Stanford MS&E 236 / CS 225: Lecture 17
Beyond discrete optimization: Transformers as algorithms

Ellen Vitercik[*]
vitercik@stanford.edu

June 12, 2024

In this class, we will investigate a surprising phenomenon: large neural sequence models can perform *in-context learning* [e.g., 1, 3, 7]. We begin by recapping the motivation from the previous class.

Let T_θ be a *trained* transformer with weights denoted by θ. A test prompt for this transformer is defined by a set of "training points" $x_1, \dots, x_N$ in some set $\mathcal{X}$, a function $f : \mathcal{X} \to \mathbb{R}$, and a test point $x' \in \mathcal{X}$. The test prompt has the following form:

$$P = \left[(x_1, f(x_1)), (x_2, f(x_2)), \dots, (x_N, f(x_N)), x'\right].$$

The transformer has already been trained, so we will not retrain it on this test prompt (hence the adjective "test"). Instead, we want to have already trained T_θ so that for any[a] such test prompt P,

$$T_\theta(P) \approx f(x'). \tag{1}$$

Intriguingly, the transformer hasn't learned a single function f, it seems to have learned an *algorithm*. For example, suppose f and f' are two linear functions that define two prompts

$$\begin{aligned} P &= \left[(\boldsymbol{x}_1, f(\boldsymbol{x}_1)), (\boldsymbol{x}_2, f(\boldsymbol{x}_2)), \dots, (\boldsymbol{x}_N, f(\boldsymbol{x}_N)), \boldsymbol{x}'\right] \\ P' &= \left[(\boldsymbol{x}_1, f'(\boldsymbol{x}_1)), (\boldsymbol{x}_2, f'(\boldsymbol{x}_2)), \dots, (\boldsymbol{x}_N, f'(\boldsymbol{x}_N)), \boldsymbol{x}'\right]. \end{aligned}$$

It is not that the transformer has learned a single weight vector $\boldsymbol{w}$ and outputs $T_\theta(P) = \boldsymbol{w}^T\boldsymbol{x}$ and $T_\theta(P') = \boldsymbol{w}^T\boldsymbol{x}'$. Clearly, Equation (1) would not be even close to satisfied for all prompts. Instead, the transformer seems to have learned a regression *algorithm*. Is it ridge regression, ordinary least-squares, or something else?

[a]This is too much to ask for, so we will discuss relaxations of this goal in this class.

[*]These notes are course material and have not undergone formal peer review. Please feel free to send me any typos or comments.

1 In-context learning of function classes

We begin with the training procedure for in-context learning. Let $\mathcal{F}$ be a class of functions mapping $\mathcal{X}$ to $\mathbb{R}$ and $\mathcal{D}_{\mathcal{F}}$ be a distribution over $\mathcal{F}$. Moreover, let $\mathcal{D}_{\mathcal{X}}$ be a distribution over inputs $\mathcal{X}$. To create training prompts, we sample $x_1, \ldots, x_{k+1} \sim \mathcal{D}_{\mathcal{X}}$ and $f \sim \mathcal{D}_{\mathcal{F}}$. A training prompt is then defined as

$$P = [x_1, f(x_1), x_2, f(x_2), \ldots, x_{k+1}, f(x_{k+1})] .$$

We use the notation $P^i = [x_1, f(x_1), x_2, f(x_2), \ldots, x_i, f(x_i), x_{i+1}]$ to denote the prefix of P of length $2i + 1$. Given a loss function $\ell(\cdot, \cdot)$, we train the transformer T_θ to predict $f(x_{i+1})$ from P^i, i.e., to minimize the expected loss

$$\mathop{\mathbb{E}}_{P}\left[\frac{1}{k+1}\sum_{i=0}^{k} \ell\left(T_\theta\left(P^i\right), f\left(x_{i+1}\right)\right)\right].$$

