Transformers, parallel computation, and logarithmic depth

Daniel Hsu (Columbia)

Joint work with:
Clayton Sanford (Columbia)
Matus Telgarsky (NYU)

Context

What do we know about Transformers (TFs) [Vaswani et al, 2017]?

- TFs are universal approximators
  [Yun et al, 2020; Pérez et al, 2021; Strobl et al, 2024; …]
- Many limitations of constant size/depth TFs
  [Hahn, 2020; Merrill & Sabharwal, 2022; Sanford et al, 2023; …]

What distinguishes TFs from other neural architectures?

Self-attention and transformers

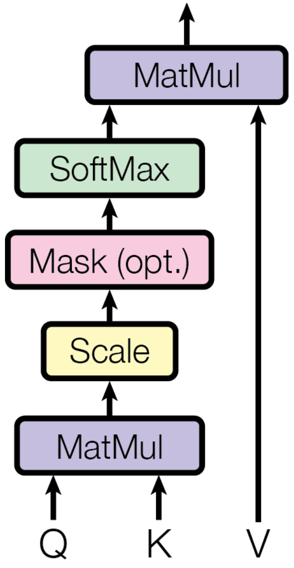
- Self-attention head:

$$\mathrm{SA}^{Q,K,V}(x_1, \ldots, x_N) = \sum_{j=1}^{N} \alpha_{i,j} V(x_j)$$

  where

$$\alpha_i = \mathrm{softmax}\big(Q(x_i) \cdot K(x_1), \ldots, Q(x_i) \cdot K(x_N)\big)$$

  - Embedding functions Q, K, V have embedding dimension m



- Self-attention layer: sum of H self-attention heads (width)
- Transformer: composition of L self-attention layers (depth)
- **This work**: $\log N$ precision numbers, $\mathrm{poly}(N)$ size alphabets, etc.

What we do

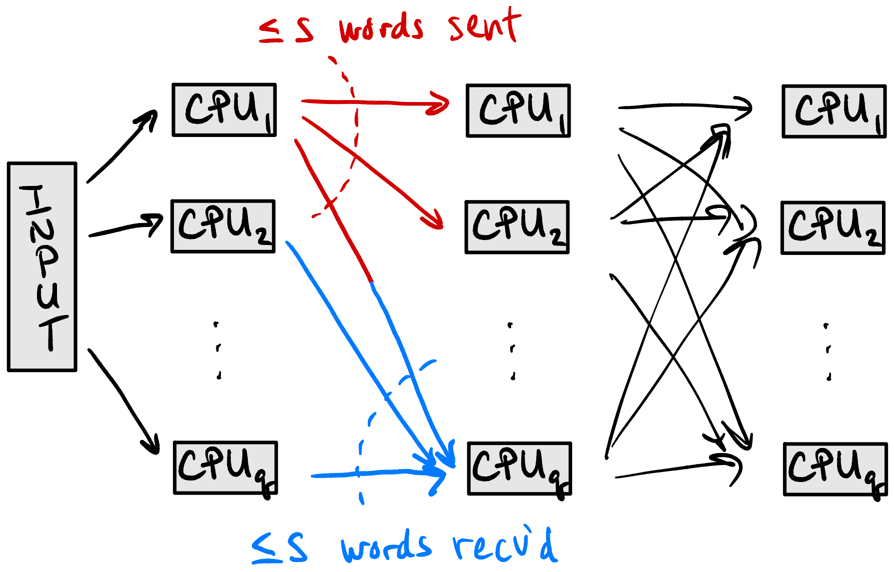
Goal: Use parallelism to distinguish TFs from other architectures

- **Part I** Relate TFs to Massively Parallel Computation
- **Part II** Distinguish TFs using "k-hop induction heads"

Part I: MPC vs TFs

Massively Parallel Computation (MPC)

- Culmination of theoretical models to study MapReduce, Hadoop, etc.
  [Karloff et al, 2010; Goodrich et al, 2011; Beame et al, 2013; Andoni et al, 2014]
  - Input size: n [$n \leq q \times s$]
  - Number of machines: q
  - Memory size per machine: s [$s = \Theta(n^{\delta})$ for small $\delta \in (0,1)$]



How many rounds R are needed?

