

Inductive Biases and Variable Creation in Self-Attention Mechanisms

Benjamin L. Edelman



Surbhi Goel



Sham Kakade



Cyril Zhang

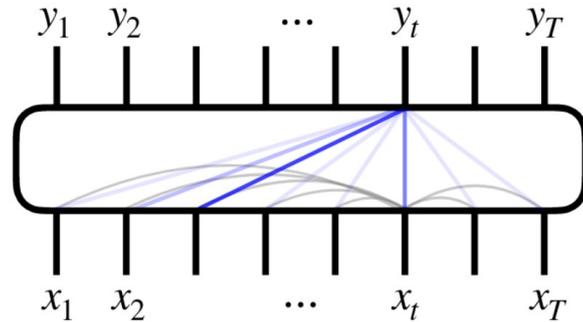


Harvard SEAS

Microsoft Research NYC

The Self-Attention Revolution

Language

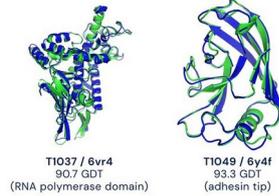


The Self-Attention Revolution

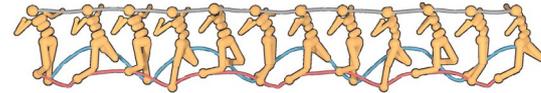
Language



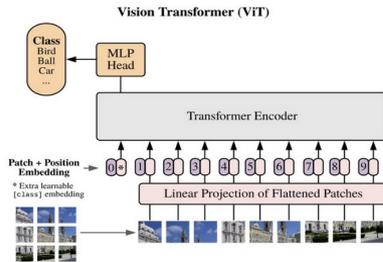
Computational biology



Reinforcement learning



Computer vision



Automated programming

```
1 import datetime
2
3 def parse_expenses(expenses_string):
4     """Parse the list of expenses and return the list of triples (date, value, currency).
5     Ignore lines starting with #,
6     parse the date using datetime.
7     Example expenses_string:
8     2016-01-02 -26.01 USD
9     2016-01-03 2.50 DKK
10    2016-01-03 -2.72 EUR
11    """
12
13    expenses = []
14    for line in expenses_string.splitlines():
15        if line.startswith("#"):
16            continue
17        date, value, currency = line.split(" ")
18        expenses.append((datetime.datetime.strptime(date, "%Y-%m-%d"),
19                        float(value),
20                        currency))
21    return expenses
```



Symbolic Mathematics

$$\int_{-\infty}^{\infty} e^{-x^2} dx = \sqrt{\pi}.$$

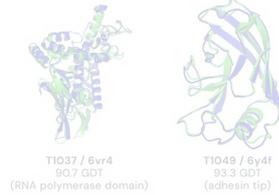
etc.

The Self-Attention Revolution

Language

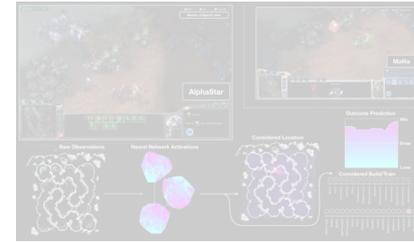


Computational biology



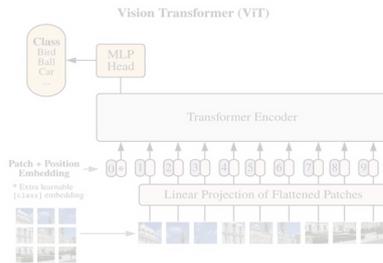
● Experimental result
● Computational prediction

Reinforcement learning



Why?

Computer vision



Automated programming

```
def main_solution():  
    """Parse the list of expenses and return the list of triples (date, value, currency).  
    Ignore lines starting with '#',  
    merge the data using asterisks.  
    Example input: 2017,  
    2016-01-01 1.50 USD,  
    2016-01-01 2.50 USD,  
    2016-01-01 3.75 EUR  
    """  
    expenses = []  
    for line in sys.stdin.readlines():  
        if line.startswith("#"):  
            continue  
        date, value, currency = line.split(",")  
        expenses.append((date.strip(), float(value.strip()), currency.strip()))  
    return expenses
```

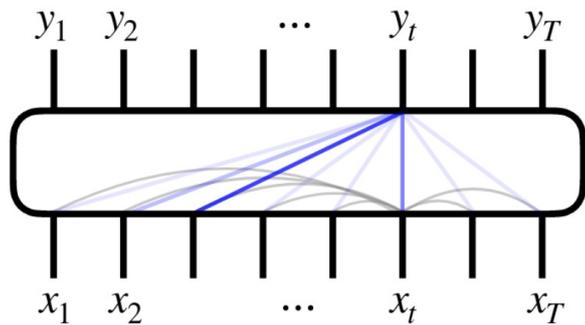


OpenAI Codex

Symbolic Mathematics

$$\int_{-\infty}^{\infty} e^{-x^2} dx = \sqrt{\pi}.$$

etc.