2 Experimental setup

In all of the experiments we describe, $\mathcal{F}$ will be a class of linear functions $\mathcal{F} = \{f : f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}, \boldsymbol{w} \in \mathbb{R}^d\}$, so the test prompts have the form

$$P = \left[\boldsymbol{x}_1, \boldsymbol{w}^T\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{w}^T\boldsymbol{x}_2, \ldots, \boldsymbol{x}_k, \boldsymbol{w}^T\boldsymbol{x}_k, \boldsymbol{x}_{k+1}\right] .$$

We can think of

$$S = \left\{\left(\boldsymbol{x}_1, \boldsymbol{w}^T\boldsymbol{x}_1\right), \ldots, \left(\boldsymbol{x}_k, \boldsymbol{w}^T\boldsymbol{x}_k\right)\right\} \tag{2}$$

as the "training set" encoded in the prompt. As such, we can view any regression algorithm $\mathcal{A}$ as a function $\mathcal{A}(S)(\boldsymbol{x}_{k+1})$ where, ideally, $\mathcal{A}(S)(\boldsymbol{x}_{k+1}) \approx \boldsymbol{w}^T\boldsymbol{x}_{k+1}$. To unify notation, we denote transformer's output as $T_\theta(S)\left(\boldsymbol{x}_{k+1}\right)$ as well.

2.1 Within-distribution experiments

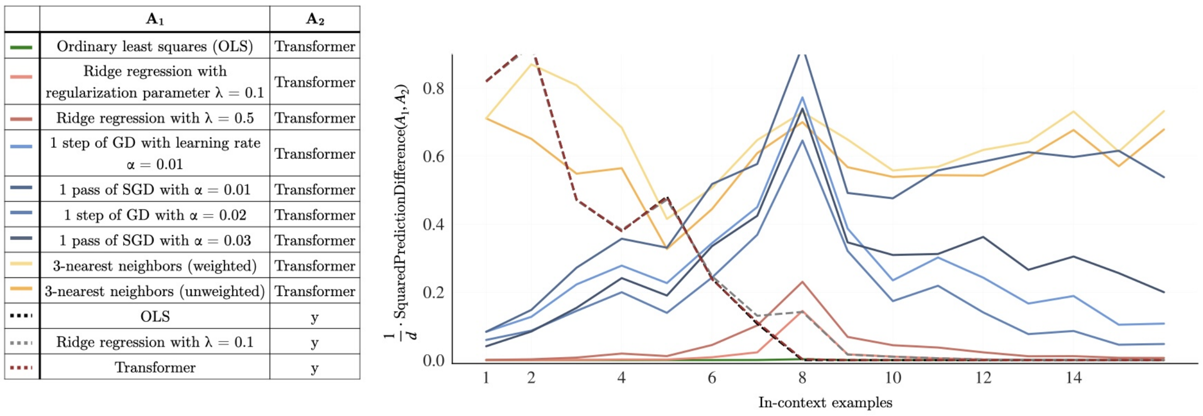
We begin by describing within-distribution experiments by Akyürek et al. [1], under which the in-context examples encoded in the test prompt, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$, have the same distribution as the test point $\boldsymbol{x}_{k+1}$. In these experiments, $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{k+1}, \boldsymbol{w} \sim N(\boldsymbol{0}, I_d)$ are Normally distributed with $d = 8$.

2.1.1 Squared prediction difference

In the first set of experiments, Akyürek et al. [1] compare the transformer's prediction to a variety of different standard regression algorithms, including ordinary least squares, ridge regression, and others. To compare two algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$, Akyürek et al. [1] use the squared difference between their predictions,

$$\mathsf{SquaredPredictionDifference}(\mathcal{A}_1, \mathcal{A}_2) = \mathop{\mathbb{E}}_{\boldsymbol{w}, S, \boldsymbol{x}_{k+1}}\left[\left(\mathcal{A}_1(S)\left(\boldsymbol{x}_{k+1}\right) - \mathcal{A}_2(S)\left(\boldsymbol{x}_{k+1}\right)\right)^2\right].$$

The results are in Figure 1. The solid lines compare the transformer to a variety of different



	A₁	A₂
━	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

$\frac{1}{d}$ · SquaredPredictionDifference($\mathcal{A}_1, \mathcal{A}_2$)
0.0
0.2
0.4
0.6
0.8
1
2
4
6
8
10
12
14
In-context examples

