Guiding machine learning design with insights from simple testbeds

Bingbin Liu

Carnegie Mellon University

Jordan T. Ash

Surbhi Goel

Daniel Hsu

Akshay Krishnamurthy

Yuchen Li

Pradeep Ravikumar

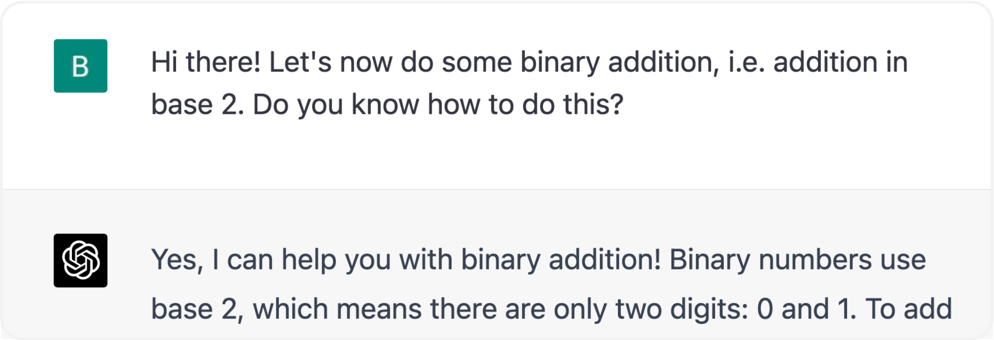
Andrej Risteski

Elan Rosenfeld

Kaiyue Wen

Cyril Zhang

How to improve large-scale machine learning?



$$
\begin{array}{r}
11100111 \\
+\ 11011111 \\
\hline
11\ 10001010
\end{array}
$$

errors

Trial and Error?

(Too many tuning knobs!)

$63M

$85k

What is a <u>principled</u> way to make progress?

Theory?

Understanding machine learning methods

"The Gates building was built in 2014.
How old is Gates in 2024?"

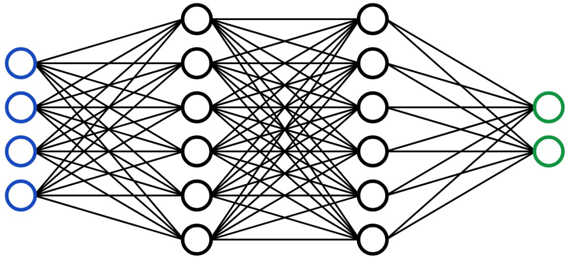
hard to formally analyze

data

(source, size, ordering...)

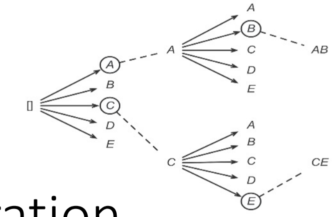
optimization

(loss, optimizer, hyperparameters, ...)



model

(size, activation, normalization, ...)



generation

(order, sampling strategy, ...)

"10 years old."

This talk

With proper simplification,
theory can inform practical machine learning methods.

1. Classic theory toolkits can be applied to understand modern ML.

Understand task design and solutions.

2. Theory-inspired lens can provide practical insights.

As diagnosis tools, improving performance.

Theory toolkits for understanding modern machine learning

Understand the task

[LHRR22]

Masking ratio in masked prediction *[Devlin et al. 2018]?*

→ Formalize with hidden Markov model

→ Tensor decomposition *[Kruskal 1977]*

Understand the solution

[LAGKZ23a]

How Transformers *[Vaswani et al. 2017]* *"reason"?*

→ Formalize with finite automata.

→ Circuits, (semi)groups *[Krohn & Rhodes 1965]*

Understanding design choices of a task

Masked prediction [Devlin et al. 18]: predicting missing words in a sentence.

"Ithaca takes its name from the Greek island of Ithaca in Homer's Odyssey."

Model

"Ithaca takes [MASK] name from the [MASK] island of [MASK] in [MASK] Odyssey."

Competitive performance for language understanding.

How much should we mask?

Understanding design choices of a task

Masked prediction [Devlin et al. 18]: predicting missing words in a sentence.

from

Model

"Ithaca takes its name [MASK] the Greek island of Ithaca in Homer's Odyssey."

Mask too little: easy to infer.

Understanding design choices of a task

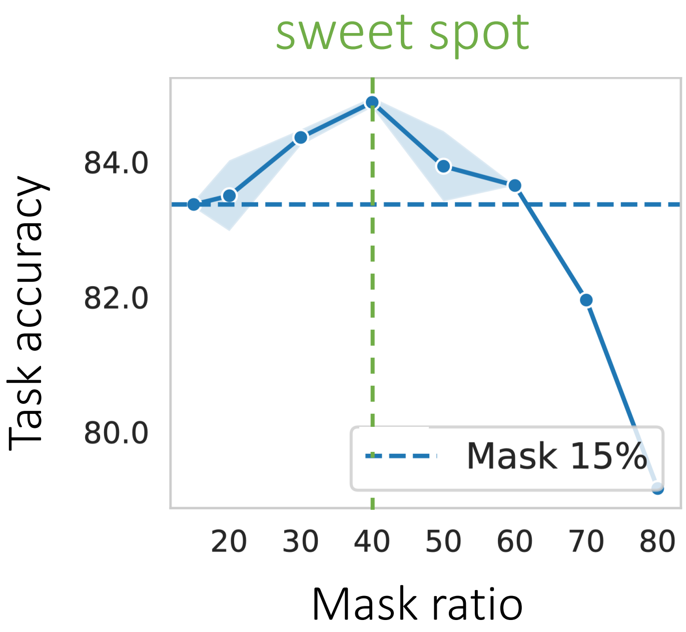
Masked prediction [Devlin et al. 18]: predicting missing words in a sentence.

Mask too much: expensive, or even impossible.

Task design: how much masking is sufficient?

for recovering HMM parameters

15%? [Devlin et al. 18]



[Wettig et al. 22]

[LDRR 22]: with Hidden Markov Model data,

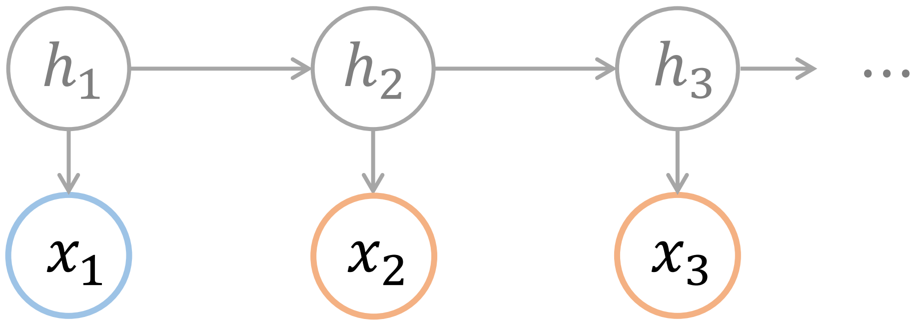
- Masked predictor → tensors;
- Tensor decomposition → HMM parameters.

Masked Prediction with HMM

Data: Hidden Markov Model (HMM): latents $\{h_t\} \rightarrow$ observables $\{x_t\}$.