What inductive biases does self-attention have?

Inductive biases of attention

Generalization:

Can we prove
Transformers won't
overfit?

Representation:

What functions can
Transformers
approximate?

Optimization:

Do Transformers
learn these functions
in practice?

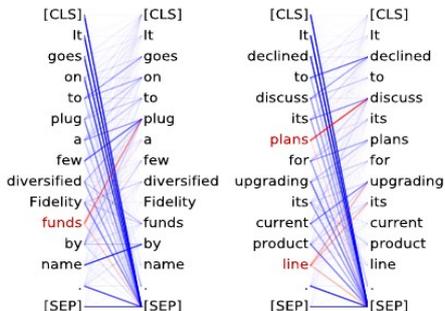
Theory

Experiments

Inductive biases of attention

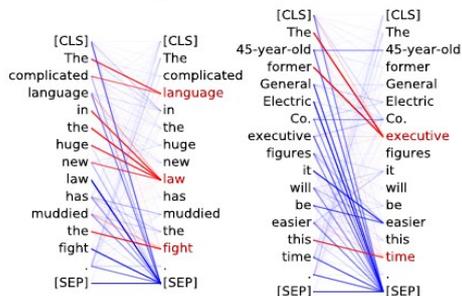
Head 8-10

- **Direct objects** attend to their verbs
- 86.8% accuracy at the **doobj** relation



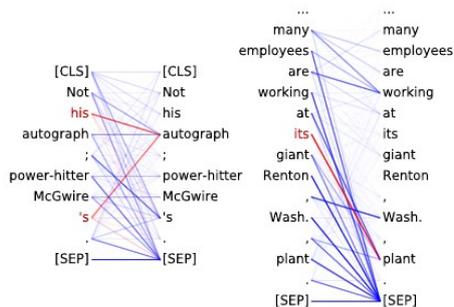
Head 8-11

- **Noun modifiers** (e.g., determiners) attend to their noun
- 94.3% accuracy at the **det** relation



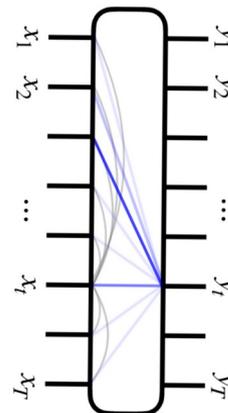
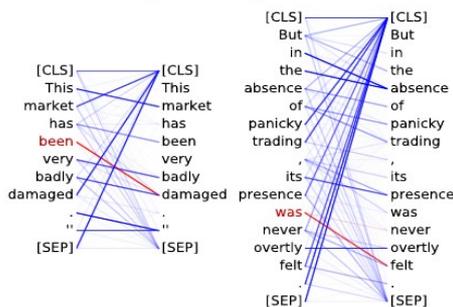
Head 7-6

- **Possessive pronouns** and apostrophes attend to the head of the corresponding NP
- 80.5% accuracy at the **poss** relation



Head 4-6

- **Passive auxiliary verbs** attend to the verb they modify
- 82.5% accuracy at the **auxpass** relation