Figure 1: Results by Akyürek et al. [1], explained in Section 2.1.1.

regression algorithms, whereas the dotted lines illustrate the mean squared error of the transformer, OLS, and ridge regression:

$$\mathop{\mathbb{E}}_{\boldsymbol{w},S,\boldsymbol{x}_{k+1}}\left[\left(\mathcal{A}(S)\left(\boldsymbol{x}_{k+1}\right)-\boldsymbol{w}^T\boldsymbol{x}_{k+1}\right)^2\right].$$

We can see that the transformer closely tracks the performance of OLS, both in terms of the squared prediction difference and the mean squared error.

2.1.2 Implicit linear weights difference

Although the transformer does not necessarily learn a linear weights vector, we can still try to understand the linear function that best fits the transformer's predictions. To do so, in addition to sampling the in-context examples S from Equation (2), we will also sample a set $S' = \{\boldsymbol{x}'_1, \ldots, \boldsymbol{x}'_m\} \sim N(\mathbf{0}, I_d)$ of test points. We define the *implicit weights* of algorithm $\mathcal{A}$ (which may be the transformer) to be those that minimize the squared difference between the implicit weights evaluated on the test set and the predictions of $\mathcal{A}(S)$ on the test set:

$$\hat{\boldsymbol{w}}_{\mathcal{A}} = \text{argmin}_{\hat{\boldsymbol{w}}}\sum_{i=1}^{m}\left(\hat{\boldsymbol{w}}^T\boldsymbol{x}'_i - \mathcal{A}(S)\left(\boldsymbol{x}'_i\right)\right)^2.$$

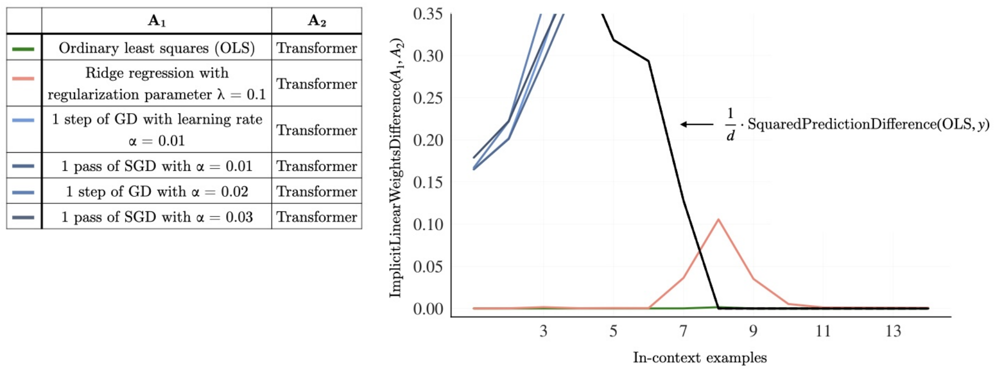
To compare two algorithms, we compare their implicit linear weights:

$$\mathsf{ImplicitLinearWeightsDifference}(\mathcal{A}_1, \mathcal{A}_2) = \mathop{\mathbb{E}}_{\boldsymbol{w},S,S'}\left[\|\hat{\boldsymbol{w}}_{\mathcal{A}_1} - \hat{\boldsymbol{w}}_{\mathcal{A}_2}\|_2^2\right].$$

(If $\mathcal{A}_1$ or $\mathcal{A}_2$ is a standard regression algorithm that returns linear weights, we use those instead of $\hat{\boldsymbol{w}}_{\mathcal{A}_1}$ or $\hat{\boldsymbol{w}}_{\mathcal{A}_2}$ in the definition above.) The results are illustrated in Figure 2. In this case, the transformer's implicit weights closely match OLS's.

