

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](#)

Motivation

- In-context learning (ICL): transformer trained to produce map
 - **Input:** sequences $[(x_1, f(x_1)), (x_2, f(x_2)), \dots, x_n]$
 - **Output:** prediction of $f(x_n)$
- **This paper:** algorithmic reasoning as a lens to understand ICL
- Algorithmic task: regression
 - $\mathbf{x} \in \mathbb{R}^d, f(\mathbf{x}) \in \mathbb{R}$
- 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):

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

Transformer T_{θ} with trainable parameters θ

- Train T_{θ} to be an **in-context learner**:

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

Linear regression: Refresher

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

Regularized linear regression objective:

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

$\lambda = 0$: 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

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

One-step of gradient descent:

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

Theorem: transformer can implement this with

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

Closed-form regression by a transformer

- OLS solution $\mathbf{w}^* = (X^\top X)^{-1} X^\top \mathbf{y}$; for simplicity, set $\lambda = 0$
- Iterative algorithm (suitable for a layer of a transformer):

1. Define $P_0 = \mathbf{0} \in \mathbb{R}^{d \times d}$; $\mathbf{q}_0 = \mathbf{0} \in \mathbb{R}^d$

2. For $i = 1, \dots, n$:

- i. Compute $P_i = P_{i-1} + \mathbf{x}_i \mathbf{x}_i^\top$ and its inverse

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

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

- ii. Compute $\mathbf{q}_i = \mathbf{q}_{i-1} + y_i \mathbf{x}_i$

- Return $\mathbf{w}^* = P_n^{-1} \mathbf{q}_n$

Closed-form regression by a transformer

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

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

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 $D = [\mathbf{x}_1, y_1, \dots, \mathbf{x}_n, y_n]$, output prediction $\mathcal{A}(D)(\mathbf{x}) \in \mathbb{R}$

- **Squared prediction difference:**

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

What computation does ICL perform?

Behavioral metrics to quantify the extent two algorithms agree:

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

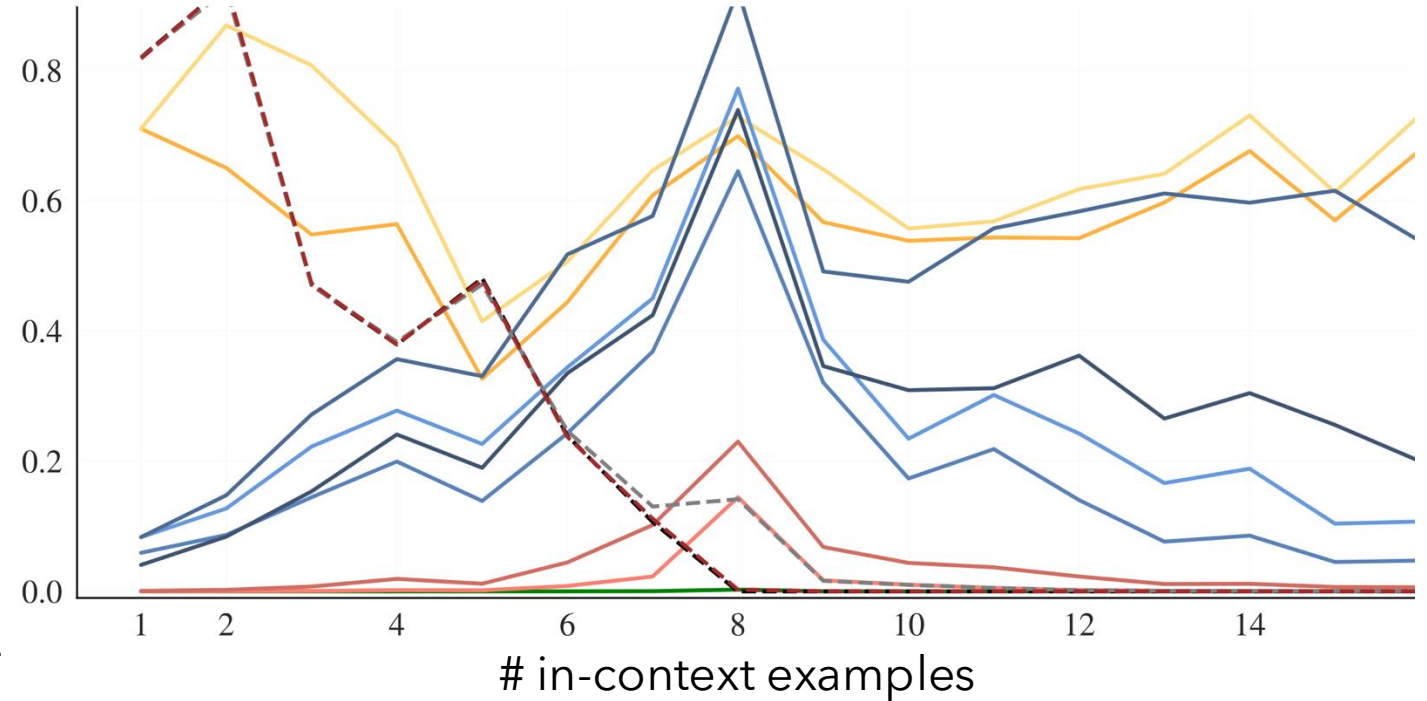
Experimental setup: Noiseless setting

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

ICL matches OLS on noiseless data

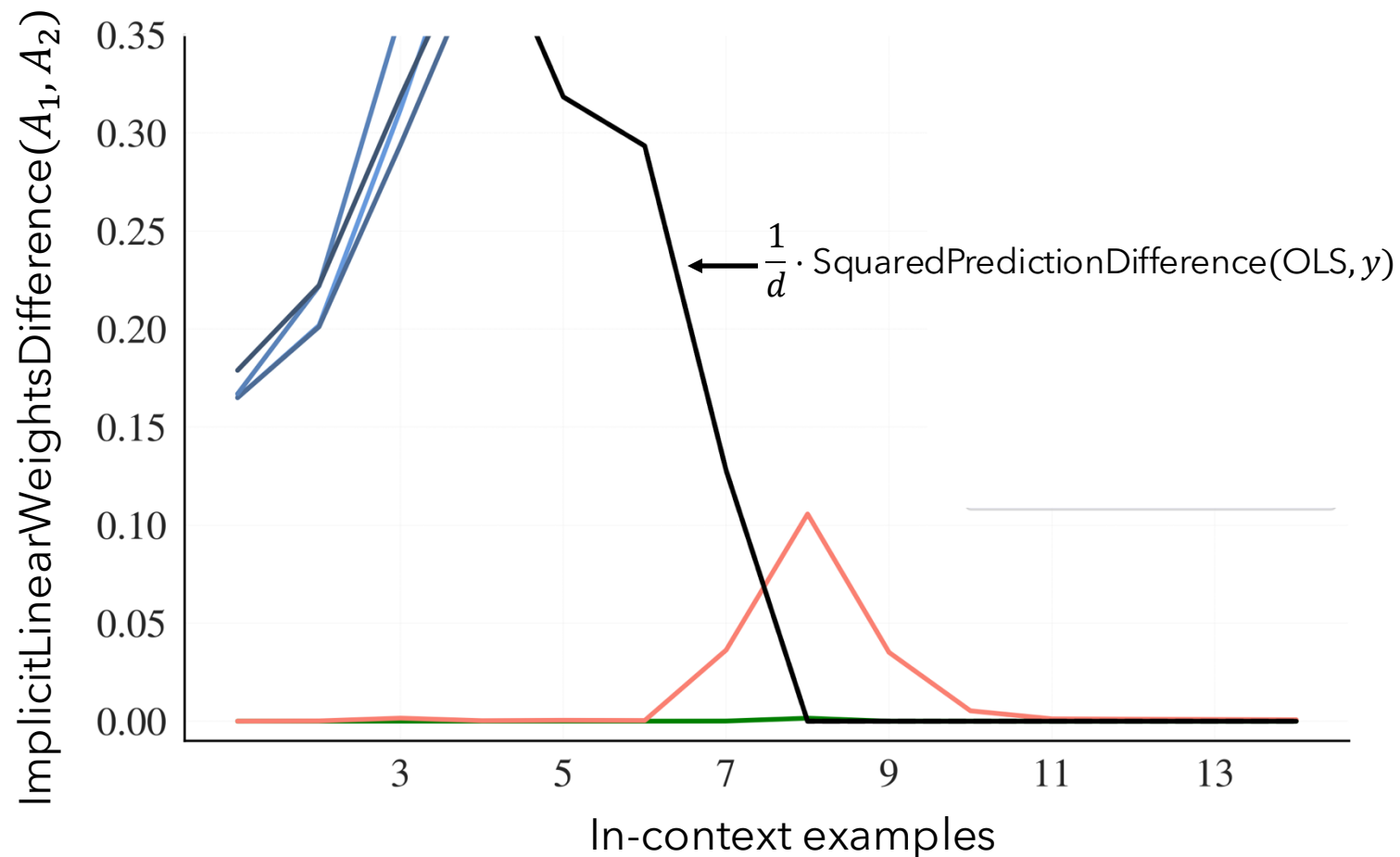
	\mathcal{A}_1	\mathcal{A}_2
■	Ordinary least squares (OLS)	Transformer
■	Ridge regression with regularization parameter $\lambda = 0.1$	Transformer
■	Ridge regression with $\lambda = 0.5$	Transformer
■	1 step of GD with learning rate $\alpha = 0.01$	Transformer
■	1 pass of SGD with $\alpha = 0.01$	Transformer
■	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

$\frac{1}{d} \cdot \text{SquaredPredictionDifference}(A_1, A_2)$



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
	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



Experimental setup: Noisy setting

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

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_{θ} encode meaningful quantities?

