NEURAL INFORMATION PROCESSING SYSTEMS

USC University of Southern California

Transformers Learn to Achieve Second-Order Convergence Rates for In-Context Linear Regression




Deqing Fu   Tian-Qi Chen   Robin Jia   Vatsal Sharan

How do Models do In-Context Learning?

Transformers Learn In-Context by Gradient Descent

Johannes von Oswald [1 2] **Eyvind Niklasson** [2] **Ettore Randazzo** [2] **João Sacramento** [1]
Alexander Mordvintsev [2] **Andrey Zhmoginov** [2] **Max Vladymyrov** [2]

Transformers learn to implement preconditioned gradient descent for in-context learning

Kwangjun Ahn* MIT EECS/LIDS kjahn@mit.edu
Xiang Cheng* MIT LIDS chengx@mit.edu
Hadi Daneshmand* MIT LIDS/FODSI hdanesh@mit.edu
Suvrit Sra TU Munich / MIT suvrit@mit.edu

One Step of Gradient Descent is Provably the Optimal In-Context Learner with One Layer of Linear Self-Attention

Arvind Mahankali, Stanford University, amahanka@stanford.edu
Tatsunori B. Hashimoto, Stanford University, thashim@stanford.edu
Tengyu Ma, Stanford University, tengyuma@stanford.edu

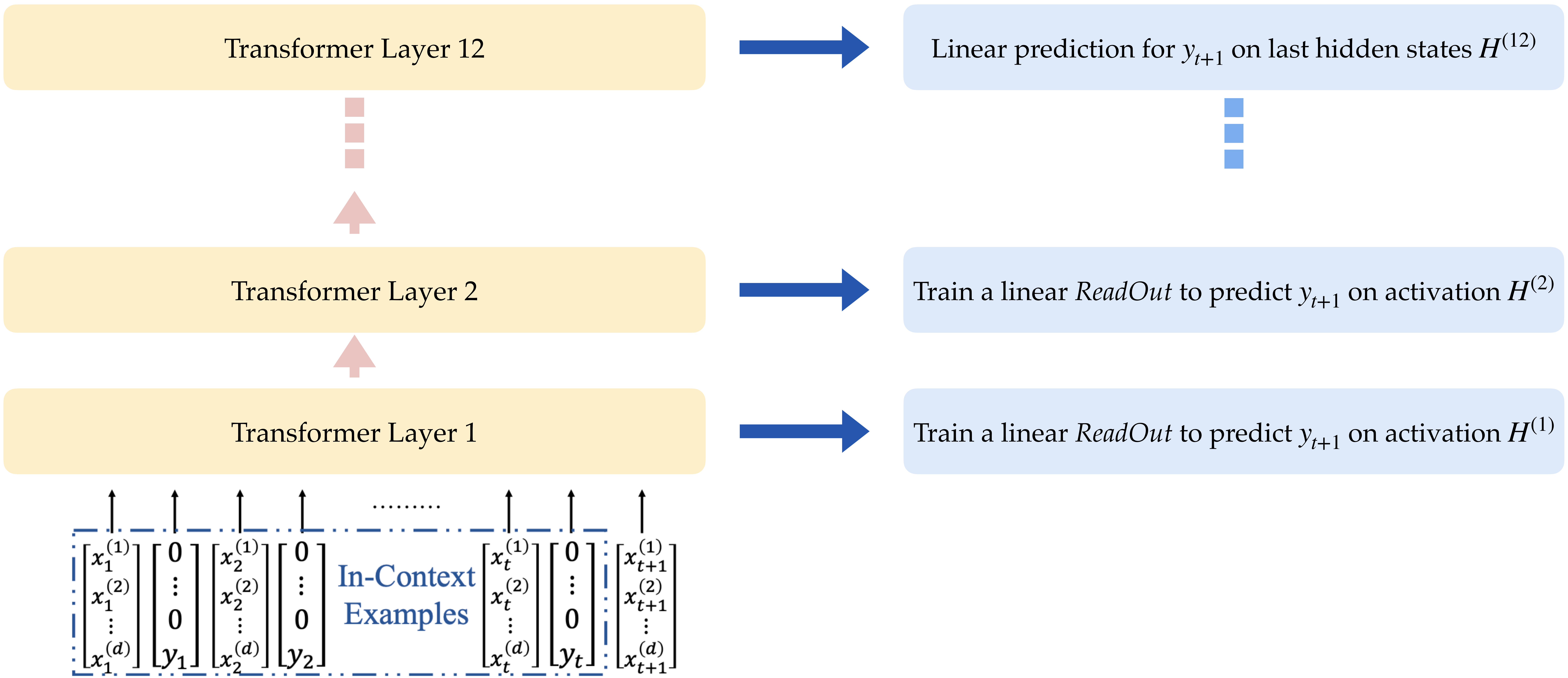
Why Can GPT Learn In-Context?
Language Models Implicitly Perform Gradient Descent as Meta-Optimizers

Damai Dai[†]*, **Yutao Sun**[‖]*, **Li Dong**[‡], **Yaru Hao**[‡], **Shuming Ma**[‡], **Zhifang Sui**[†], **Furu Wei**[‡]
[†] MOE Key Lab of Computational Linguistics, Peking University
[‖] Tsinghua University [‡] Microsoft Research