MPC algorithms for many problems

- Broadcast $R = O(1)$
- Sorting $R = O(1)$
- Prefix sum $R = O(1)$
- Problems on sparse graphs [Andoni et al, 2018, Behnezhad et al, 2019, …]
  - Connected components $R = \log(\text{Diameter})$
  - Minimum spanning forest $R = \log(\text{Diameter})$

  …

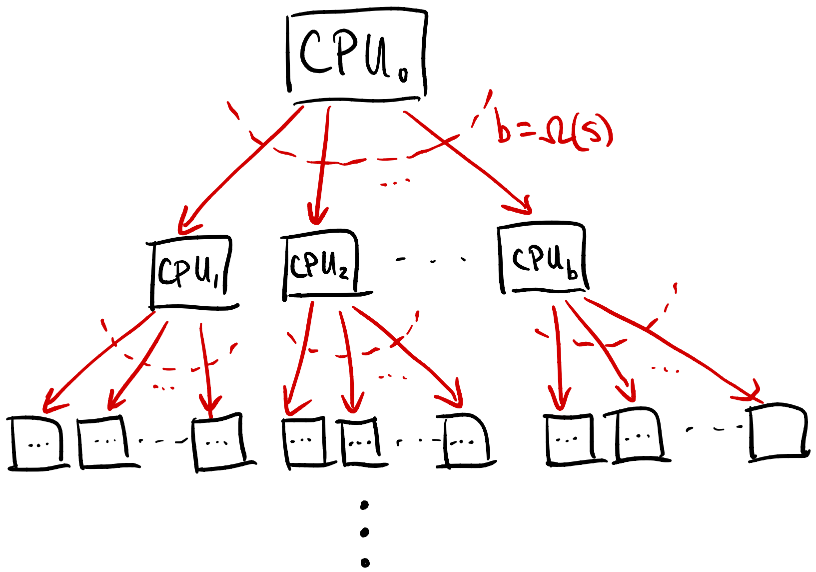
…

- **Open question**: $o(\log n)$ round algorithm for connectivity?

Example: MPC algorithm to broadcast a word

$s = \Theta(n^\delta)$, $q = \text{poly}(n)$

Propagate word using
$b = \Omega(s)$-ary broadcast tree



- # Rounds: $R = O\left(\frac{\log q}{\log s}\right) = O\left(\frac{1}{\delta}\right)$

Two very deep thoughts

1. If TFs can simulate MPC algorithms efficiently, then an efficient MPC algorithm implies a small TF
2. If MPC algorithms can simulate TFs efficiently, then problems hard for MPC are also hard for TFs

TFs can simulate MPC algorithms

- **Theorem** [SHT'24]: If $f : \Sigma^n \to \Sigma^n$ can be computed by R-round MPC algorithm using $q = \Theta\left(n^{1-\delta}\right)$ machines and $s = \Theta\left(n^{\delta}\right)$ word memory/machine, then f can be computed by TF with
  - $L = O(R)$ layers
  - $H = O(\log \log n)$ heads/layer
  - Embedding dimension $m = O\left(n^{4\delta} \log n\right)$
- **Corollary**: $\log(\text{Diameter})$-layer TF for connectivity in sparse graphs, …

Two very deep thoughts

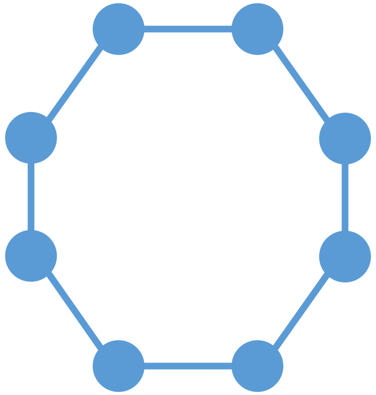
1. If TFs can simulate MPC algorithms efficiently, then an efficient MPC algorithm implies a small TF
2. If MPC algorithms can simulate TFs efficiently, then problems hard for MPC are also hard for TFs