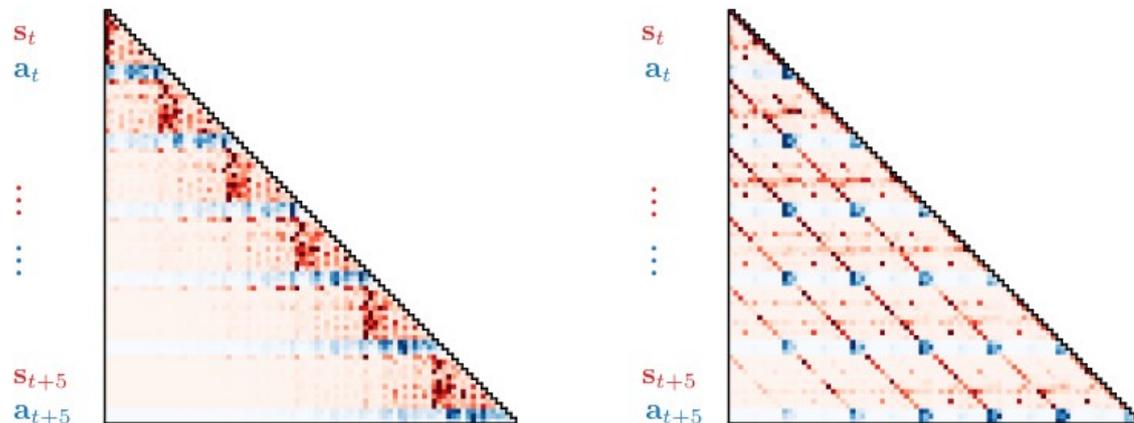


Attention weights are sparse
(or close to uniform)

Source: "What Does BERT Look At? An Analysis of BERT's Attention"

Clark, Khandelwal, Levy, Manning, 2019

Inductive biases of attention

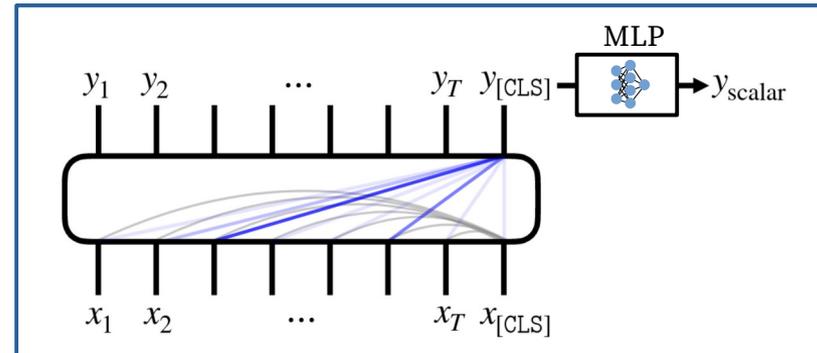
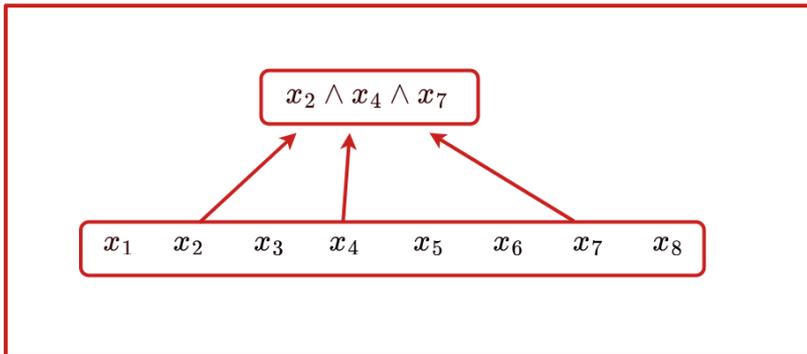


Attention weights are sparse
(or close to uniform)

Source: "Offline Reinforcement Learning as One Big
Sequence Modeling Problem"
Janner, Li, Levine

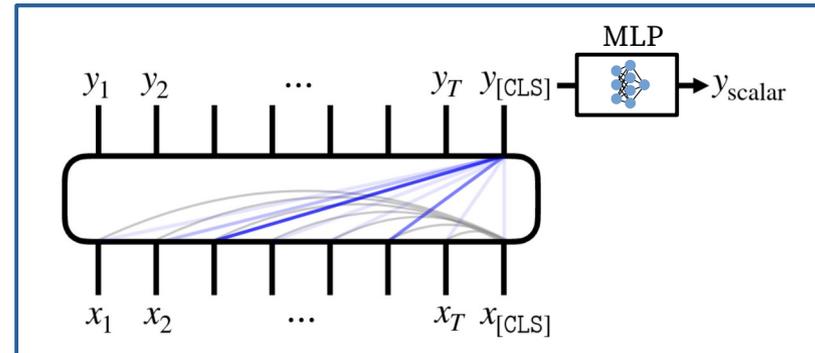
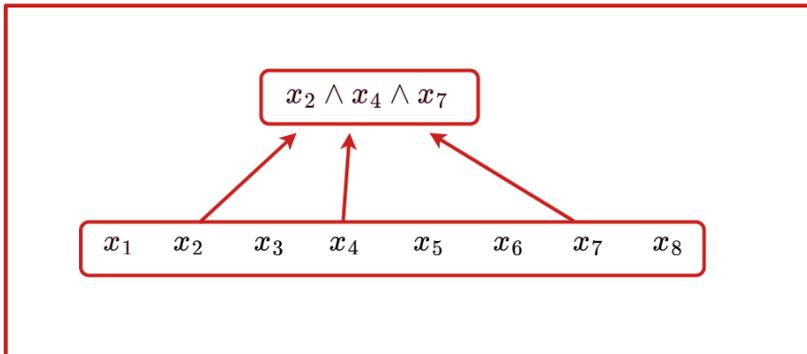
Main result: Sparse variable creation