- Discrete latents $\{h_t\}$: $P(h_{t+1}|h_t) \leftarrow T$ *(transition matrix)*
- Discrete observables $\{x_t\}$: $P(x_t|h_t) \leftarrow O$ *(emission matrix)*

Masked prediction task: e.g. $x_2, x_3 | x_1$ … **up to 3 tokens*



Parameter Identifiability

An identifiable task: parameters can be recovered* from the predictor f.

**Up to permutation:* $O = \tilde{O}\Pi, T = \Pi^\top \tilde{T}\Pi$ *for some permutation* Π *.*

- Predicting with the squared loss.
- Optimal predictor, e.g. $f^{(2,3|1)}(x) = \mathbb{E}[x_2 \otimes x_3 | x_1 = x]$.

Why predicting more helps with identifiability

- Pairwise $(x_{t'}|x_t)$: *not identifiable*
- Triplet $(x_{t_2}, x_{t_3}|x_{t_1})$

(Thm) $\exists \tilde{O}, O$ s.t. $\tilde{O} \neq O$, but produce the same pairwise predictors.

(i.e. $x_2|x_1, x_1|x_2, x_3|x_1, x_1|x_3$)

Intuition: matrix *(2-tensor)* factorization is not unique.

- Matching $f^*(x_2|x_1) \rightarrow$ matching $OTO^\top = \tilde{O}\tilde{T}\tilde{O}^\top$.
- $\tilde{O} \coloneqq OR, \tilde{T} := R^\top TR$ for an orthogonal R (~rotation of a small angle).

Why predicting more helps with identifiability

- Pairwise $(x_{t'}|x_t)$: *not identifiable*
- Triplet $(x_{t_2}, x_{t_3}|x_{t_1})$: *identifiable*

(Thm) O, T are identifiable from the task $x_{t_2} \otimes x_{t_3} \mid x_{t_1}$,

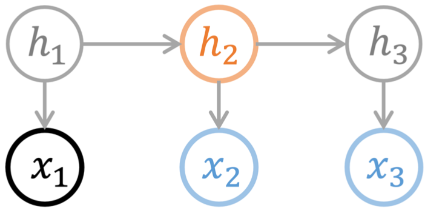
for t_1, t_2, t_3 being any permutation of $\{1,2,3\}$.

Intuition: (3-)tensor decomposition is unique.

- 3-tensor = input $\otimes$ predictor output:

$$W \coloneqq \sum_{x_1} x_1 \otimes \mathbb{E}[x_2 \otimes x_3 | x_1]$$

**prior work: 3rd order moments $\mathbb{E}[x_1 \otimes x_2 \otimes x_3]$*



$\rightarrow W = \sum_i \cdots \otimes O_i \otimes (OT)_i$ ... ***Kruskal's theorem*** $\rightarrow$ *unique* $\{O_i\}, \{(OT)_i\}$.

$(x_2 \perp x_3 | h_2)$ *[Kruskal 1977]*

Theory toolkits for understanding modern machine learning

Understand the task
[LHRR22]

Masking ratio in masked prediction [*Devlin et al. 2018*]*?*

→ Recovering parameters in hidden Markov model

→ Tensor decomposition [*Kruskal 1977*]

Understand the solution
[LAGKZ23a]

How Transformers [*Vaswani et al. 2017*] *"reason"?*

→ How Transformers learn finite automata?

→ Circuits, (semi)groups [*Krohn & Rhodes 1965*]

Theory toolkits for understanding modern machine learning

Understand the task

[LHRR22]

Masking ratio in masked prediction [*Devlin et al. 2018*]*?*

→ Recovering parameters in hidden Markov model

→ Tensor decomposition [*Kruskal 1977*]

Understand the solution

[LAGKZ23a]

How Transformers [*Vaswani et al. 2017*] *"reason"?*

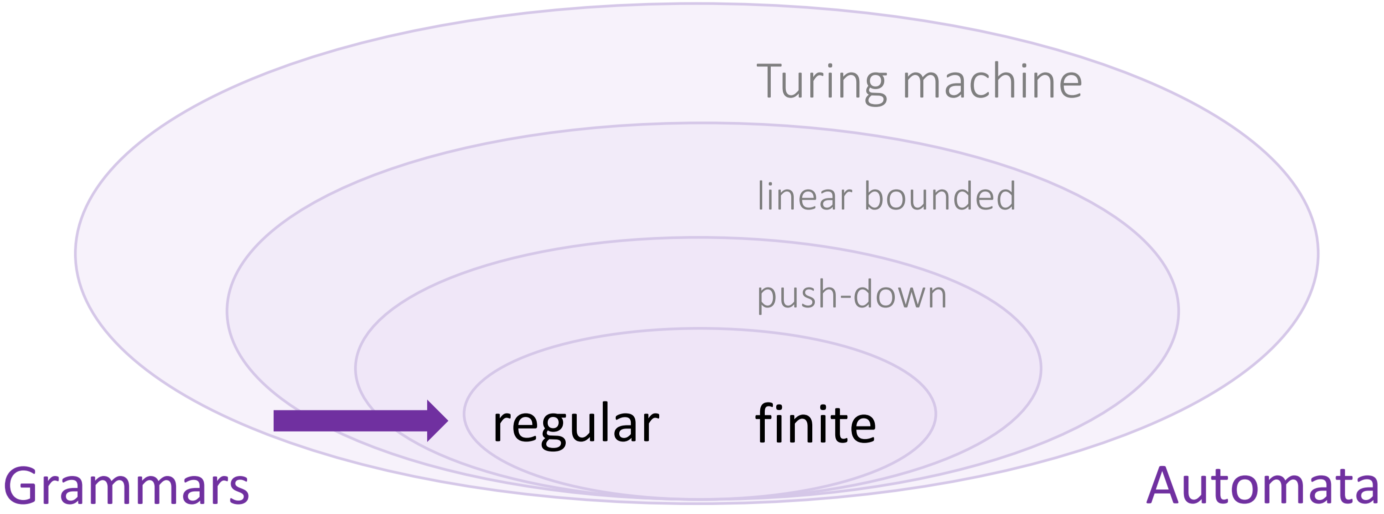
→ How Transformers learn finite automata?

→ Circuits, (semi)groups [*Krohn & Rhodes 1965*]

Transformers for “reasoning”

?

Reasoning is a form of computation.



Goal: classify solutions for learning finite automata.

Prior work: parity (Hanh 20), bounded Dyck (Yao et al. 22),

Sequential reasoning via automata

$$\mathcal{A} = (Q, \Sigma, \delta) \qquad q_t = \delta(q_{t-1}, \sigma_t)$$

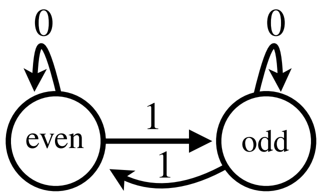
states **inputs** **transitions**

(Q is finite)

parity counter

$Q = \{\text{even}, \text{odd}\}$

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

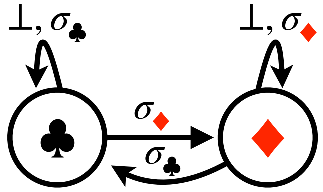


1-bit memory unit

$Q = \{\clubsuit, \diamondsuit\}$

$\Sigma = \{\sigma_{\clubsuit}, \sigma_{\diamondsuit}, \perp\}$

(no-op)



(will reappear later)

Task: simulating the dynamics of $\mathcal{A}$.

