a\Sigma^*)^{(k-1)/2}b\Sigma^*, & k \text{ odd}. \end{cases}$$

A_k requires the alternating subsequence that starts with a;
B_k requires the opposite alternating subsequence that starts with b.

Theorem E.1.8(Upper bound)

$L_k = A_k \cap \overline{B_k}$.

Therefore L_k is k-piecewise testable, hence

$L_k \in \mathbf{TL}[\#]_k$.

For $L_3 = a^+b^+a^+$

$A_3 = \Sigma^*a\Sigma^*b\Sigma^*a\Sigma^*$,

$B_3 = \Sigma^*b\Sigma^*a\Sigma^*b\Sigma^*$.

Thus L_3 consists of the words that contain an aba subsequence but contain no bab subsequence.

Why $a^+b^+a^+$ needs depth 3

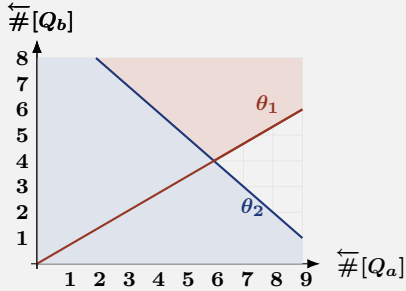
Goal: $a^+b^+a^+ \notin \mathbf{TL}[\#]_2$.

A depth-2 formula with depth-1 tests $\theta_1 := 2\overleftarrow{\#}[Q_a] \leq 3\overleftarrow{\#}[Q_b], \theta_2 := \overleftarrow{\#}[Q_a] + \overleftarrow{\#}[Q_b] \leq 10$.

Why $a^+b^+a^+$ needs depth 3

Goal: $a^+b^+a^+ \notin \mathbf{TL}[\#]_2$.

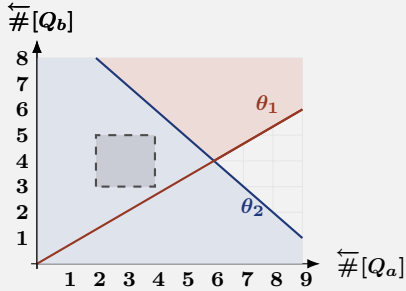
A depth-2 formula with depth-1 tests $\theta_1 := 2\overleftarrow{\#}[Q_a] \leq 3\overleftarrow{\#}[Q_b], \theta_2 := \overleftarrow{\#}[Q_a] + \overleftarrow{\#}[Q_b] \leq 10$.



Why $a^+b^+a^+$ needs depth 3

Goal: $a^+b^+a^+ \notin \mathbf{TL}[\#]_2$.

A depth-2 formula with depth-1 tests $\theta_1 := 2\overleftarrow{\#}[Q_a] \leq 3\overleftarrow{\#}[Q_b], \theta_2 := \overleftarrow{\#}[Q_a] + \overleftarrow{\#}[Q_b] \leq 10$.

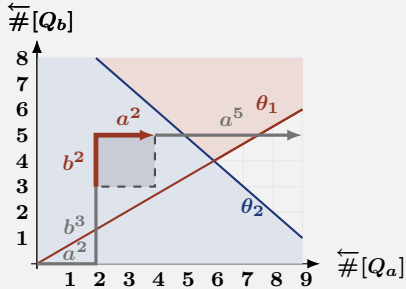


Inside the rectangle, all depth-1 subformulas are constant.
So the depth-2 formula locally behaves like depth 1: it sees counts of letters but not their order.

Why $a^+b^+a^+$ needs depth 3

Goal: $a^+b^+a^+ \notin \mathbf{TL}[\#]_2$.

A depth-2 formula with depth-1 tests $\theta_1 := 2\overleftarrow{\#}[Q_a] \leq 3\overleftarrow{\#}[Q_b], \theta_2 := \overleftarrow{\#}[Q_a] + \overleftarrow{\#}[Q_b] \leq 10$.



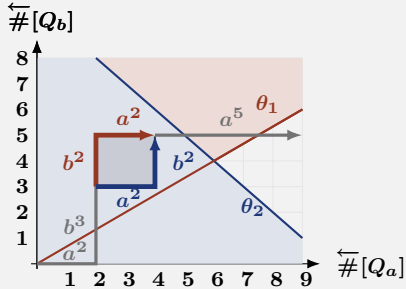
Inside the rectangle, all depth-1 subformulas are constant.
So the depth-2 formula locally behaves like depth 1: it sees counts of letters but not their order.

There is a word in $a^+b^+a^+$ with a large intersection.

Why $a^+b^+a^+$ needs depth 3

Goal: $a^+b^+a^+ \notin \mathbf{TL}[\#]_2$.

A depth-2 formula with depth-1 tests $\theta_1 := 2\overleftarrow{\#}[Q_a] \leq 3\overleftarrow{\#}[Q_b], \theta_2 := \overleftarrow{\#}[Q_a] + \overleftarrow{\#}[Q_b] \leq 10$.