The class of **s-sparse functions of length- T inputs**
can be learned by
the class of **Transformers layers with weight norms $2^{O(s)}$**
with **sample complexity scaling as $\log(T)$**



Main result: Sparse variable creation

The class of **s-sparse functions of length- T inputs**
can be learned by
the class of **Transformers layers with weight norms $2^{O(s)}$**
with **sample complexity scaling as $\log(T)$** ← **optimal**



Generalization result

When learning any bounded distribution using a Transformer layer, w.p. 99%,

$$|\text{error on training set} - \text{error on distribution}| = \tilde{O}\left(\sqrt{\frac{\text{poly}(C) \cdot \log(T)}{m}}\right).$$

Norms of attention
& MLP weight matrices

Number of
training samples

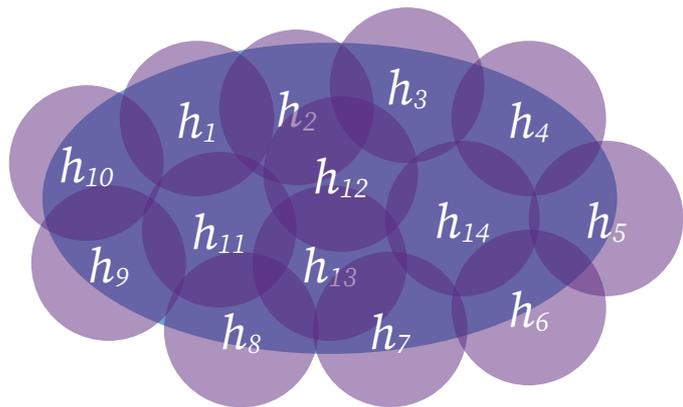
Sequence length

Generalization result

When learning any bounded distribution using a Transformer layer,

$$|\text{error on training set} - \text{error on distribution}| = \tilde{O}\left(\sqrt{\frac{\text{poly}(C) \cdot \log(T)}{m}}\right).$$

Technique: ∞ -covering numbers



Norms of attention
& MLP weight matrices

Number of
training samples

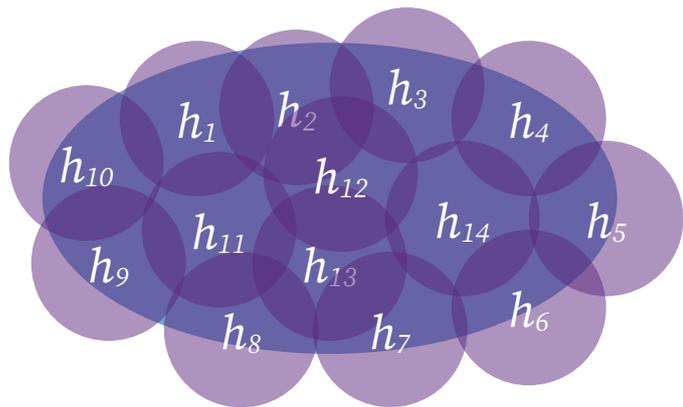
Sequence length

Generalization result

When learning any bounded distribution using a Transformer layer,

$$|\text{error on training set} - \text{error on distribution}| = \tilde{O}\left(\sqrt{\frac{\text{poly}(C) \cdot \log(T)}{m}}\right).$$

