What learning algorithm is in-context learning? Investigations with linear models

ICLR 2023

Ekin Akyürek, Dale Shuurmans, Jacob Andreas, Tengyu Ma, Denny Zhuo

Stanford CS/MS&E 331

Discussions: No laptops, tablets, phones

- Strict rule I feel very passionately about
- No PSETs; active discussion essential for learning
- Demoralizing when student presenters face inattentive class
  - You'll thank me when you're presenting!
- Rule applies to auditors and non-auditors
- Printouts of the paper will be provided during my preview
  - Recycled (and recyclable) paper; main body only
- Please bring printout to the next class for paper discussion

Motivation

- In-context learning (ICL): transformer trained to produce map
  - **Input:** sequences $\left[\left(x_1, f(x_1)\right), \left(x_2, f(x_2)\right), \ldots, x_n\right]$
  - **Output:** prediction of $f(x_n)$
- **This paper:** algorithmic reasoning as a lens to understand ICL
- Algorithmic task: regression
  - $\boldsymbol{x} \in \mathbb{R}^d, f(\boldsymbol{x}) \in \mathbb{R}$
- ICL isn't learning a **regressor;** rather a regression **algorithm**
  - ICL doesn't explicitly specify inner learning procedure
  - Procedure exists only implicitly through transformer's parameters

Motivation

Goal: move toward algorithmic understanding of ICL

Motivating questions:

- What algorithms are implementable by transformers?
- Can we understand what algorithm it's using?

Contributions

Theory: Transformers can implement
- Gradient descent updates
- Closed-form ridge regression updates

Behavior: ICL matches:
- OLS on noiseless data
- Ridge regression under noisy data
  - Minimum Bayes risk predictor

Mechanism: Hidden states encode meaningful quantities
- Encoding is non-linear, revealed by probe models

ICL training objective

Learning setup (linear regression):

- $\mathcal{F} = \{f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} \mid \boldsymbol{w} \in \mathbb{R}^d\}$
- Loss function $\mathcal{L}(y, y') = (y - y')^2$
- Distribution $p(f)$ over $\mathcal{F}$
- Distribution $p(\boldsymbol{x})$ over $\mathbb{R}^d$

Transformer $T_{\boldsymbol{\theta}}$ with trainable parameters $\boldsymbol{\theta}$

- Train $T_{\boldsymbol{\theta}}$ to be an **in-context learner**:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \underset{\substack{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n \sim p(\boldsymbol{x}) \\ f \sim p(f)}}{\mathbb{E}} \left[\sum_{i=1}^{n} \mathcal{L}\big(f(\boldsymbol{x}_i), T_{\boldsymbol{\theta}}([\boldsymbol{x}_1, f(\boldsymbol{x}_1), \boldsymbol{x}_2, f(\boldsymbol{x}_2), \ldots, \boldsymbol{x}_i])\big) \right]$$

Linear regression: Refresher

Inputs $X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]$ and $\boldsymbol{y} = [y_1, \ldots, y_n]$

Regularized linear regression objective:

$$\operatorname*{argmin}_{\boldsymbol{w}} \sum_{i=1}^{n} \mathcal{L}(\boldsymbol{w}^\top \boldsymbol{x}_i, y_i) + \lambda \|\boldsymbol{w}\|_2^2$$

$\lambda = 0$: Ordinary least-squares regression (OLS)

$\lambda > 0$: Ridge regression

Outline

1. **Theory**
2. Empirics

Implementation primitives

- Need simple building blocks for algorithm implementation
- Four primitives: mov, mul, div, aff
    - **mov**: copy values between hidden state positions
    - **mul**: matrix multiplication from hidden state entries
    - **div**: entry-wise division of hidden state entries
    - **aff**: affine transform combining hidden state subsets

- **Lemma:** each primitive implementable by a transformer layer

Gradient descent in a transformer

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_{i=1}^{n} \mathcal{L}(\boldsymbol{w}^\top \boldsymbol{x}_i, y_i) + \lambda \|\boldsymbol{w}\|_2^2$$

One-step of gradient descent:

$$\boldsymbol{w}' = \boldsymbol{w} - 2\alpha(\boldsymbol{x}_i \boldsymbol{w}^\top \boldsymbol{x}_i - y_i \boldsymbol{x}_i + \lambda \boldsymbol{w})$$