MPC algorithms can simulate TFs

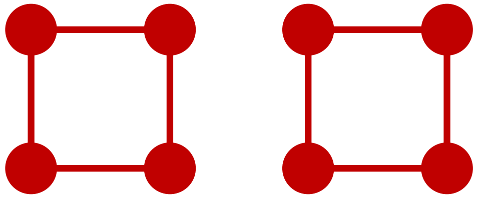
- **Theorem** [SHT'24]: If $f: \Sigma^N \to \Sigma^N$ can be computed by TF with L layers, H heads/layer, and embedding dimension m satisfying $mH = O(N^\delta)$, then for any $\gamma > 0$, f can be computed by MPC algorithm with
  - $R = O(L/\gamma)$ rounds
  - $q = O(N^2)$ machines
  - $s = O(N^{\delta+\gamma})$ word memory/machine

What problems are hard for MPC?

- **1-vs-2 cycle problem**: Given graph G that is promised to be either cycle on n vertices or union of two cycles on $n/2$ vertices each,



versus



decide if G is connected.

- **1-vs-2 cycle hypothesis**: All MPC algorithms for this problem with $s = O(n^{1-\epsilon})$ for some $\epsilon > 0$ and $q = \text{poly}(n)$ use $R = \Omega(\log n)$ rounds

Logarithmic depth is necessary for TFs

- **Corollary**: Assuming 1-vs-2 cycle hypothesis, every TF with $mH = O(n^{1-\epsilon})$ for some $\epsilon > 0$ that decides connectivity has $L = \Omega(\log n)$

Summary of Part I

- Efficient MPC algorithms give small TFs
- TFs face same limitations as MPC algorithms

Part II: k-hop induction heads

Induction heads

- Induction heads [Olsson et al, 2022] identified in existing pre-trained TFs solve a certain next-token prediction task
  - Given baebcabebdea, what comes next?
  - Answer: b

Multi-step induction heads task ("k-hop")

- Given baebcabebdea, what comes next?

baebcabebdea

- Answer ($k = 2$): c
- Multi-step reasoning problem [Peng, Narayanan, Papadimitriou, 2024]:
  - Prompt: "Jane is a teacher. Helen is a doctor. […] The mother of John is Helen. The mother of Charlotte is Eve. […] What's the profession of John's mother?"
  - Answer: doctor

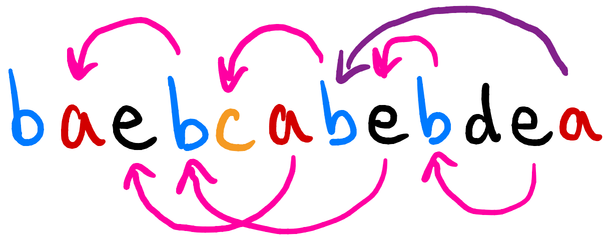
Why is k-hop important?

- Captures natural + simple multi-step reasoning problem
- TFs can compute it efficiently
- Non-parallel architectures (e.g., RNNs) have difficulty with it

TFs can efficiently compute k-hop predictions

- **Theorem** [SHT'24]: For any $k \in \mathbb{N}$, there is a causally-masked TF with
$$m = O(1), \qquad H = 1, \qquad L \leq 2 + \log_2(k)$$
that computes k-hop predictions (at all positions)

- Solution exploits parallelism in manner similar to [Bietti et al, 2023]

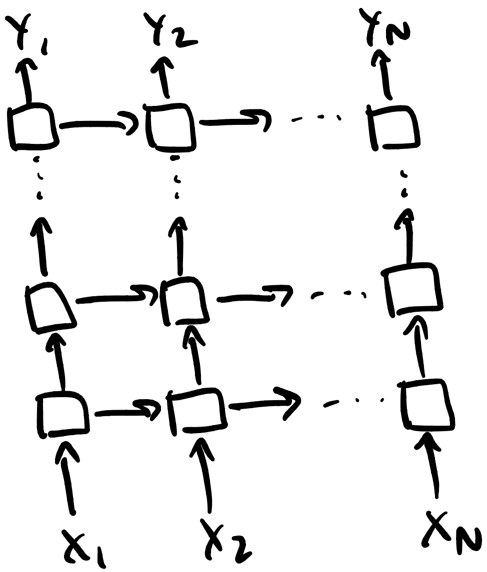


Every layer doubles the "reach"

- **Surprise**: SGD empirically appears to find the same solution!