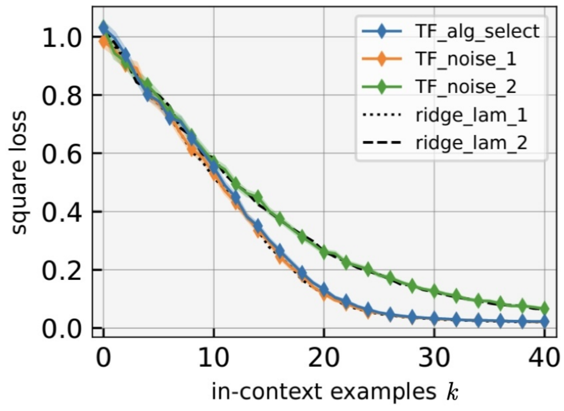
2.2 Algorithm selection on noisy data

When noise is added to the training set, the best choice of a regression algorithm depends on the noise's magnitude. A statistician would need to make this choice based on trial and error.

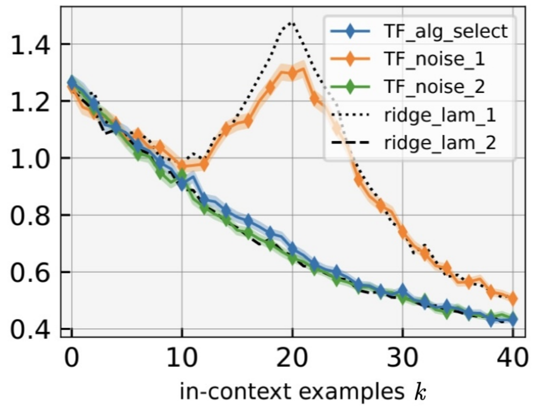


	A_1	A_2
▬	Ordinary least squares (OLS)	Transformer
▬	Ridge regression with regularization parameter $\lambda = 0.1$	Transformer
▬	1 step of GD with learning rate $\alpha = 0.01$	Transformer
▬	1 pass of SGD with $\alpha = 0.01$	Transformer
▬	1 step of GD with $\alpha = 0.02$	Transformer
▬	1 pass of SGD with $\alpha = 0.03$	Transformer

Figure 2: Results by Akyürek et al. [1], explained in Section 2.1.2. The black line is the mean squared error of OLS.



(a) Noise level $\sigma = 0.1$.



(b) Noise level $\sigma = 0.5$.

Figure 3: Results by Bai et al. [3], explained in Section 2.2.

Bai et al. [3] show that transformers seamlessly adapt to the noise level in the in-context training set, seeming to perform algorithm selection on the fly.

In the experiments by Bai et al. [3], $\boldsymbol{w} \sim N(\mathbf{0}, I_d/d)$ and $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k+1} \sim N(\mathbf{0}, I_d)$ for $d = 20$. We add noise to the in-context training set, defining it as

$$S = \left\{ \left(\boldsymbol{x}_1, \boldsymbol{w}^T \boldsymbol{x}_1 + \sigma\epsilon_1\right), \ldots, \left(\boldsymbol{x}_k, \boldsymbol{w}^T \boldsymbol{x}_k + \sigma\epsilon_k\right) \right\}$$

with $\epsilon_i \sim N(0, 1)$ and $\sigma \geq 0$. In this setting, ridge regression with regularization parameter

$$\lambda = \frac{d\sigma^2}{k} \tag{3}$$

is known to return the *minimum Bayes risk predictor*. That is, its predictions are roughly equal to $\mathbb{E}[y \mid S, \boldsymbol{x}_{k+1}]$.

The results by Bai et al. [3] are in Figure 3. In Figure 3a, $\sigma = 0.1$ and in Figure 3b, $\sigma = 0.5$. In both figures, the orange lines are the mean squared error

$$\mathsf{SquaredLoss}(\mathcal{A}) = \mathop{\mathbb{E}}_{\boldsymbol{w}, S, \boldsymbol{x}_{k+1}} \left[\left(\mathcal{A}(S)(\boldsymbol{x}_{k+1}) - \boldsymbol{w}^T \boldsymbol{x}_{k+1}\right)^2 \right]$$