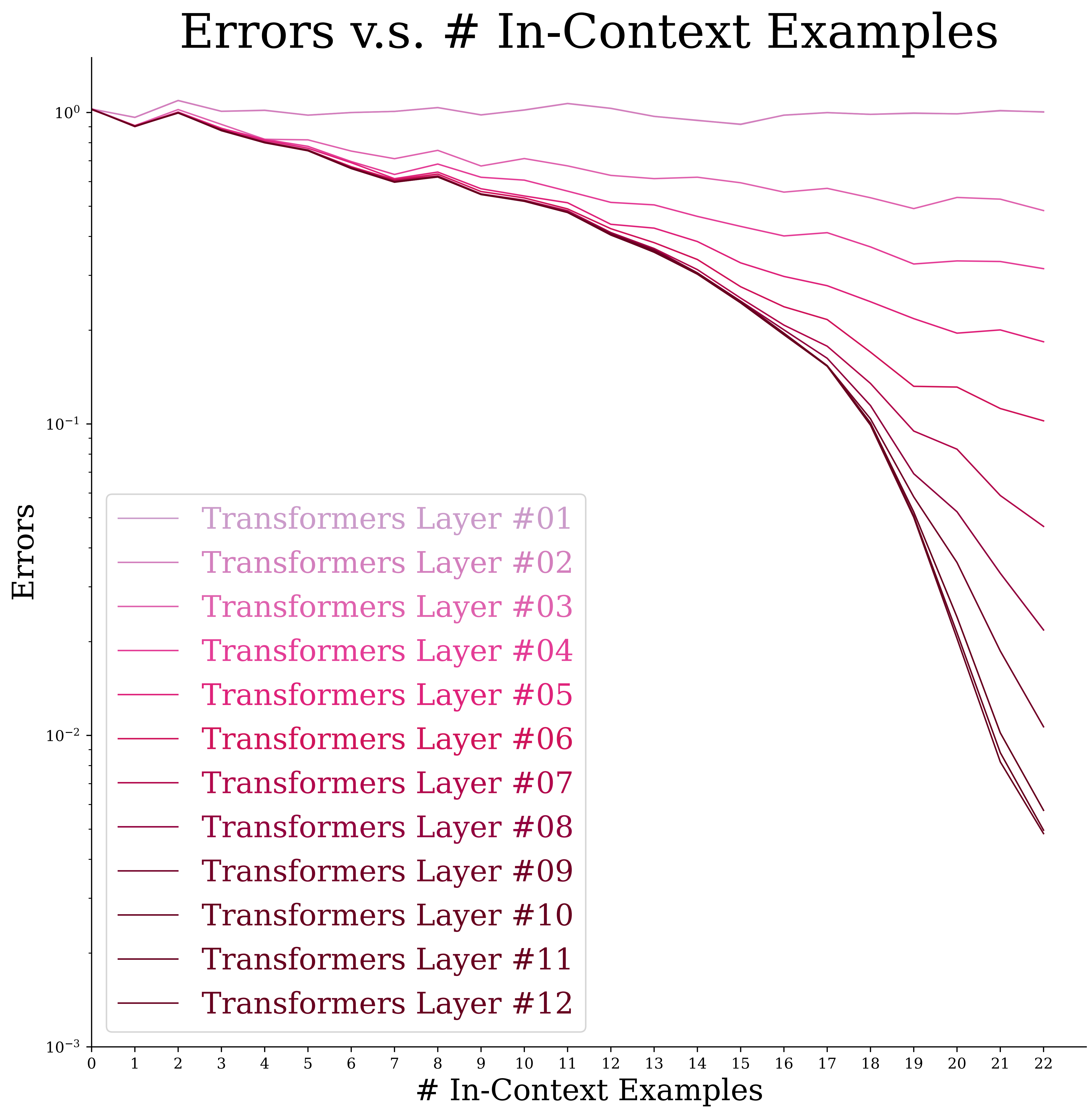
How do Models do In-Context Learning?