Task: Simulating automata

$\mathcal{A} = (Q, \Sigma, \delta)$

states, inputs, transitions

Simulating $\mathcal{A}$: learn a *seq2seq function* for sequence length T.

Output: $q_1 \quad q_2 \quad \cdots \quad q_T \quad \subset Q^T$ (states)

Model (Transformer/RNN)

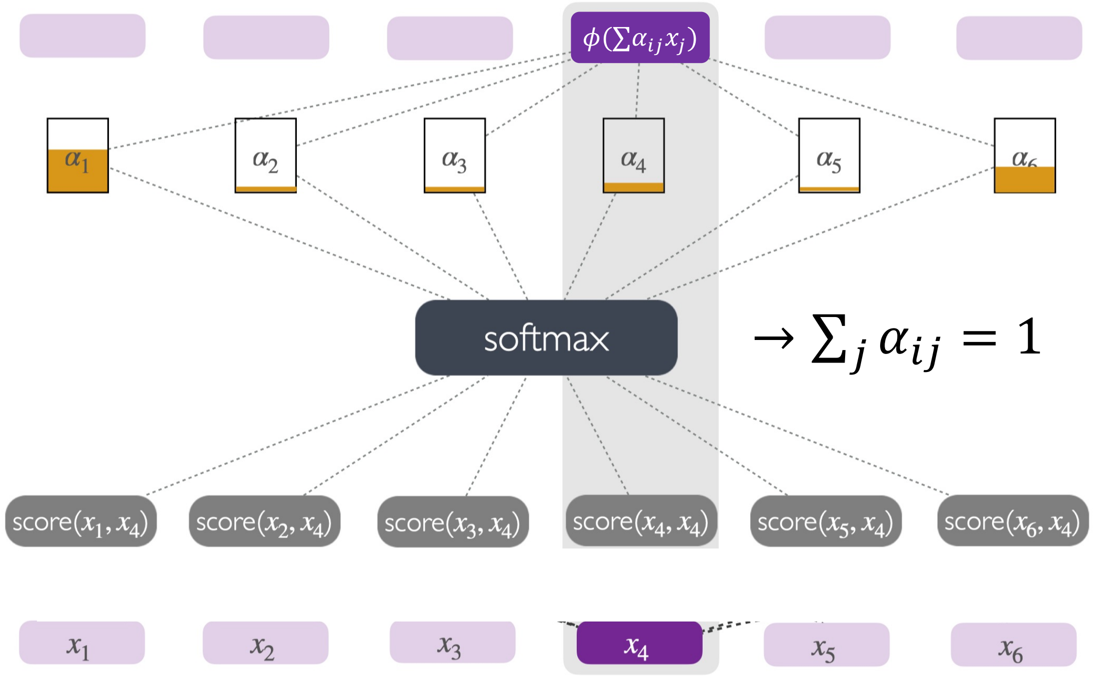
Input: $\sigma_1 \quad \sigma_2 \quad \cdots \quad \sigma_T \quad \subset \Sigma^T$ (alphabet)

The Transformer layer

Computation *parallel* across positions.

attention scores $(\sum_j \alpha_{ij} = 1)$

l^{th} layer, position $i \in [T]$: $x_i^{(l)} = \phi(\sum_{j \le i} \alpha_{ij}^{(l)} x_j^{(l-1)})$



$\rightarrow \sum_j \alpha_{ij} = 1$

Parameters shared across positions.

- ϕ: computed per-position.
- $\alpha_{ij} \propto \exp(\langle W_Q x_i, W_K x_j \rangle)$: the only source of interaction.

The Transformer layer

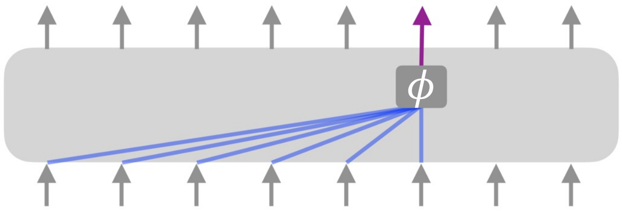
Computation *parallel* across positions.

attention scores: $\sum_j \alpha_{ij} = 1$

l^{th} layer, position $i \in [T]$: $x_i^{(l)} = \phi(\sum_{j \leq i} \alpha_{ij}^{(l)} \, x_j^{(l-1)})$

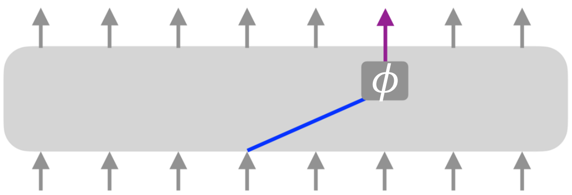
parameters

1. uniform attention / $\vec{\boldsymbol{\alpha}_i} = [\frac{1}{\mathrm{T}}, \frac{1}{\mathrm{T}}, \cdots, \frac{1}{T}]$



e.g. average, sum.

2. sparse attention / $\vec{\boldsymbol{\alpha}_i} = [0, \cdots 1, 0, \cdots]$



e.g. selection.

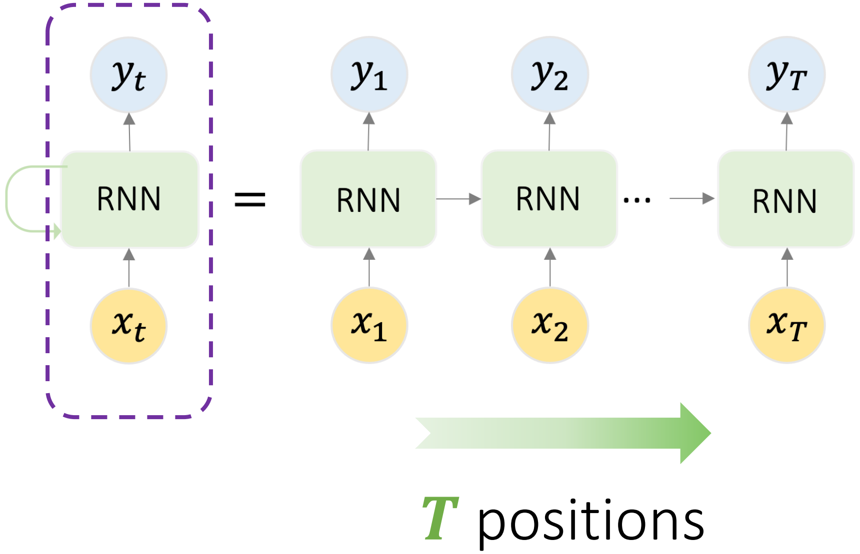
Architecture choices

L (#layers) $\ll T$ (# positions)