of a transformer trained on training prompts with $\sigma = 0.1$. Meanwhile, the green lines are the mean squared error of a transformer trained with $\sigma = 0.5$. The black dotted lines are the mean squared error of ridge regression with the optimal choice of λ for $\sigma = 0.1$ according to Equation (3) (as a function of k). We can see that the orange lines (the transformer trained with $\sigma = 0.1$) closely match the dotted lines. Meanwhile, the dashed lines are the mean squared error of ridge regression with the optimal choice of λ for $\sigma = 0.5$. In this case, the green lines (the transformer trained with $\sigma = 0.5$) closely match the dashed lines. Finally, the blue lines show the mean squared error of a transformer trained with a mixture of noise levels. In this case, the transformer is able to (nearly) match the better of all baselines. Intriguingly, the transformer acts as if it is performing algorithm selection.

2.3 Does the transformer encode meaningful intermediate quantities?

Finally, we ask if it is possible to tell whether the transformer is encoding meaningful quantities in its hidden embeddings. What could these meaningful quantities be? Examples might include the final weight vector $\boldsymbol{w}_{OLS}$ of OLS, or $X^T\boldsymbol{y}$, where

$$X = \begin{pmatrix} | & & | \\ \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_k \\ | & & | \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{y} = \begin{pmatrix} \boldsymbol{w}^T\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{w}^T\boldsymbol{x}_k \end{pmatrix}.$$

We call these two quantities *probes* $\boldsymbol{v} \in \mathbb{R}^d$, *a la* Alain and Bengio [2]. Let $H^{(\ell)} \in \mathbb{R}^{(\text{sequence length})\times D}$ be the transformer's output at layer ℓ. Our goal is to answer the question: is $\boldsymbol{v}$ "encoded" in $H^{(\ell)}$? Said another way, is $\boldsymbol{v}$ some simple function of $H^{(\ell)}$?

To answer this question, we define a *probing model* $\hat{\boldsymbol{v}} = f(\boldsymbol{s}^T H^{(\ell)})$ where $\boldsymbol{s} \in \mathbb{R}^{(\text{seq. length})}$ is a learned weight vector (with $\boldsymbol{s} \geq 0$ and $\|s\|_1 = 1$) and $f : \mathbb{R}^D \to \mathbb{R}^d$ is a learned function. Akyürek et al. [1] perform two experiments: one where f is a linear function (which would correspond to as simple an encoding as we could hope for) and another where f is a 2-layer multi-layer perceptron (MLP). We then train $\boldsymbol{s}$ and f to minimize the mean squared error

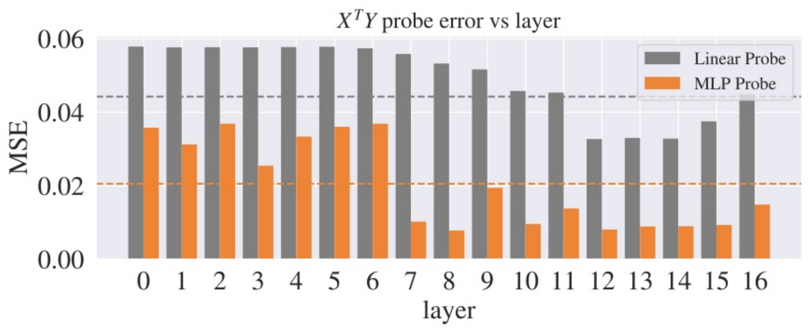
$$\|\boldsymbol{v} - \hat{\boldsymbol{v}}\|_2^2 . \tag{4}$$

We train a different $\boldsymbol{s}$ and f for each sequence length and layer. Intuitively, if this error is small for some layer ℓ, then $\boldsymbol{v}$ is (approximately) a simple function of $H^{(\ell)}$.