Do Transformers really **learn to implement gradient descent** *for ICL?*

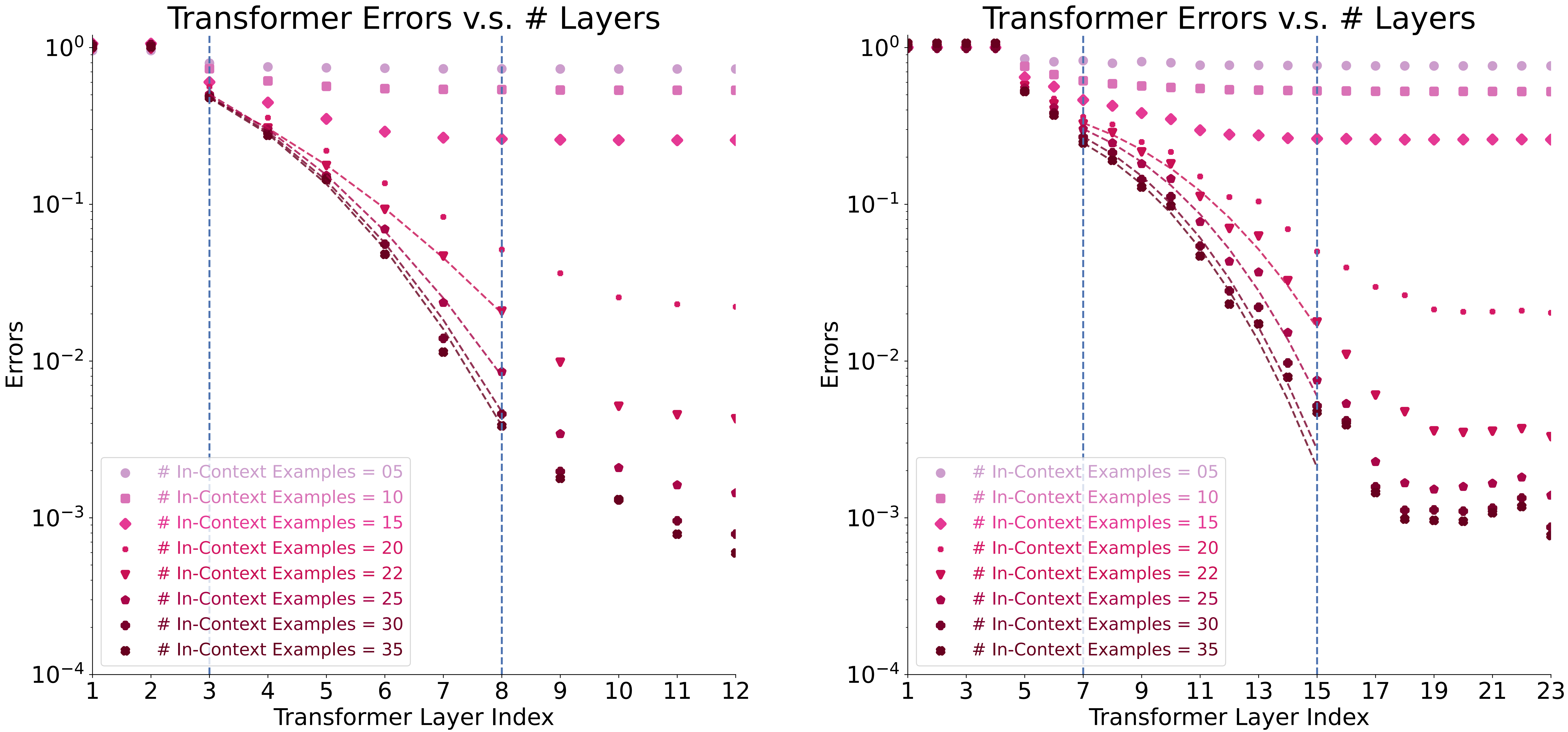
Claim 1: Transformers as Iterative Algorithms



Claim 1: Transformers as Iterative Algorithms



Claim 2: Transformers Learn to Achieve *Second-Order* Convergence Rates



Preliminaries: Known Algorithms

- **Ordinary Least Squares**

This method finds the minimum-norm solution to the objective:

$$\mathscr{L}(w \mid X, y) = \frac{1}{2n}\|y - Xw\|_2^2.$$

The Ordinary Least Squares (OLS) solution has a closed form given by the Normal Equations:

$$\hat{w}^{\text{OLS}} = (X^\top X)^\dagger X^\top y$$

where we denote $S := X^\top X$ and $S^\dagger$ is the pseudo-inverse S.

Preliminaries: Known Algorithms

- **Gradient Descent**

Gradient descent (GD) finds the weight vector $\hat{w}^{\mathrm{GD}}$ with initialization $\hat{w}_0^{\mathrm{GD}} = 0$ and using the iterative update rule:

$$\hat{w}_{k+1}^{\mathrm{GD}} = \hat{w}_k^{\mathrm{GD}} - \eta \nabla_w \mathscr{L}(\hat{w}_k^{\mathrm{GD}} \mid X, y)$$

It is known that Gradient Descent requires $\mathcal{O}\left(\kappa(S)\log(1/\epsilon)\right)$ steps to converge to an ϵ error where $\kappa(S) = \dfrac{\lambda_{\max}(S)}{\lambda_{\min}(S)}$ is the *condition number*.

Preliminaries: Known Algorithms

- **Iterative Newton's Method**

This method finds the weight vector $\hat{w}^{\text{Newton}}$ by iteratively apply Newton's method to finding the pseudo inverse of $S = X^\top X$.

$$M_0 = \alpha S, \text{ where } \alpha = \frac{2}{\|SS^\top\|_2}, \quad \hat{w}_0^{\text{Newton}} = M_0 X^\top y,$$

$$M_{k+1} = 2M_k - M_k S M_k, \quad \hat{w}_{k+1}^{\text{Newton}} = M_{k+1} X^\top y\,.$$

This computes an approximation of the pseudo inverse using the moments of $S = X^\top X$. In contrast to GD, the Newton's method only requires $\mathcal{O}(\log \kappa(S) + \log\log(1/\epsilon))$ steps to converge. Note that this is *exponentially faster* than the convergence rate of GD.

Metric: Similarity of Errors

Measuring "Similarity" of Two Algorithms

$$x_1, \quad y_1, \quad x_2, \quad y_2, \quad x_3, \quad y_3, \quad \cdots, \quad x_t, \quad y_t$$