Recurrent Neural Nets (RNNs)

sequential across positions

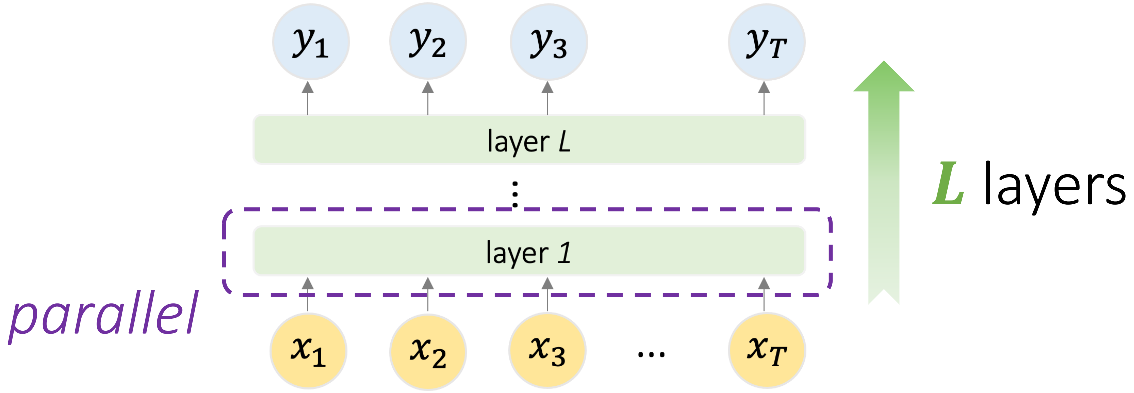
Natural for $q_t = \delta(q_{t-1}, \sigma_t)$



Transformer

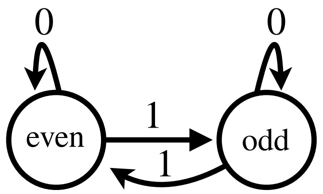
parallel across positions

sequential across layers



~ width-T, depth-L circuit, but with weight sharing.

A <u>parallel</u> model for a <u>sequential</u> task?



parity

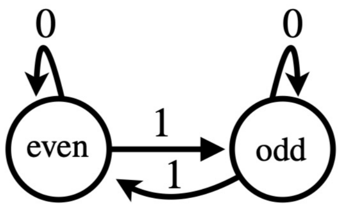
Different ways to simulate automata

$\mathcal{A} = (Q, \Sigma, \delta)$

states, inputs, transitions

Simulating = mapping from $(\sigma_1, \sigma_2, \cdots, \sigma_T) \subset \Sigma^T$ to $(q_1, q_2, \cdots, q_T) \subset Q^T$.

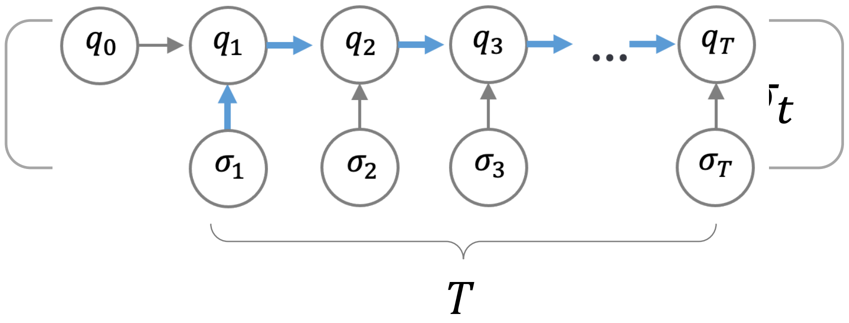
e.g. parity



Shortcut

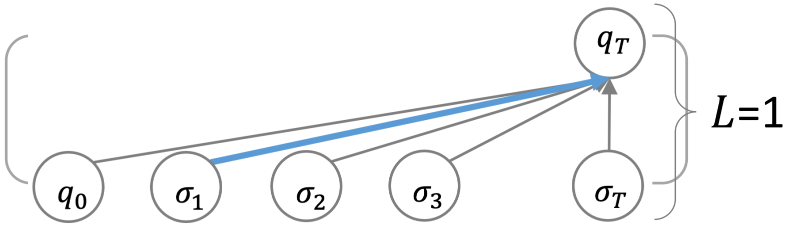
$o(T)$ # sequential steps

Iterative solution



“RNN solutions”

Parallel solution



“Transformer solutions”

Solutions of Reasoning

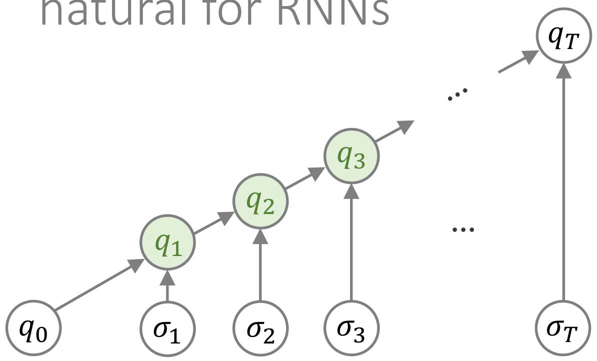
$\mathcal{A} = (Q, \Sigma, \delta), \quad q_t = \delta(q_{t-1}, \sigma_t).$

 # steps = # sequential computation steps

Sequential solutions

(# steps = T)

By δ's definition;
natural for RNNs



iterative state emulation 🐢

Shortcuts (#steps = $o(T)$)

Transformer can simulate $\mathcal{A}$ with:

(Thm 1) $\boldsymbol{O(\log T)}$ layers

(Thm 2) $\boldsymbol{\tilde{O}(|Q|^2)}$ layers

Task structure?

Why Transformer?

$O(\log T)$ steps

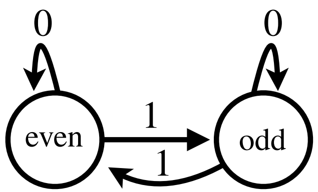
$\mathcal{A} = (Q, \Sigma, \delta)$,
$q_t = \delta(q_{t-1}, \sigma_t)$.

Goal: compute $q_t = \big(\delta(\cdot, \sigma_t) \circ \cdots \circ \delta(\cdot, \sigma_1)\big)(q_0),\ t \in [T]$.

$\delta(\cdot, \sigma)\colon Q \to Q$

function $\longleftrightarrow$ matrix

composition $\longleftrightarrow$ multiplication



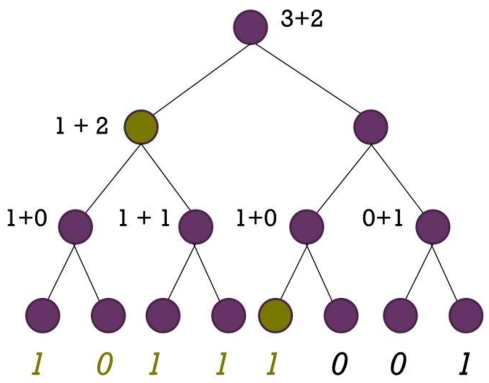
$Q = \{\text{even}, \text{odd}\}$
$\Sigma = \{0, 1\}$

parity counter

$$\delta(\cdot, 0) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

$$\delta(\cdot, 1) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

associativity



$O(\log T)$

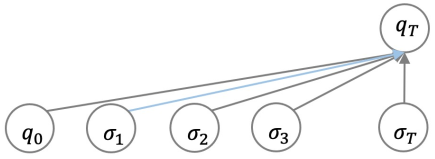
How to use $o(\log T)$ layers?

$$q_t = \left(\delta(\cdot, \sigma_t) \circ \cdots \circ \delta(\cdot, \sigma_1)\right)(q_0)$$

We already have positive results.