Technique: ∞ -covering numbers



Norms of attention
& MLP weight matrices

Number of
training samples

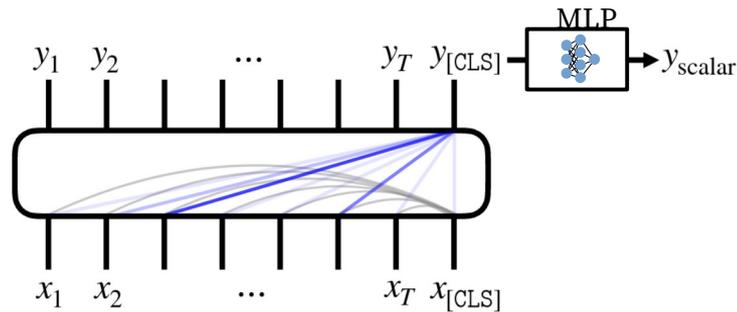
Sequence length

*original draft had dependence
on embedding dimension d

Representation result

Any s -sparse Boolean function f can be exactly represented by a Transformer layer with weight norms $2^{O(s)}$.

If f is symmetric, only $\text{poly}(s)$ weight norms are required.



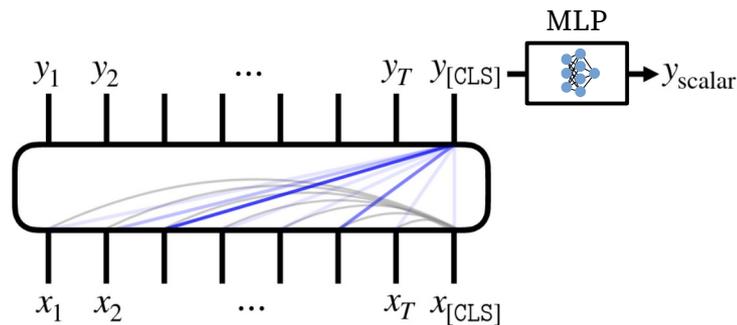
Representation result

Any s -sparse Boolean function f can be exactly represented by a Transformer layer with weight norms $2^{O(s)}$.

If f is symmetric, only $\text{poly}(s)$ weight norms are required.

Intuition

- Softmax allows sparse variable selection
- MLP allows arbitrary function to be applied



Optimization (sparse conjunctions)

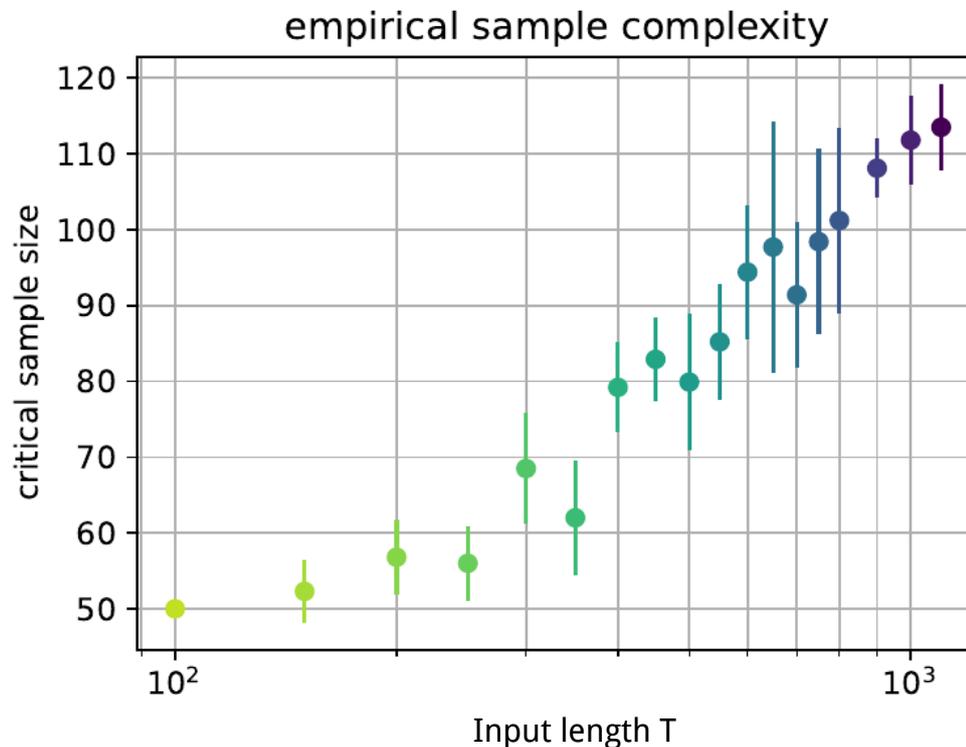
$$\vec{x} \sim \text{unif}(\{0,1\}^T)$$

$$y = x_{i_1} x_{i_2} x_{i_3}$$

Train a Transformer layer

- As input length T grows, how large does the training set need to be to avoid overfitting?