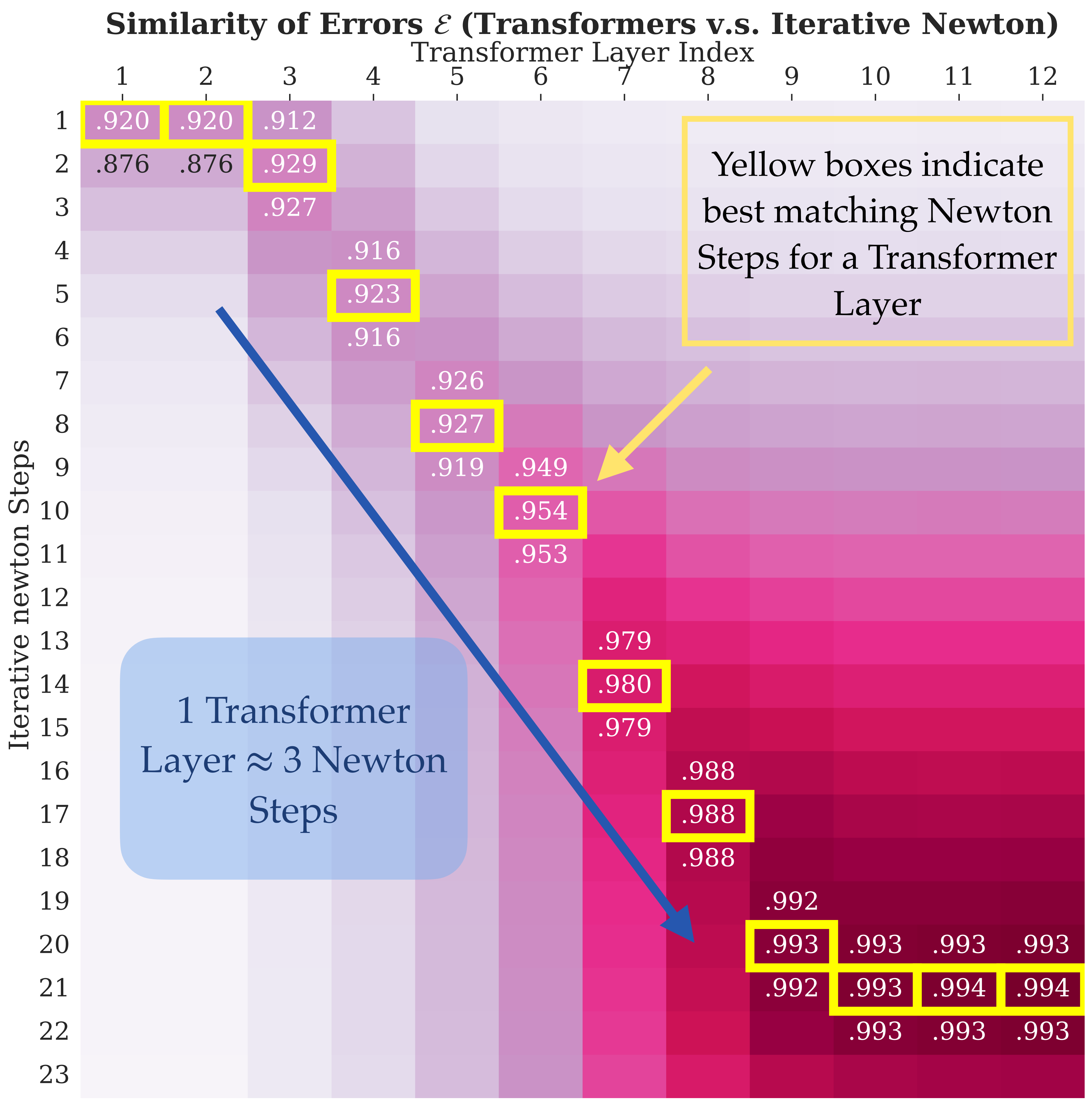
Algorithm A	$y_1^A,$	$y_2^A,$	$y_3^A,$	$\cdots$	y_t^A
Algorithm B	$y_1^B,$	$y_2^B,$	$y_3^B,$	$\cdots$	y_t^B