- Parity: only need to count #1s.

$$q_t = \left(\textstyle\sum_{\tau \le t} \sigma_\tau\right) \bmod 2$$



$$f \circ g = g \circ f$$

Counting works for commutative function composition: $O(1)$ layers.

$$f \circ g \neq g \circ f$$

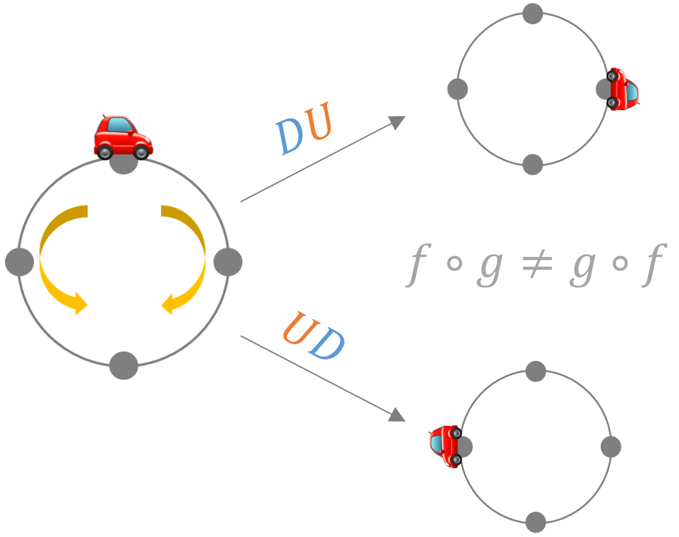
How about *non-commutative* compositions?

Decomposition

Decomposition: car on a circle

$Q = \{$ 🚗 🚗 $\} \times \{0,1,2,3\}, \Sigma = \{D(\text{drive}), U(\text{U–turn})\}.$

$q_0 = ($🚗$, 0), \ \sigma_{1:T} = DDDUDDUUD \rightarrow q_T?$



- Direction = parity (sum) of U. (parity: {1, -1} ↔ {0, 1})
- Position = signed sum mod 4 : sign = parity of U.

$O(1)$ layer each

	q_0	D	D	D	U	D	D	U	U	D	
Parity:	1	1	1	1	-1	-1	-1	1	-1	-1	→ 🚗
Signed sum:	0	1	1	1	0	-1	-1	0	0	-1	→ 0

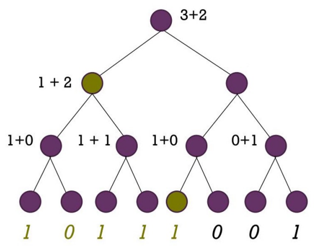
Decomposition: general

What are we decomposing?

Transformation group: $\mathcal{T}(\mathcal{A}) \coloneqq \{\delta(\cdot, \sigma) : \sigma \in \Sigma\}$ under composition.

Set with a binary operator

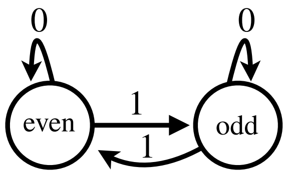
- Associativity: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
- Identity: $e \cdot a = a \cdot e = a$
- Inverse: $a \cdot b = b \cdot a = e$



Decomposition: general

What are we decomposing?

Transformation group: $\mathcal{T}(\mathcal{A}) \coloneqq \{\delta(\cdot, \sigma) : \sigma \in \Sigma\}$ under composition.



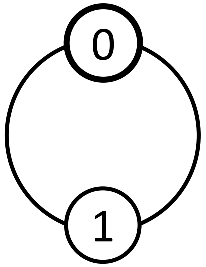
parity counter

(mod 2)

$Q = \{\text{even}, \text{odd}\}$
$\Sigma = \{0, 1\}$

$$\delta(\cdot, 0) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \delta(\cdot, 1) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$\xrightarrow{\mathcal{T}(\mathcal{A})}$



cyclic group C_2

~ 2 (prime factor)

decomposition ≈ prime factorization

e.g. the car example $C_4 \rtimes C_2$.

Decomposing $\mathcal{T}(\mathcal{A})$

Group: associative + invertible

"Prime factorization" for groups:

$$(\mathcal{T}(\mathcal{A}) =)\ G = H_n \rhd \cdots \rhd H_2 \rhd H_1 \text{ (Jordan \& Hölder)}$$

$n = O(\log|G|)$

H_{i+1}/H_i are simple groups

~ prime numbers

→ *If abelian* → *cyclic* → 1 Transformer layer

G is solvable if all $\{H_{i+1}/H_i\}$ are abelian.

Solvable G can be simulated with $O(\log|G|)$ layers.

Decomposition $\mathcal{T}(\mathcal{A})$

Semigroup: associative + ~~invertible~~

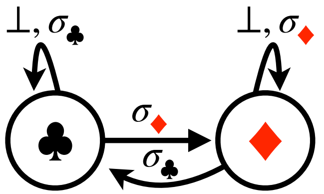
Transformation ~~group~~ semigroup: $\mathcal{T}(\mathcal{A}) \coloneqq \{\delta(\cdot, \sigma) : \sigma \in \Sigma\}$ under composition.

Invertible?

1-bit memory unit

$Q = \{\clubsuit, \color{red}{\blacklozenge}\}$

$\Sigma = \{\sigma_{\clubsuit}, \sigma_{\color{red}{\blacklozenge}}, \perp\}$



$$\delta(\cdot, \sigma_{\clubsuit}) = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$$ ... *singular*

Decomposition $\mathcal{T}(\mathcal{A})$

Semigroup: associative + ~~invertible~~

Transformation ~~group~~ semigroup: $\mathcal{T}(\mathcal{A}) \coloneqq \{\delta(\cdot,\sigma) : \sigma \in \Sigma\}$ under composition.

$\mathcal{T}(\mathcal{A})$ is a group: factorized into groups by Jordan & Hölder.

$\mathcal{T}(\mathcal{A})$ is a semigroup: factorized into *permutation-reset automata* by **Krohn-Rhodes**.

Def: $\delta(\cdot,\sigma)$ is a permutation (forming G) or a constant.