Theorem: transformer can implement this with

- Constant number of layers
- $O(d)$ hidden space (where $\boldsymbol{x}, \boldsymbol{w} \in \mathbb{R}^d$)

Closed-form regression by a transformer

- OLS solution $\boldsymbol{w}^* = (X^\top X)^{-1} X^\top \boldsymbol{y}$; for simplicity, set $\lambda = 0$
- Iterative algorithm (suitable for a layer of a transformer):
  1. Define $P_0 = \mathbf{0} \in \mathbb{R}^{d \times d}$; $\boldsymbol{q}_0 = \mathbf{0} \in \mathbb{R}^d$
  2. For $i = 1, \dots, n$:
     i. Compute $P_i = P_{i-1} + \boldsymbol{x}_i \boldsymbol{x}_i^\top$ and its inverse

$$P_i^{-1} = \left(P_{i-1} + \boldsymbol{x}_i \boldsymbol{x}_i^\top\right)^{-1} = P_{i-1}^{-1} - \frac{1}{1 + \boldsymbol{x}_i P_{i-1}^{-1} \boldsymbol{x}_i} \left(P_{i-1}^{-1} \boldsymbol{x}_i\right)\left(P_{i-1}^{-1} \boldsymbol{x}_i\right)^\top$$

Main point: storing P_{i-1}^{-1} in the hidden state,
update can be calculated with primitives mov, mul, div, aff

     ii. Compute $\boldsymbol{q}_i = \boldsymbol{q}_{i-1} + y_i \boldsymbol{x}_i$
- Return $\boldsymbol{w}^* = P_n^{-1} \boldsymbol{q}_n$

Closed-form regression by a transformer

Theorem: transformer can compute P_i, P_i^{-1}, $\boldsymbol{q}_i$ with

- Constant number of layers
- $O(d^2)$ hidden space (where $\boldsymbol{x}, \boldsymbol{w} \in \mathbb{R}^d$)

Outline

1. Theory
2. **Empirics**

What computation does ICL perform?

Behavioral metrics to quantify the extent two algorithms agree:

- Given learning algorithm $\mathcal{A}$:
  - Input dataset $D = [\boldsymbol{x}_1, y_1, \dots, \boldsymbol{x}_n, y_n]$, output prediction $\mathcal{A}(D)(\boldsymbol{x}) \in \mathbb{R}$
- **Squared prediction difference:**

$$\mathrm{SPD}(\mathcal{A}_1, \mathcal{A}_2) = \underset{\substack{D \\ \boldsymbol{x}' \sim p(\boldsymbol{x})}}{\mathbb{E}} \left[\left(\mathcal{A}_1(D)(\boldsymbol{x}') - \mathcal{A}_2(D)(\boldsymbol{x}') \right)^2 \right]$$

What computation does ICL perform?

Behavioral metrics to quantify the extent two algorithms agree:

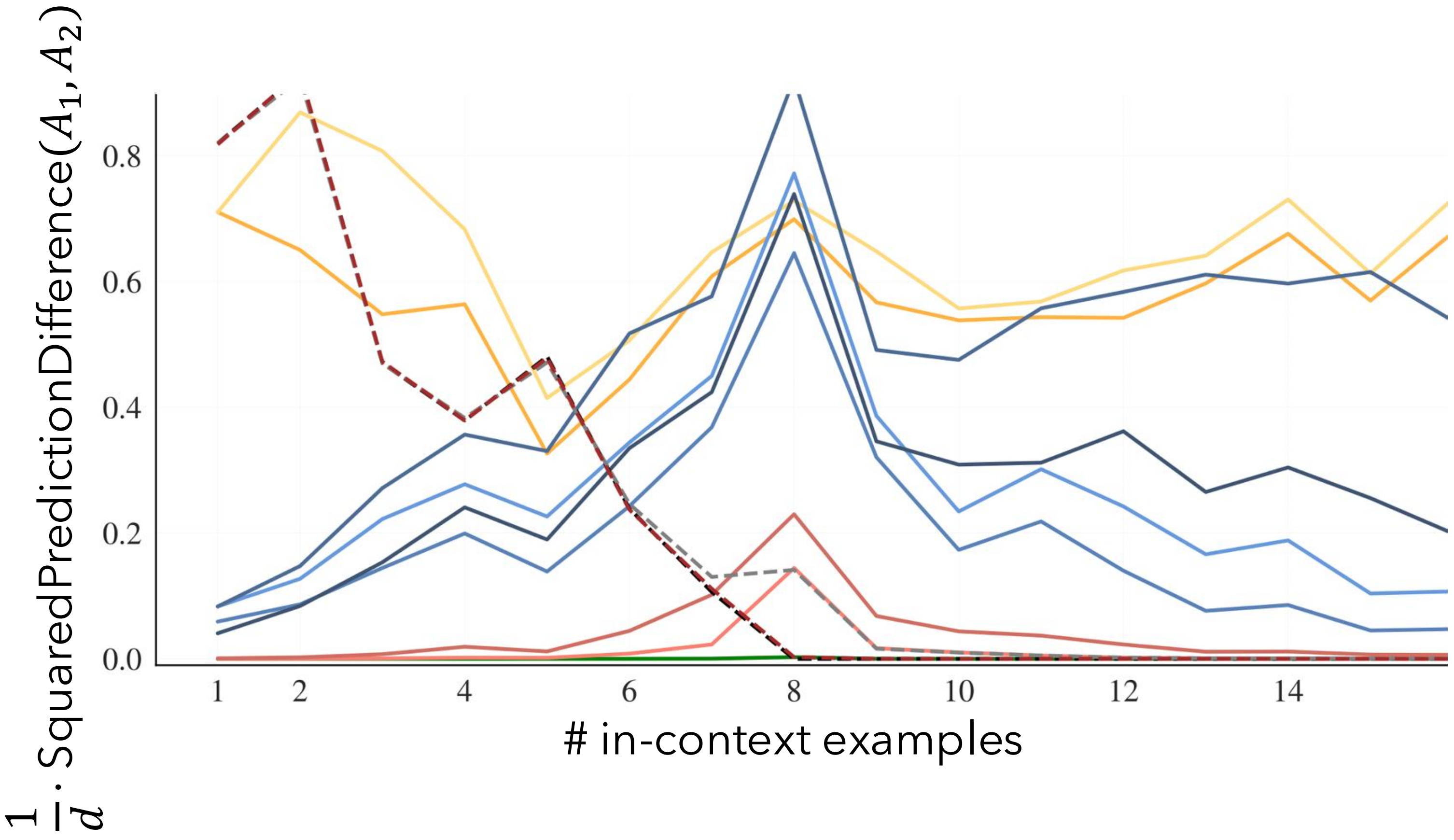
- Given learning algorithm $\mathcal{A}$:
  - Input dataset $D = [\boldsymbol{x}_1, y_1, \dots, \boldsymbol{x}_n, y_n]$, output prediction $\mathcal{A}(D)(\boldsymbol{x}) \in \mathbb{R}$
- If $T_{\boldsymbol{\theta}}$ learning a linear function, what are the function's weights?
- Sample a set $D' = \{\boldsymbol{x}'_1, \dots, \boldsymbol{x}'_m\} \sim p(\boldsymbol{x})$ of test points
- "Implicit weights" of $\mathcal{A}$: $\widehat{\boldsymbol{w}}_{\mathcal{A}} = \underset{\widehat{\boldsymbol{w}}}{\operatorname{argmin}} \sum_{i=1}^{m} \left(\widehat{\boldsymbol{w}}^\top \boldsymbol{x}'_i - \mathcal{A}(S)(\boldsymbol{x}'_i) \right)^2$
- ImplicitLinearWeightsDifference$(\mathcal{A}_1, \mathcal{A}_2) = \mathbb{E}\left[\left\| \widehat{\boldsymbol{w}}_{\mathcal{A}_1} - \widehat{\boldsymbol{w}}_{\mathcal{A}_2} \right\|_2^2 \right]$

Experimental setup: Noiseless setting

- Each in-context training dataset consists of 40 $(\boldsymbol{x}, y)$ pairs
- $p(\boldsymbol{x}) = \mathcal{N}(\mathbf{0}, I), p(\boldsymbol{w}) = \mathcal{N}(\mathbf{0}, I)$ over $\mathbb{R}^8$