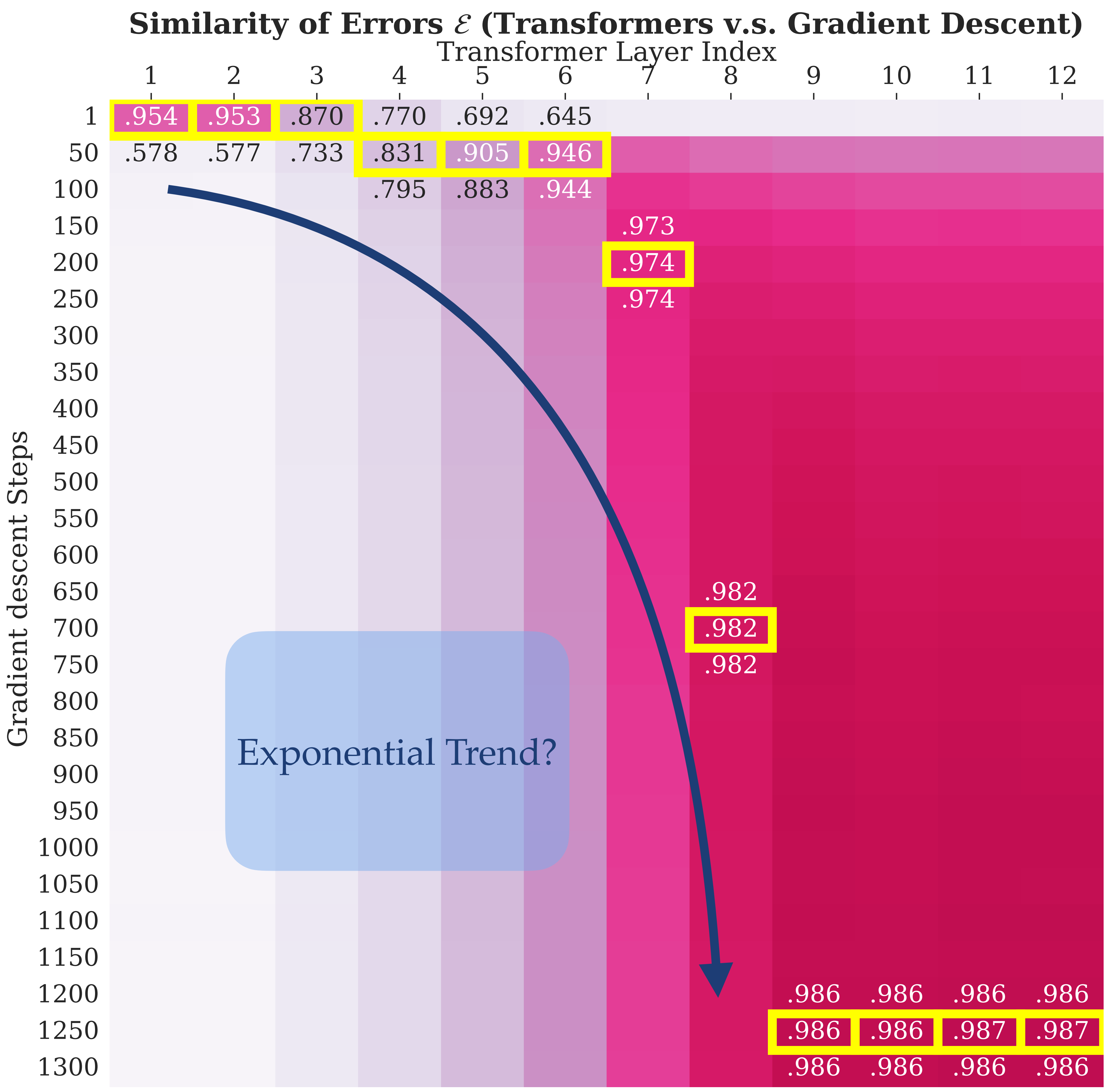
Residual A $(y_1^A - y_1), \ (y_2^A - y_2), (y_3^A - y_3), \ \cdots (y_t^A - y_t)$