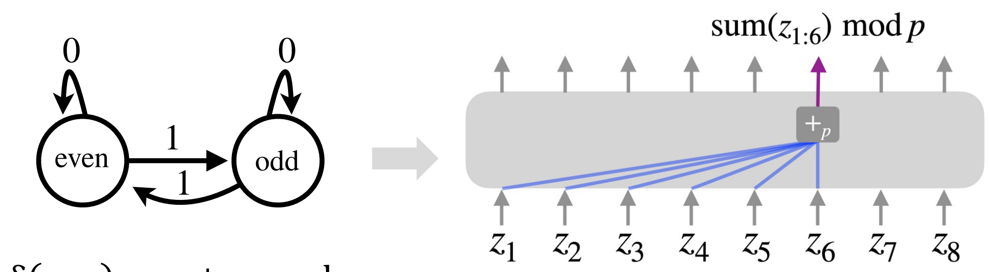
A permutation-reset automaton can be simulated with $O(\log|G|)$ layers.

$\leq |Q|$ $\mathcal{T}(\mathcal{A})$: $\tilde{O}(|Q|^2)$ $O(|Q|\log|Q|)$

$\tilde{O}(|Q|^2)$ steps decomposition with Transformers

Krohn-Rhodes: solvable $\mathcal{A}$ decomposes into permutations and resets.

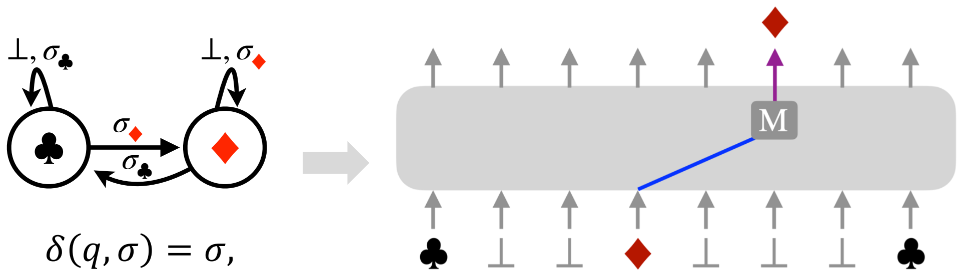
Each representable by 1 Transformer layer



$\delta(q, \sigma) = q + \sigma \bmod p$

cyclic groups

uniform attention



$\delta(q, \sigma) = \sigma,$
$\delta(q, \perp) = q.$

memory unit

sparse attention

Solutions of Reasoning

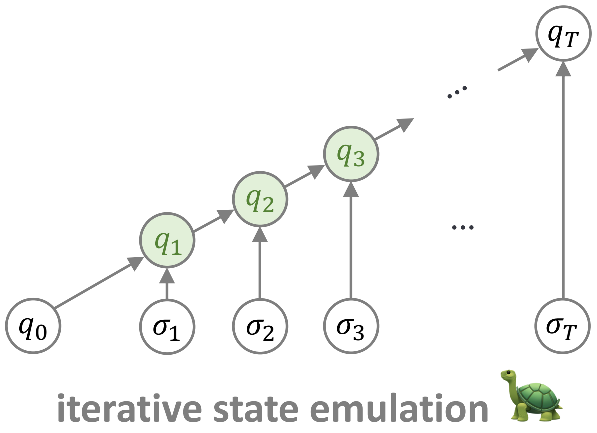
 # steps = # sequential computation steps

$$\mathcal{A} = (Q, \Sigma, \delta), \quad q_t = \delta(q_{t-1}, \sigma_t).$$

Sequential solutions

#steps = T,

by δ or RNNs.



Shortcuts (#steps = $o(T)$)

Transformer can simulate $\mathcal{A}$ with:

(Thm 1) $\boldsymbol{O(\log T)}$ layers.

associativity

tree: divide and conquer

(Thm 2) $\tilde{\boldsymbol{O}}(|\boldsymbol{Q}|^2)$ layers.

algebraic structure

Krohn-Rhodes decomposition

(solvable $\mathcal{A}$ only)

Remarks

All $\mathcal{A}$: $\boldsymbol{O(\log T)}$ **layers**.

Solvable $\mathcal{A}$: $\boldsymbol{\tilde{O}(|Q|^2)}$ **layers**.

1. Can we improve $O(\log T)$ in general? Likely not.

- Constant-depth Transformer is in TC0 [Merrill et al. 21].
- Some automata are NC1 complete (e.g. S_5).

$\rightarrow \Omega(\log T)$ unless TC0 = NC1.

2. What is special about Transformers?

- **Parameter sharing**: T times more efficient than directly "compiling" a circuit.
- **Parallelism**: for a cyclic group C_{2^k}, **1 Transformer layer** vs k steps in Jorden-Holder.

(for any abelian group)

Can theoretical insights lead to practical benefits?

1. Diagnosing trained Transformers [LAGKZ23b]

2. Improving performance [WLLR23]

1. Diagnosing trained Transformers

"Is Transformer always better than RNNs?"

Sanity check: *can shortcuts be found* through finite-sample training?

- Good in-distribution accuracy.
  out-of-distribution?
- Deeper factorization → more layers.
  - Rows ordered by #factorization steps.

Transformer depth $\boldsymbol{L}$ (T=100)

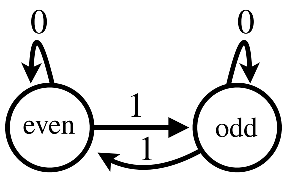
automaton	1	2	3	4	5	6	7	8	12	16
Dyck	99.3	100	100	100	100	100	100	100	100	100
Grid_9	92.2	100	100	100	100	100	100	100	100	100
C_2	77.6	99.8	99.9	100	100	99.5	100	99.7	100	100
C_3	54.6	94.6	96.7	99.4	100	100	99.8	100	100	100
C_2^3	65.0	77.9	99.9	97.9	100	99.8	98.2	99.9	95.9	80.6
D_6	25.4	27.2	47.4	75.2	100	100	100	100	100	100
D_8	45.6	98.0	100	100	100	100	100	100	100	100
Q_8	31.6	49.2	59.6	60.4	73.5	99.3	100	100	100	100
A_5	12.5	23.1	32.5	46.7	71.2	98.8	100	100	100	100
S_5	7.9	11.8	14.6	19.7	26.0	28.4	32.8	51.8	97.2	99.9

non-solvable

1. Diagnosing trained Transformers

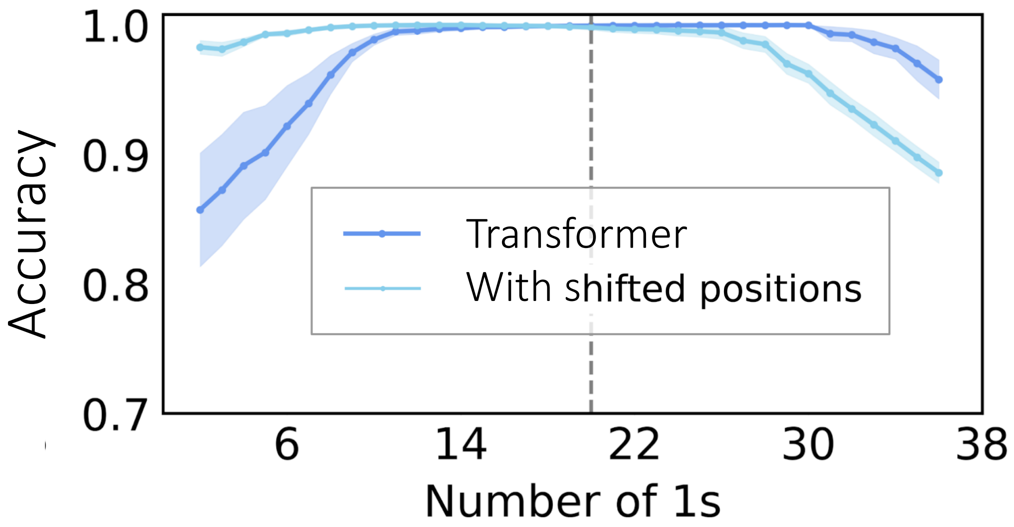
“Is Transformer always better than RNNs?”