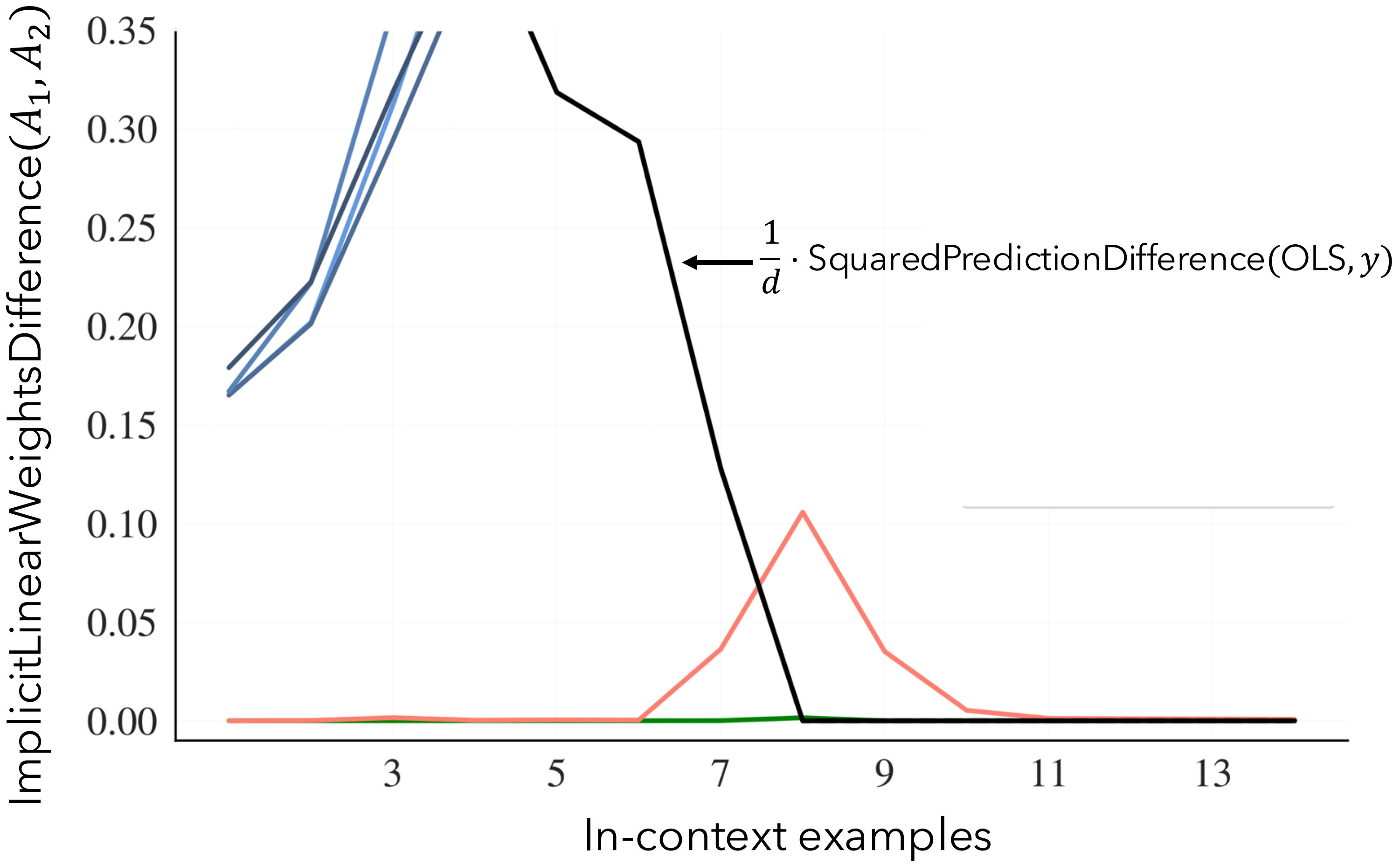
ICL matches OLS on noiseless data

	$\mathcal{A}_1$	$\mathcal{A}_2$
	Ordinary least squares (OLS)	Transformer
	Ridge regression with regularization parameter $\lambda = 0.1$	Transformer
	Ridge regression with $\lambda = 0.5$	Transformer
	1 step of GD with learning rate $\alpha = 0.01$	Transformer
	1 pass of SGD with $\alpha = 0.01$	Transformer
	1 step of GD with $\alpha = 0.02$	Transformer
	1 pass of SGD with $\alpha = 0.03$	Transformer
	3-nearest neighbors (weighted)	Transformer
	3-nearest neighbors (unweighted)	Transformer
■ ■ ■	OLS	y
■ ■ ■	Ridge regression with $\lambda = 0.1$	y
■ ■ ■	Transformer	y



ICL matches OLS on noiseless data

	$\mathcal{A}_1$	$\mathcal{A}_2$
green	Ordinary least squares (OLS)	Transformer
salmon	Ridge regression with regularization parameter λ = 0.1	Transformer
light blue	1 step of GD with learning rate α = 0.01	Transformer
dark blue	1 pass of SGD with α = 0.01	Transformer
blue	1 step of GD with α = 0.02	Transformer
slate	1 pass of SGD with α = 0.03	Transformer



Experimental setup: Noisy setting

- Each in-context training dataset consists of 40 pairs
$$[(\boldsymbol{x}_1, \boldsymbol{w}^\top \boldsymbol{x}_1 + \epsilon_1), (\boldsymbol{x}_2, \boldsymbol{w}^\top \boldsymbol{x}_2 + \epsilon_2), \dots]$$