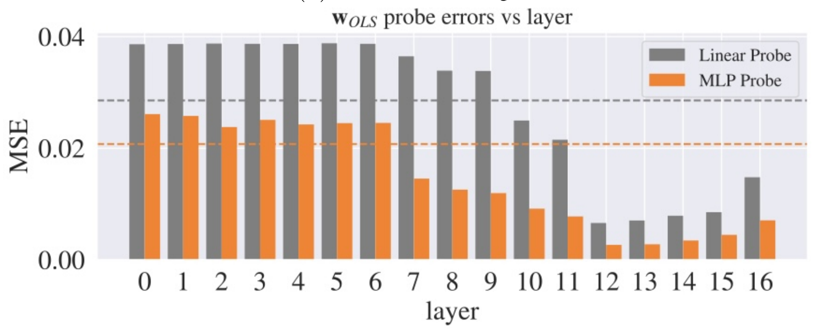
The results of probing experiments by Akyürek et al. [1] are in Figure 4. In these experiments, $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{k+1}, \boldsymbol{w} \sim N(\boldsymbol{0}, I_d)$ with $d = 8$. We can see that the test mean squared error (Equation (4)) of the linear probe is worse in both figures compared to the MLP probe, indicating that the probes are more likely to be encoded by a non-linear than a linear function. Around the seventh layer, $X^T\boldsymbol{y}$ (or something like it) appears to be computed first. Meanwhile, $\boldsymbol{w}_{OLS}$ seems to be computed around the twelfth layer.

3 Decision trees

Finally, transformers seem able to in-context learn non-linear functions, such as decision trees [7]. To create a prompt, Garg et al. [7] draw $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k+1} \sim N(\boldsymbol{0}, I_d)$ with $d = 20$ and label each point $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ by a full binary, depth-four decision tree, denoted f. Each node of the decision tree is defined by an inequality $x[j] \leq \theta$, where $j \sim \text{Unif}(\{1, 2, \ldots, 20\})$ and



(a) Probe for $X^T\boldsymbol{y}$.



(b) Probe for $\boldsymbol{w}_{OLS}$.

Figure 4: Results by Akyürek et al. [1], explained in Section 2.3.

each $\theta \sim N(0,1)$. The values associated with the leaf nodes are drawn from $N(0,1)$. The results by Garg et al. [7] are in Figure 5. In this figure, the transformer's performance is better than greedy tree learning and boosting (via XGBoost [6]). Garg et al. [7] include the following intriguing remark: "in general, we do not have a good understanding of the space of efficient algorithms for learning decision trees, and the conditions under which known heuristics work [4, 5]. At the same time, we found that Transformers can be trained to directly discover such an algorithm for the prompt distribution we considered. This suggests an intriguing possibility where we might be able to reverse engineer the algorithm encoded by a Transformer to obtain new sample efficient algorithms for existing learning problems."

References

[1] Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

[2] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

[3] Yu Bai, Fan Chen, Huan Wang, Caiming Xiong, and Song Mei. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[4] Guy Blanc, Jane Lange, Mingda Qiao, and Li-Yang Tan. Decision tree heuristics can

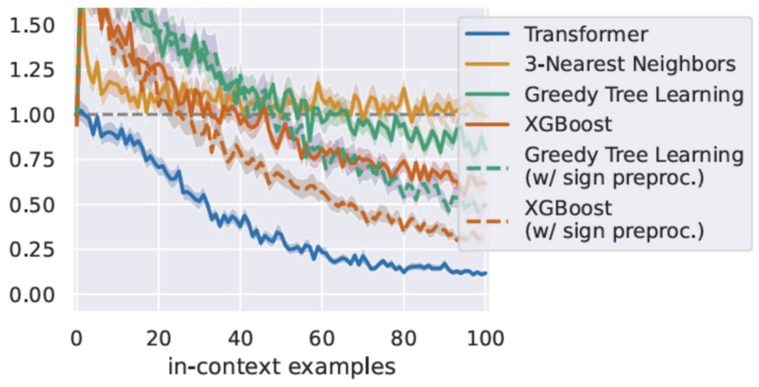


Figure 5: Results by Garg et al. [7], explained in Section 3.

fail, even in the smoothed setting. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, 2021.

[5] Alon Brutzkus, Amit Daniely, and Eran Malach. Id3 learns juntas for smoothed product distributions. In *Conference on Learning Theory (COLT)*, 2020.

[6] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016.

[7] Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.