Out-of-distribution: train distr ≠ test distr.



- train: $p(1) = 0.5$
- test: varying $p(1)$.

Transformer fails to solve parity.



Parallel shortcut: $q_t = \left(\sum_{i \in [t]} \sigma_i\right) \bmod 2$

- mod: fit by a piecewise-linear network.
  → *fail at unseen* $\left(\sum_{i \in [t]} \sigma_i\right)$.

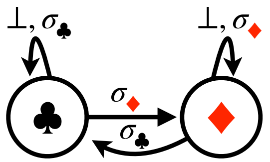
1. Diagnosing trained Transformers

"Is Transformer always better than RNNs?"

Out-of-distribution: train distr ≠ test distr.

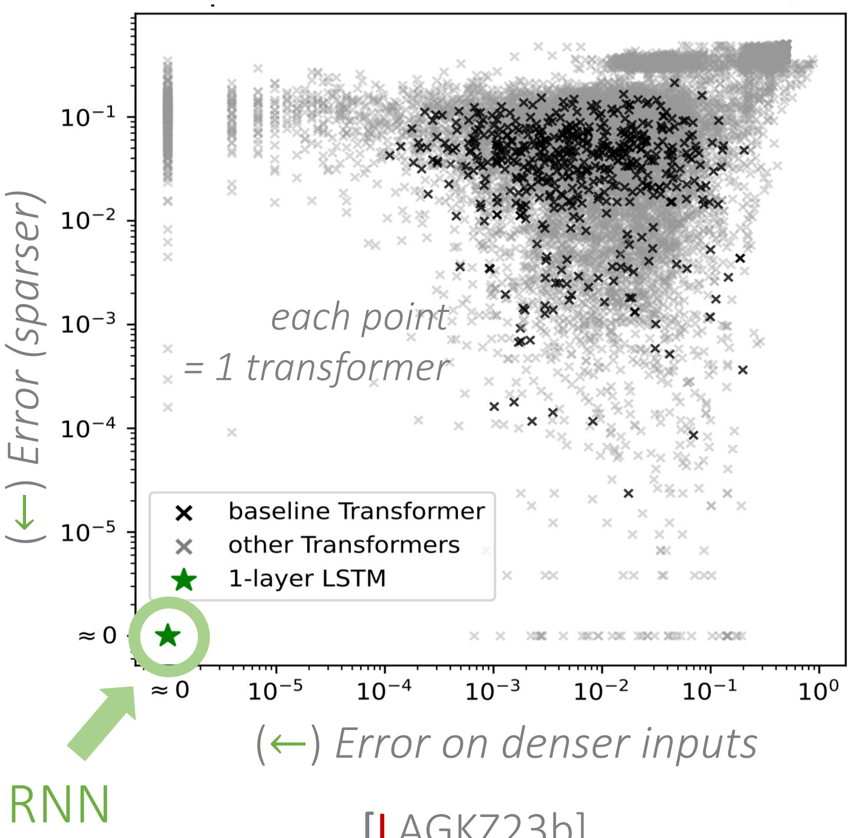
1-bit memory unit

varying $p(\perp)$



Finite-sample training: Transformer < RNN.

- Due to *inherent limitations* of attention.
- Cannot be fixed by *"scaling"* (model/data size ↑).



[LAGKZ23b]

2. Improving performance

Dyck language
(balanced parentheses)

valid ([ ])

invalid ([ )]

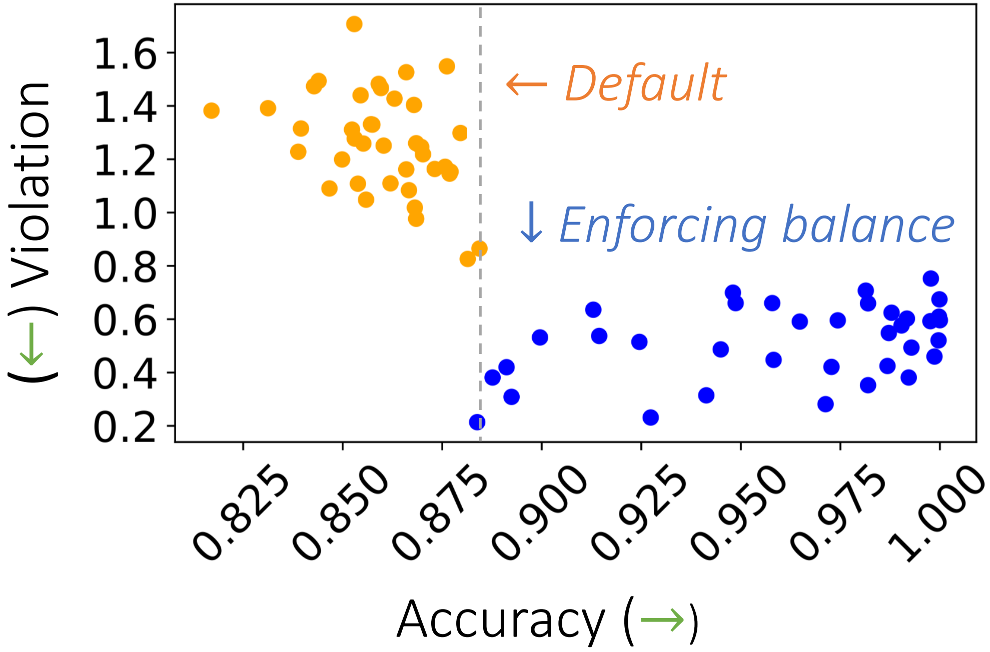
Hierarchical structure

```
for (cond) {
  x[i] = …
}
```

Process: stack or 2-layer Transformer.
[Yao et al. 20]

[WLLR 23]: all 2-layer Transformers solving Dyck need to satisfy a *balanced condition*.

~ a Transformer's version of the pumping lemma.
(informal: $xyz \in L \to xy^*z \in L$.)



Can theoretical insights lead to practical benefits?