- $p(\boldsymbol{x}) = \mathcal{N}(\mathbf{0}, I)$ over $\mathbb{R}^8$
- $p(\boldsymbol{\epsilon}) = \mathcal{N}(\mathbf{0}, \sigma^2 I)$
- $p(\boldsymbol{w}) = \mathcal{N}(\mathbf{0}, \tau^2 I)$

Ridge regression with $\lambda = \frac{\sigma^2}{\tau^2}$ returns min Bayes risk predictor

Squared prediction difference

(A_1, A_2)

σ^2/τ^2	0	1/16	1/9	1/4	4/9
(**OLS**, Transformer)	**1.25E-05**	1.34E-04	3.96E-04	1.51E-03	4.13E-03
(**Ridge(1/16)**, Transformer)	1.1E-04	**3.29E-05**	1.12E-04	8.24E-04	2.92E-03
(**Ridge(1/9)**, Transformer)	3.49E-04	9.65E-05	**3.86E-05**	4.5E-04	2.15E-03
(**Ridge(1/4)**, Transformer)	1.69E-03	8.64E-04	4.39E-04	**3.3E-05**	6.81E-04
(**Ridge(4/9)**, Transformer)	4.83E-03	3.09E-03	2.21E-03	7.52E-04	**6.1E-05**

Noisy setting: ICL matches minimum Bayes risk predictor

Does $T_{\boldsymbol{\theta}}$ encode meaningful quantities?

- What are quantities we'd expect a regression alg to compute?
  - Examples: $\boldsymbol{w}_{OLS}, X^\top \boldsymbol{y}$, where

$$X = \begin{pmatrix} | & & | \\ \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_n \\ | & & | \end{pmatrix} \text{ and } y = \begin{pmatrix} \boldsymbol{w}^\top \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{w}^\top \boldsymbol{x}_n \end{pmatrix}$$