Residual B $(y_1^B - y_1), \ (y_2^B - y_2), (y_3^B - y_3), \ \cdots (y_t^B - y_t)$

Overall Similarity of Errors between A and B =
$\mathbb{E}$ [Cosine Similarity Between Residuals of A and B]

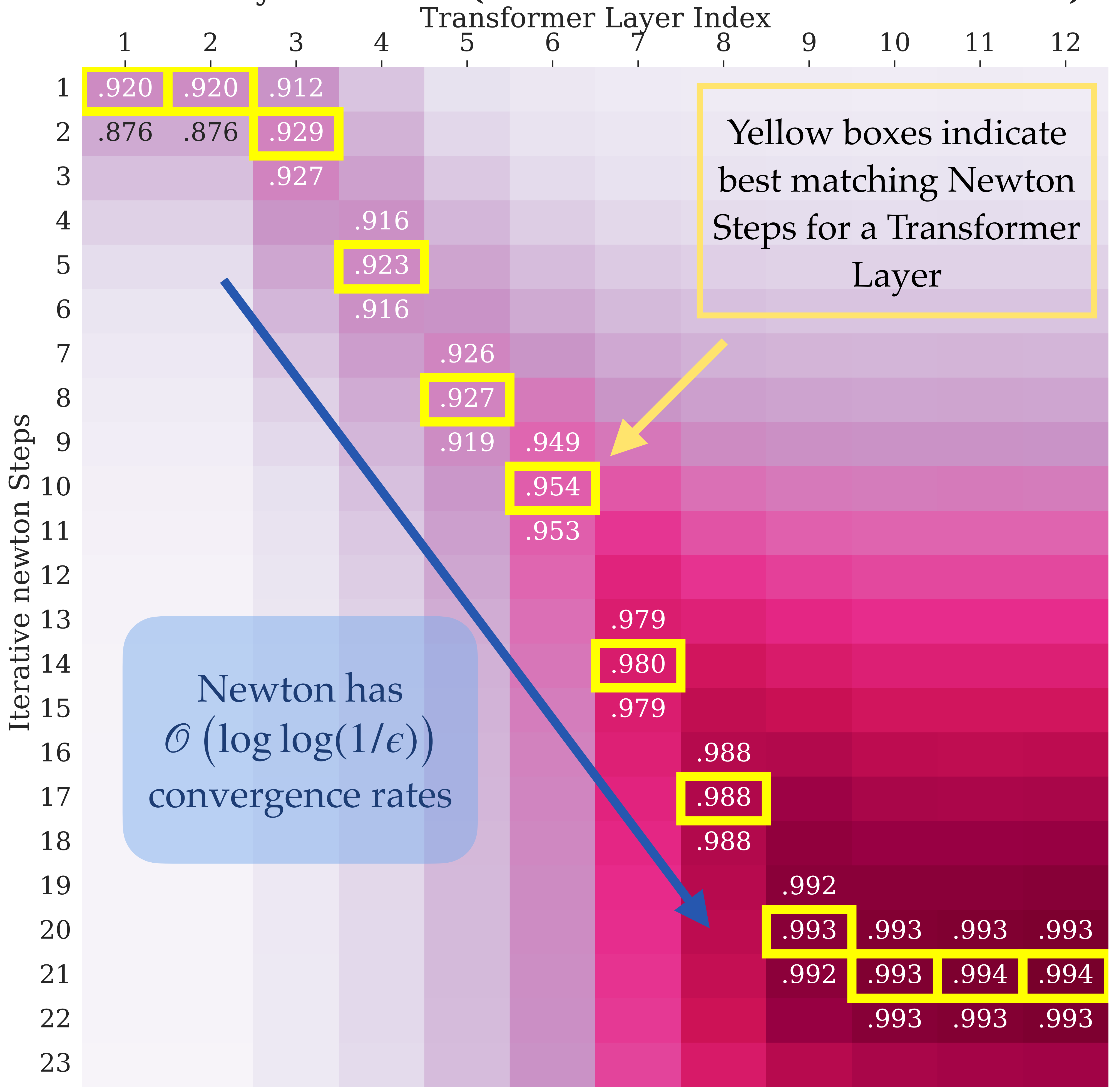
Claim 2: Transformers Learn to Achieve *Second-Order* Convergence Rates