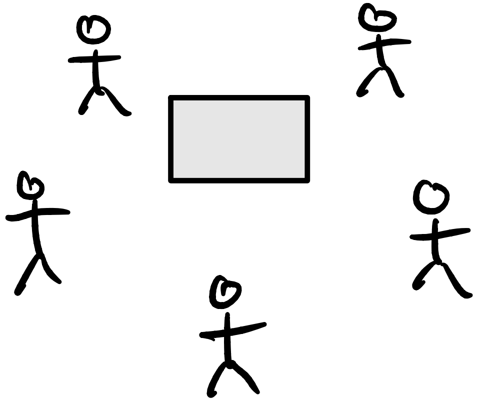
Bottleneck for non-parallel models

Small-state (multi-layer) RNNs



$\Rightarrow$

Efficient sequential k-party communication protocols



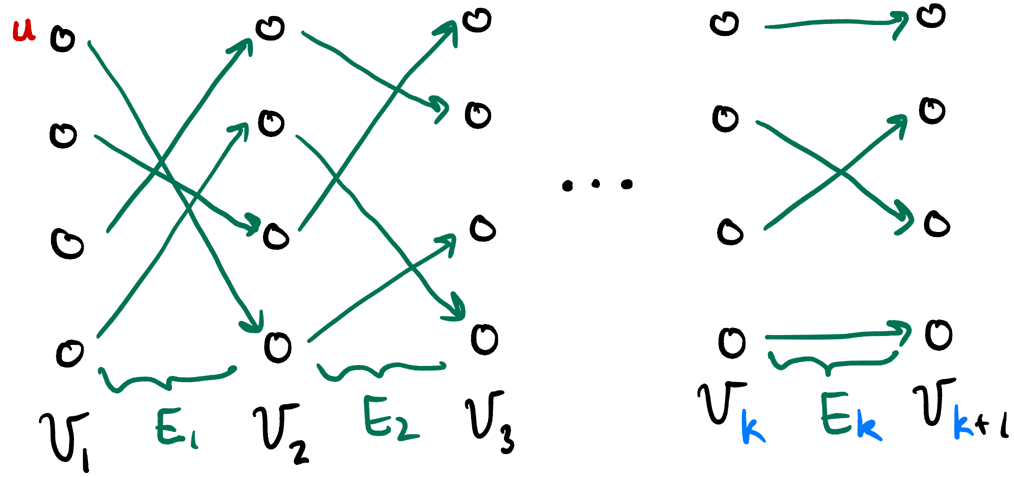
But k-hop is hard in this communication model

(Consequence of [Assadi and N, 2021])

Pointer Chasing [Nisan & Wigderson, 1993]

- **Problem**: Given k-layered graph $(\mathcal{V}_1, \dots, \mathcal{V}_{k+1}, E_1, \dots, E_k)$ and $u \in \mathcal{V}_1$, determine unique $v \in \mathcal{V}_{k+1}$ such that $u \rightsquigarrow v$

$\frac{n}{k+1}$ vertices per layer



E_i is perfect matching between $\mathcal{V}_i$ & $\mathcal{V}_{i+1}$

- **Proposition** [SHT'24]: Can encode $(E_k, \dots, E_2, E_1)$ and $u \in \mathcal{V}_1$ as $x \in \left[\frac{2n}{k+1}\right]^N$ $(N = \Theta(n))$ s.t. k-Pointer Chasing is equivalent to k-hop on x

Consequences of [Assadi & N, 2021]

Corollary: Average case lower bounds for computing k-hop predictions

- L-layer RNN (e.g., Mamba) with s-bit hidden state:
$$L \geq k \text{ or } s = \widetilde{\Omega}(n/k^6)$$
- TF using rank-r SA approximation:
$$L \geq k \text{ or } mHr = \widetilde{\Omega}(n/k^6)$$
- Single SA layer with T "chain-of-thought" tokens:
$$T \geq k \text{ or } mH = \widetilde{\Omega}(n/k^6)$$
- …