- What are quantities we'd expect a regression alg to compute?

- Examples: $\mathbf{w}_{OLS}, X^T \mathbf{y}$, where

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

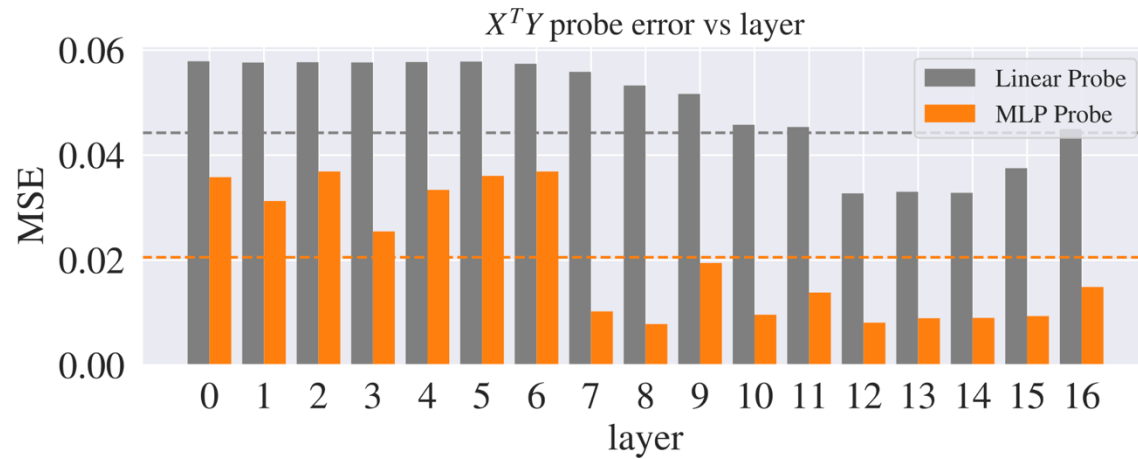
- We'll call these "probes" $\mathbf{v} \in \mathbb{R}^k$ [Alain, Bengio, '17]
- Let $H^{(\ell)}$ be the transformer's hidden states at layer ℓ
- Question: is \mathbf{v} "encoded" in $H^{(\ell)}$?
 - i.e., is it some simple function of $H^{(\ell)}$?

Does T_{θ} encode meaningful quantities?

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

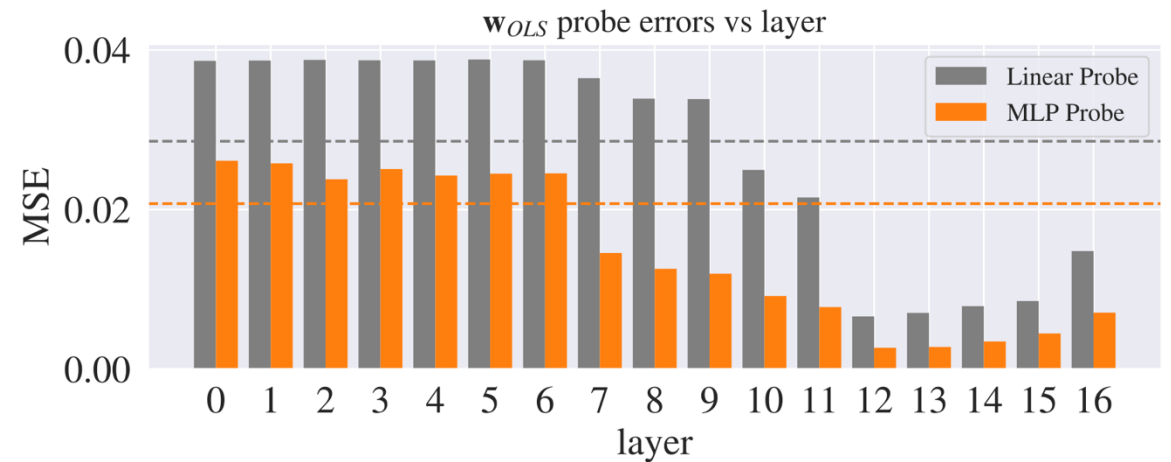
- \mathbf{s} is a learned weight vector
- f is a learned function. Two experiments:
 - f is linear
 - f is a 2-layer MLP
- Train to minimize loss $\|\mathbf{v} - \hat{\mathbf{v}}\|_2^2$
- Train a different \mathbf{s} and f for each sequence length and layer

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