- Consistent with $\log(T)$ dependence in generalization bound!



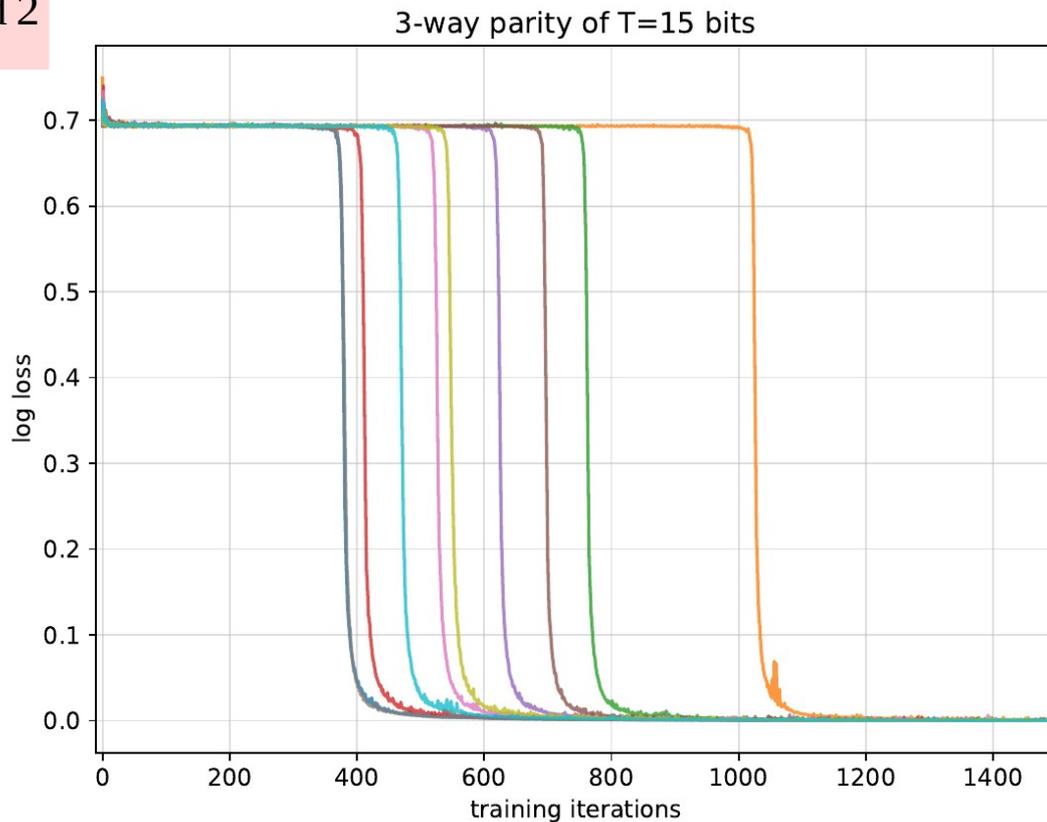
Further contributions

- We define a **general class of attention mechanisms** that includes standard Transformer attention as a special case, and prove a general-purpose generalization bound
 - $l_1 \rightarrow l_\infty$ **smoothness** of softmax is crucial for the generalization result, and the fact that softmax approximates max is crucial for the representation result.
- We extend our generalization bound to deep Transformer networks

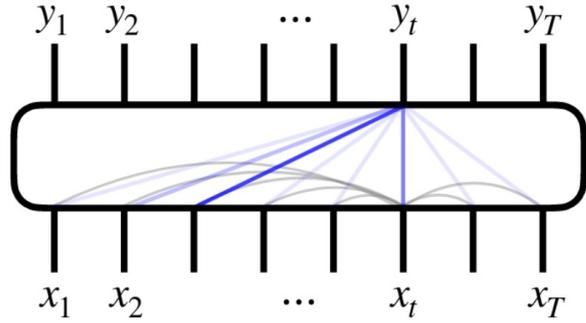
Transformer layers learn sparse parities

$$\vec{x} \sim \text{unif}(\{0,1\}^T)$$

$$y = x_{i_1} + x_{i_2} + x_{i_3} \pmod 2$$



Related: “Grokking:
Generalization Beyond Overfitting
on Small Algorithmic Datasets”
Power, Burda, Edwards,
Babuschkin, Misra



Thank you!