Summary of Part II

k-hop induction heads task

- Captures natural and simple multi-step reasoning problem
- Can be solved by TFs with $O(\log k)$ depth and $O(1)$ width
  - (This depth is necessary, assuming 1-vs-2 cycle hypothesis)
- Cannot be solved by other "non-parallel" architectures unless they have $\Omega(k)$ "depth" or $\Omega(n/k^6)$ "size"

Closing

- Parallelism distinguishes TFs from other architectures
  - Relies on log depth + sublinear width regime for TF
  - Separation exhibited by natural multi-step reasoning problem
- Future work
  - Finer-grain understanding of TFs that looks inside embedding functions
  - Learning

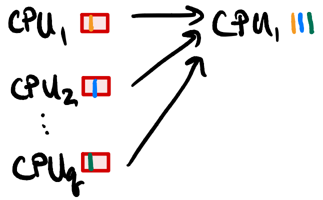
Thank you!

arXiv:2402.09268, to appear @ ICML 2024

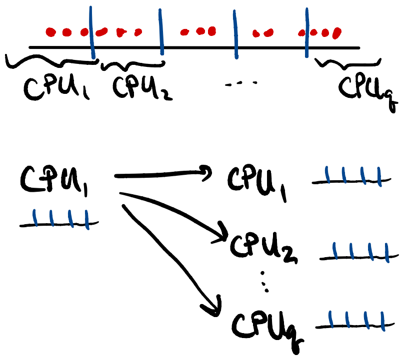
Example: MPC algorithm for sorting

$s = \Theta(n^{2/3}), q = \Theta(n^{1/3})$

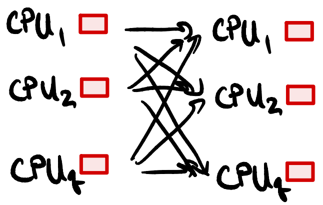
1. Each machine marks each of its elements with probability $\Theta(s/n)$, then send marked elements to Machine 1



2. Machine 1 determines q "ranges" that partition inputs (approx.) evenly; broadcast specs to all machines



3. Each machine collects input elements in "range" it is responsible for, then sort elements locally



Key idea: self-attention head for routing

- Messages to be sent (received) by machine i (machine j):

$$\mathrm{Outbox}_i \subseteq \Sigma \times [q], \qquad \mathrm{Inbox}_j = \{(\mathrm{msg}, i) : (\mathrm{msg}, j) \in \mathrm{Outbox}_i\}$$

- MPC algorithm guarantees $|\mathrm{Outbox}_i| = O(s)$ and $|\mathrm{Inbox}_j| = O(s)$
- We design a small SA head such that

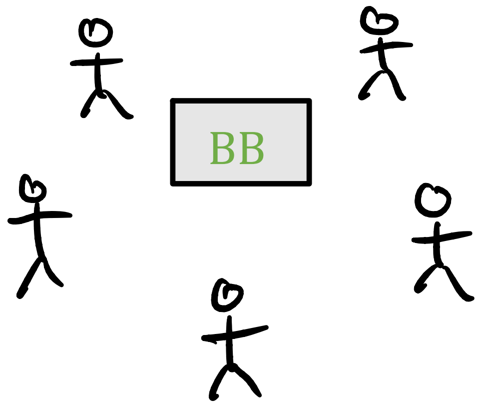
$$(\mathrm{Inbox}_1, \ldots, \mathrm{Inbox}_q) = \mathrm{SA}(\mathrm{Outbox}_1, \ldots, \mathrm{Outbox}_q)$$

- Uses "Sparse Averaging" [SHT'23] + some redundancy:

$$\mathrm{SparseAveraging}(O_1, \ldots, O_q)_j = \frac{1}{\deg(j)} \sum_{i \to j} O_i$$

Sequential multi-party communication

- Input split into k parts $x_1, \ldots, x_k$, given to k players
- Players communicate in round-robin fashion via public blackboard
- (k, R, s) protocol:
  - For $r = 1, \ldots, R$:
    - For $i = 1, \ldots, k$:
      - Player i reads content of BB, appends $F_{r,i}(\text{BB}, x_i) \in \{0,1\}^s$ to BB



- [Assadi & N, 2021]: Every (k, R, s) protocol for k-Pointer Chasing, where Player i gets E_{k+1-i}, must have either $R \geq k$ or $s = \Omega(n/k^6)$