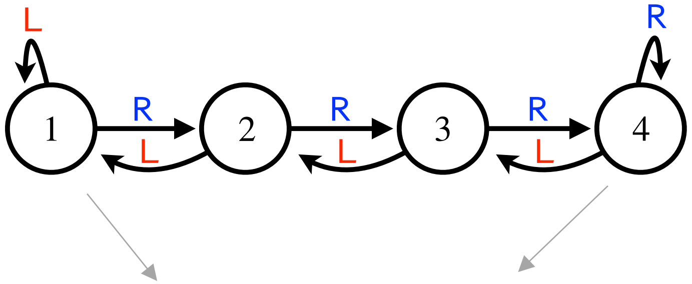
1. Diagnosing trained transformers [LAGKZ23b]

2. Improving performance [WLLR23]

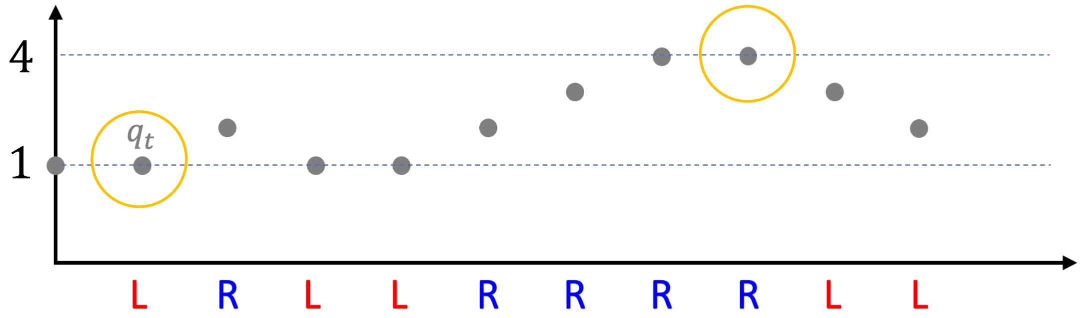
Can practical insights inform theory?

Practice informs theory

1d gridworld: $Q = \{1, 2, 3, 4\}$, $\Sigma = \{$L, R$\}$.



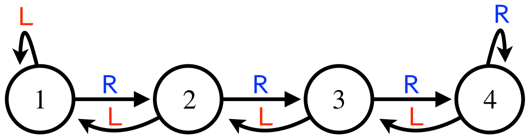
2 boundaries



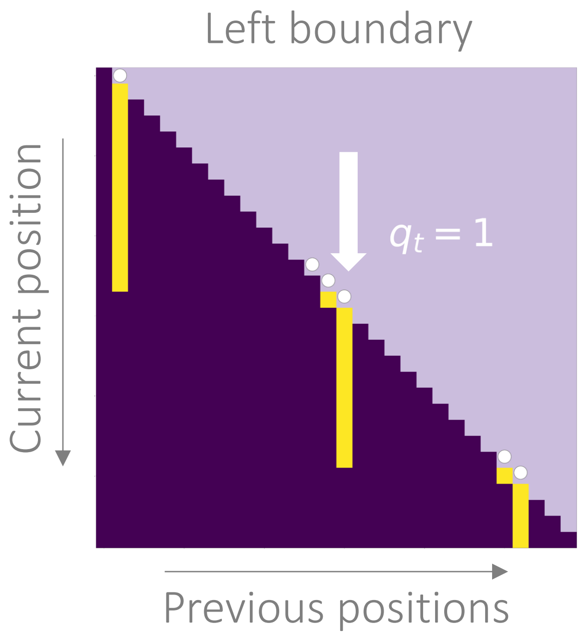
- State matters: LLRR $\neq$ LRLR at state 1, but LLRR $=$ LRLR at state 3.

How to determine the states in parallel?

Practice informs theory

Parallel solution for  ?

Hint from a trained Transformer: ***boundary detection.***



Why boundaries? No boundary → prefix sum.

→ **3-layer solution** (Krohn-Rhodes: $\tilde{O}(|Q|^2)$)

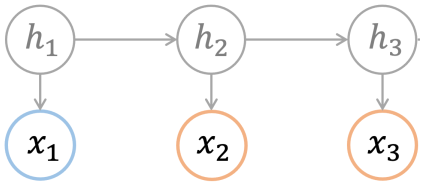
"mechanistic interpretability"

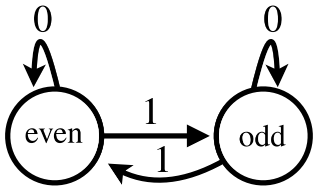
extracting algorithms from trained models

With proper simplification,
theory can inform practical machine learning methods.

1. Classic theory toolkits for understanding modern ML.

Understand task design and solutions.





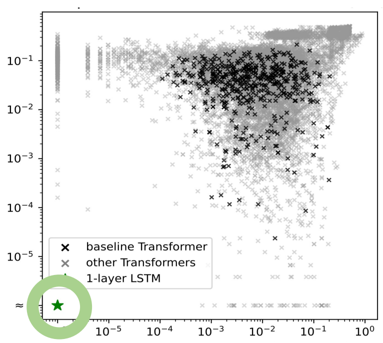
Many other connections!

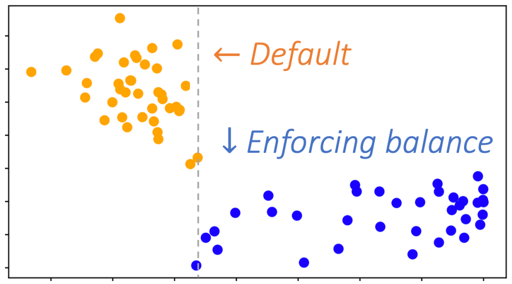
- Circuit complexity [Merrill et al. 21]
- Communication complexity [Sanford et al. 23]

With proper simplification,
theory can inform practical machine learning methods.

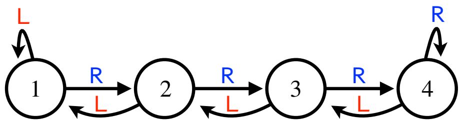
2. Theory-inspired lens can provide practical insights.

As diagnosis tools, improving performance.





... and vice versa!



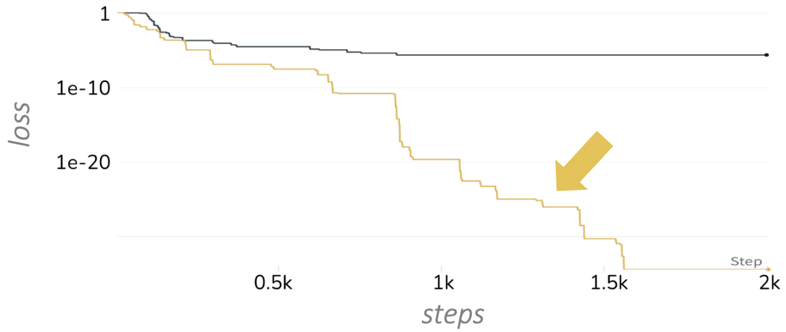
*Future direction: **efficient training***

[LRRR 22]: *why an objective fails to reach optimality in practice?*

→ A simple Gaussian setup.

→ Provable fix with practical gain.

Simple change: $O(\exp(R)) \to O(R^2)$.



Beating the "scaling laws".

- *Knowledge distillation.*
- *Effective use of data, curriculum.*

Bridging synthetic & practical setups.

- *Understanding structures in data.*
- *Behavior changes across scales.*

With the proper simplification,
theory can inform practical machine learning methods.

1. Classic theory toolkits for understanding modern ML.

Understand task design and solutions.

2. Theory-inspired lens can provide practical insights.

As diagnosis tools, improving performance.

… and *vice versa*.

Discovering cool problems and solutions.