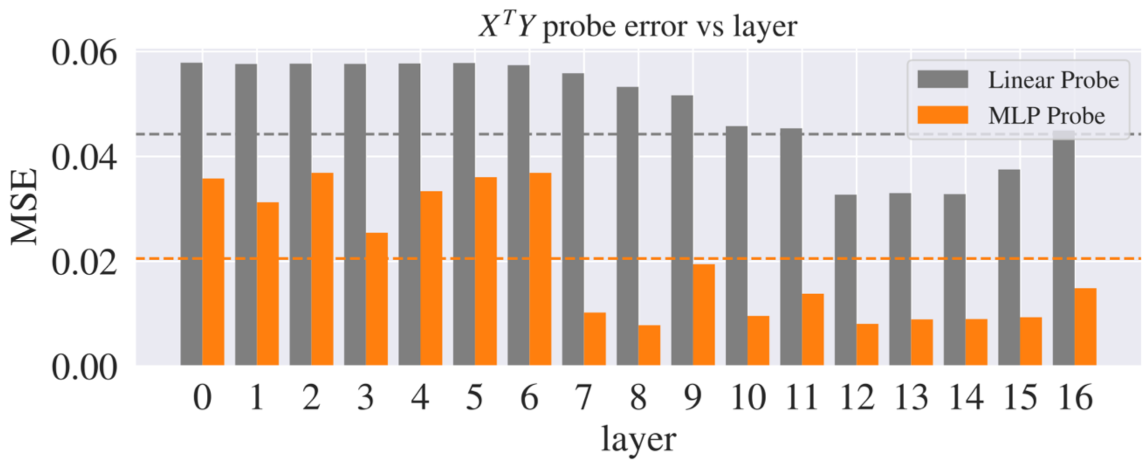
  - We'll call these "probes" $\boldsymbol{v} \in \mathbb{R}^k$ [Alain, Bengio, '17]
- Let $H^{(\ell)}$ be the transformer's hidden states at layer ℓ
- Question: is $\boldsymbol{v}$ "encoded" in $H^{(\ell)}$?
  - i.e., is it some simple function of $H^{(\ell)}$?

Does $T_{\boldsymbol{\theta}}$ encode meaningful quantities?

Probing model: $\hat{\boldsymbol{v}} = f\left(\boldsymbol{s}^{\top} H^{(\ell)}\right)$ where:

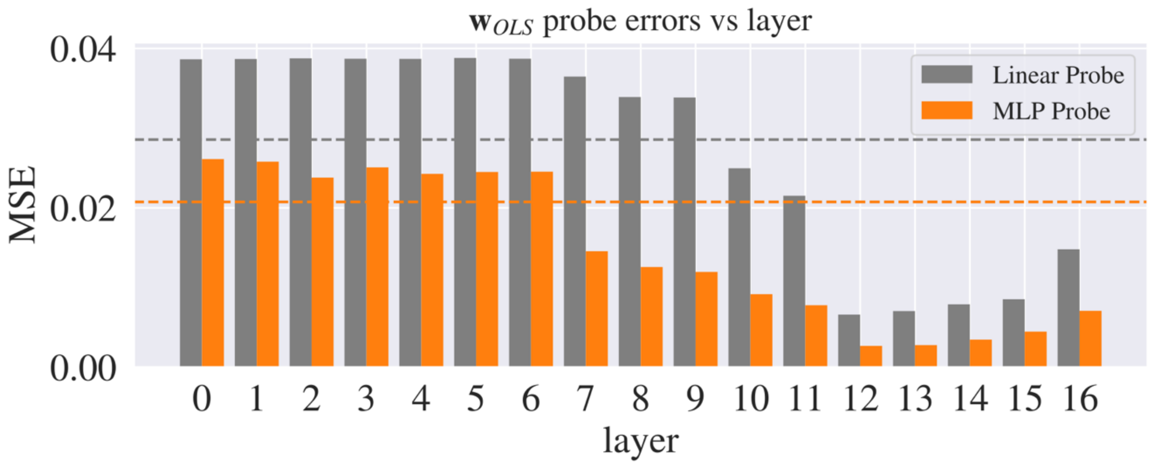
- $\boldsymbol{s}$ is a learned weight vector
- f is a learned function. Two experiments:
  - f is linear
  - f is a 2-layer MLP
- Train to minimize loss $||\boldsymbol{v} - \hat{\boldsymbol{v}}||_2^2$
- Train a different $\boldsymbol{s}$ and f for each sequence length and layer

Probing results



Phase transitions:
around layers 7 and 12

Probes encoded non-linearly



Contributions

- **Goal:** move toward an algorithmic understanding of ICL
- **Theory:** Transformers can implement
  - Gradient descent updates
  - Closed-form ridge regression updates
- **Behavior:** ICL matches:
  - OLS on noiseless data
  - Minimum Bayes risk predictor under noisy data
- **Mechanism:** Hidden states encode meaningful quantities
  - Encoding is non-linear, revealed by probe models