Inside the rectangle, all depth-1 subformulas are constant.
So the depth-2 formula locally behaves like depth 1: it sees counts of letters but not their order.

There is a word in $a^+b^+a^+$ with a large intersection.

The formula cannot distinguish b^2a^2 from a^2b^2.
Switching them changes membership in $a^+b^+a^+$.

Experimental validation

Accuracy on $[351, 400]$

	depth → 1	2	3	4	5	6	7	8	9	10
L_3	100	100	100	100	100	100	100	100	100	100
L_4	22	100	100	100	100	100	100	100	100	100
L_5	11	51	100	100	100	100	100	100	100	100
L_6	10	8	37	100	98	100	100	100	100	100
L_7	8	8	20	49	100	100	100	100	98	100
L_8	5	6	19	52	75	95	100	100	100	100
L_9	7	7	6	24	29	66	95	99	98	93
L_{10}	2	2	2	6	12	49	77	96	96	98
L_{11}	2	2	3	3	6	11	46	31	96	89
L_{12}	0	0	0	1	2	4	21	30	27	62

The experiments use future-masked transformers **without positional encodings** on the L_k tasks.

The training-length bin is

$$[351, 400].$$

The predicted threshold is the black staircase.
A depth-d model should solve L_{d+2}, but not L_{d+3}.

The observed transition closely follows the theoretical boundary: below it, accuracies are low; above it, they are usually near perfect.

What can transformers learn?

A good candidate

Definition

A transformer T **length-generalize** a language L if T can be trained to learn L on **small instances**, and be correct on **larger instances**.

A good candidate

Definition

A transformer T **length-generalize** a language L if T can be trained to learn L on **small instances**, and be correct on **larger instances**.

There is a theoretical model: **limit transformers**.

A good candidate

Definition

A transformer T **length-generalize** a language L if T can be trained to learn L on **small instances**, and be correct on **larger instances**.

There is a theoretical model: **limit transformers**.

Empirically, length-generalizable languages seems to be precisely $\boldsymbol{K_t}[\#]$.

A good candidate

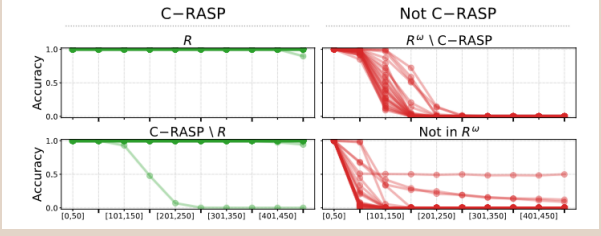
Definition

A transformer T **length-generalize** a language L if T can be trained to learn L on **small instances**, and be correct on **larger instances**.

There is a theoretical model: **limit transformers**.

Empirically, length-generalizable languages seems to be precisely $K_t[\#]$.

Training on words fo size 50:



A good candidate

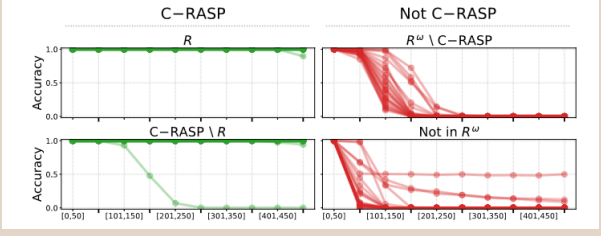
Definition

A transformer T **length-generalize** a language L if T can be trained to learn L on **small instances**, and be correct on **larger instances**.

There is a theoretical model: **limit transformers**.

Empirically, length-generalizable languages seems to be precisely $K_t[\#]$.

Training on words fo size 50:



The regular languages of $K_t[\#]$ are decidable.

Conclusion

SMAT seem more interesting that UHAT!

Conclusion

SMAT seem more interesting that UHAT!

Logic can be used to understand SMAT and transformers in general:

- To bound the expressivity,
- To have a programming language that can be compiled.

Conclusion

SMAT seem more interesting that UHAT!

Logic can be used to understand SMAT and transformers in general:

- To bound the expressivity,
- To have a programming language that can be compiled.

Capturing learning behaviour of transfomers need new insights form theory.

References

- This chapter is based to a some extent on the lecture "Expressivity of Transformers: Logic, Circuits, and Formal Languages" given at ESSLI 2024

See also:

- David Chiang, Peter Cholak, and Anand Pillay. Tighter bounds on the expressivity of transformer encoders. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 5544–5562. PMLR, 2023
- Andy Yang and David Chiang. Counting like transformers: Compiling temporal counting logic into softmax transformers. *CoRR*, abs/2404.04393, 2024
- Andy Yang, Michaël Cadilhac, and David Chiang. Knee-deep in C-RASP: A transformer depth hierarchy. *CoRR*, abs/2506.16055, 2025