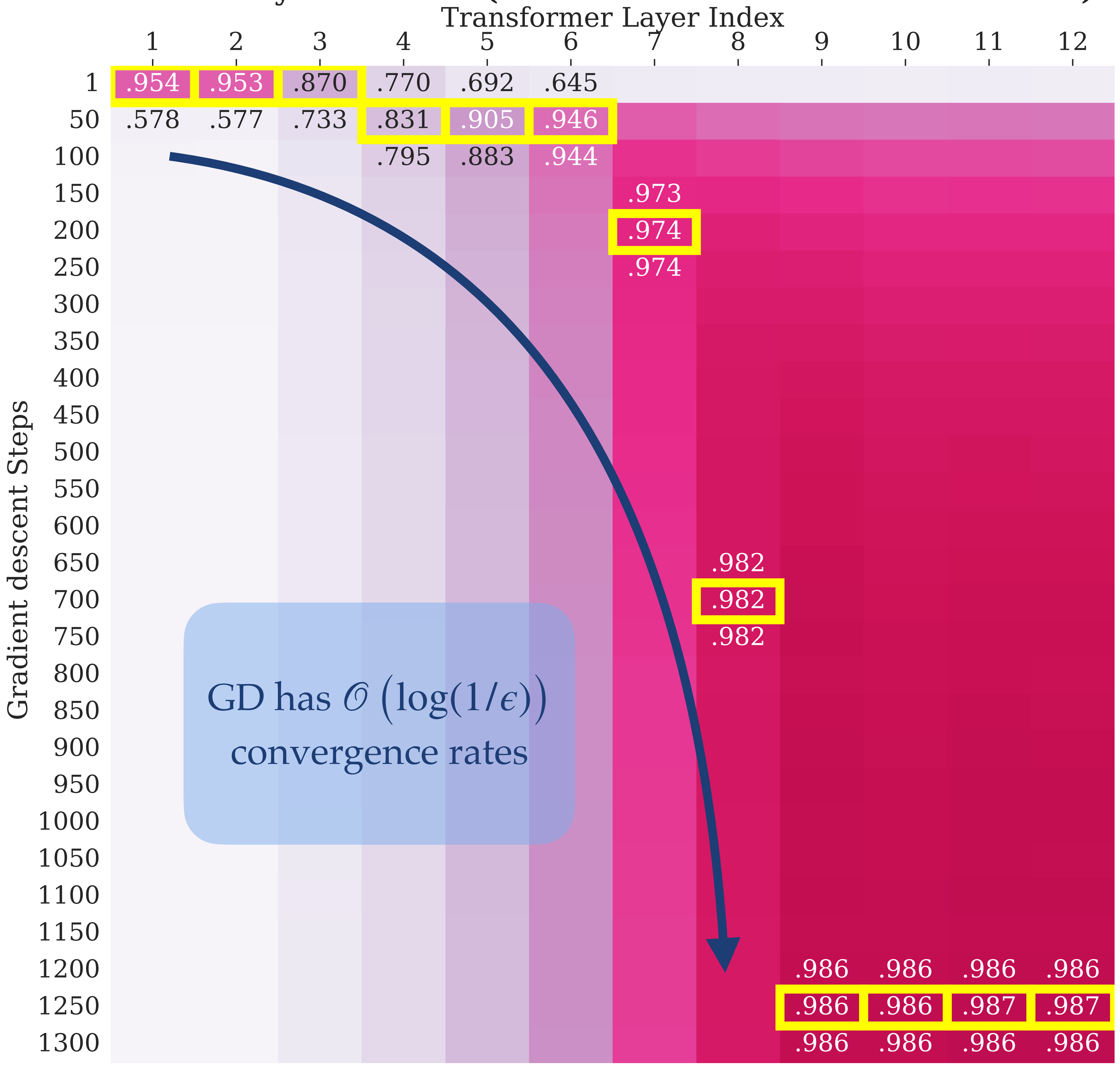


Claim 2: Transformers Learn to Achieve *Second-Order* Convergence Rates

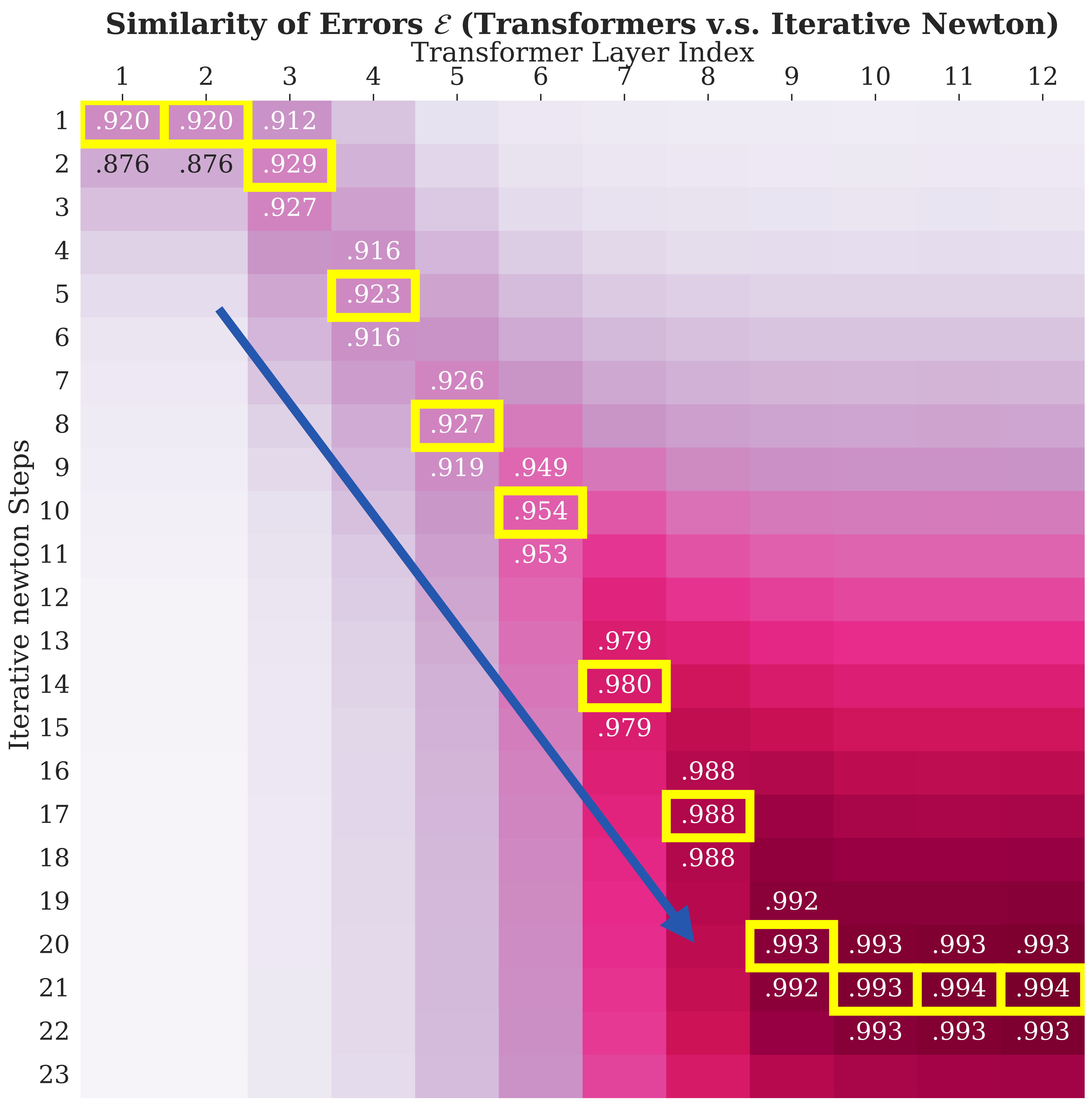
Similarity of Errors $\mathcal{E}$ (Transformers v.s. Iterative Newton)

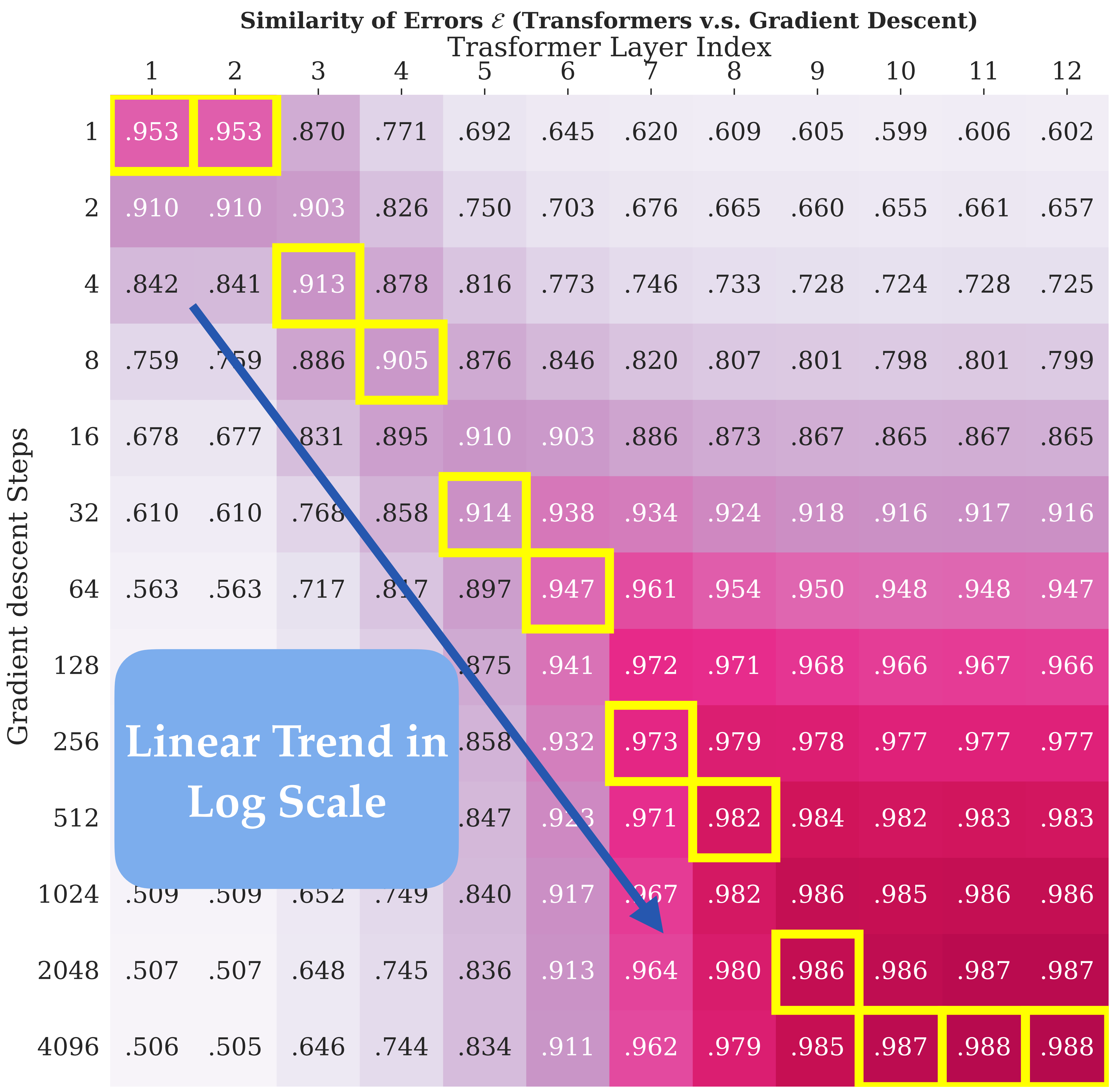


Similarity of Errors $\mathcal{E}$ (Transformers v.s. Gradient Descent)

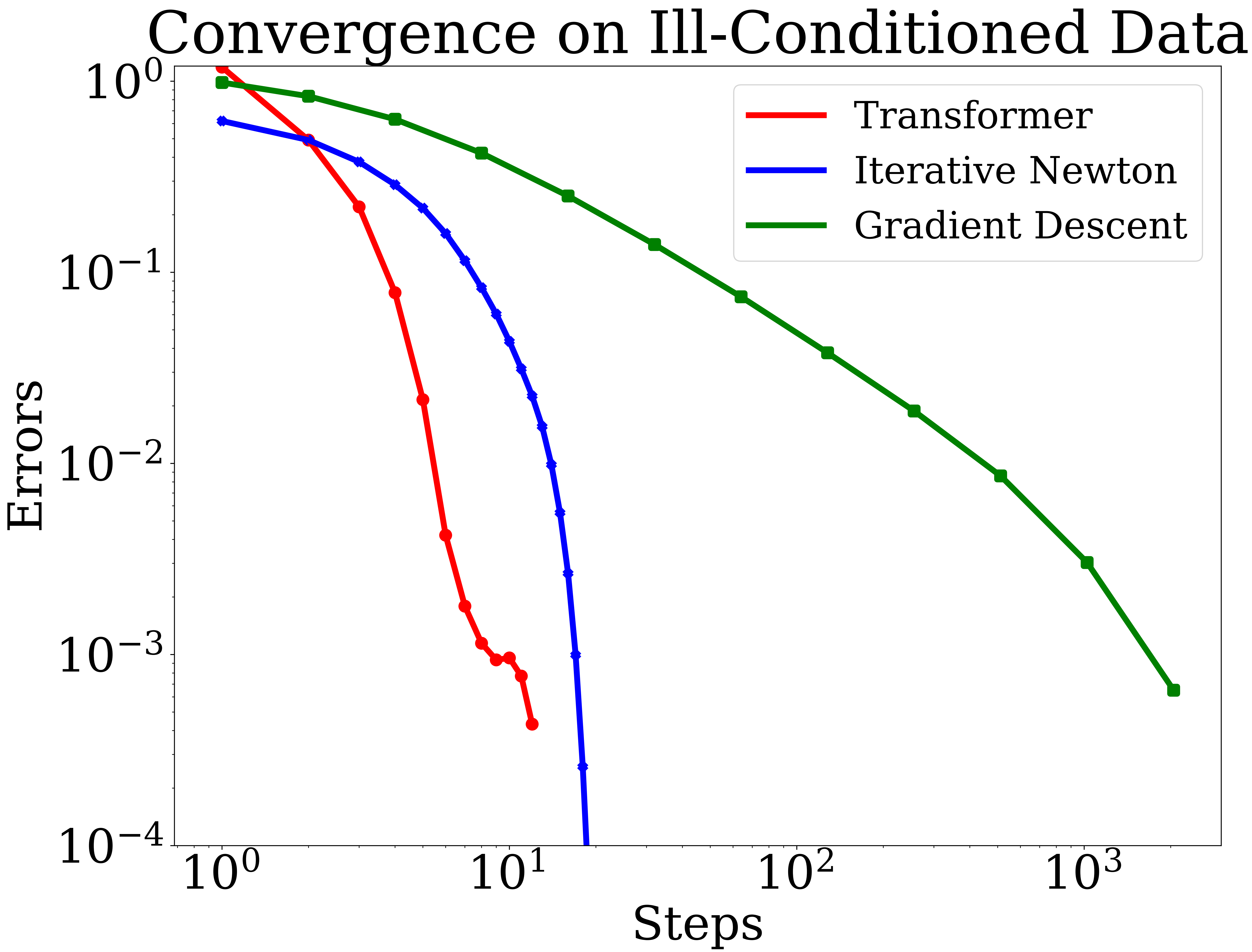


Claim 2: Transformers Learn to Achieve *Second-Order* Convergence Rates





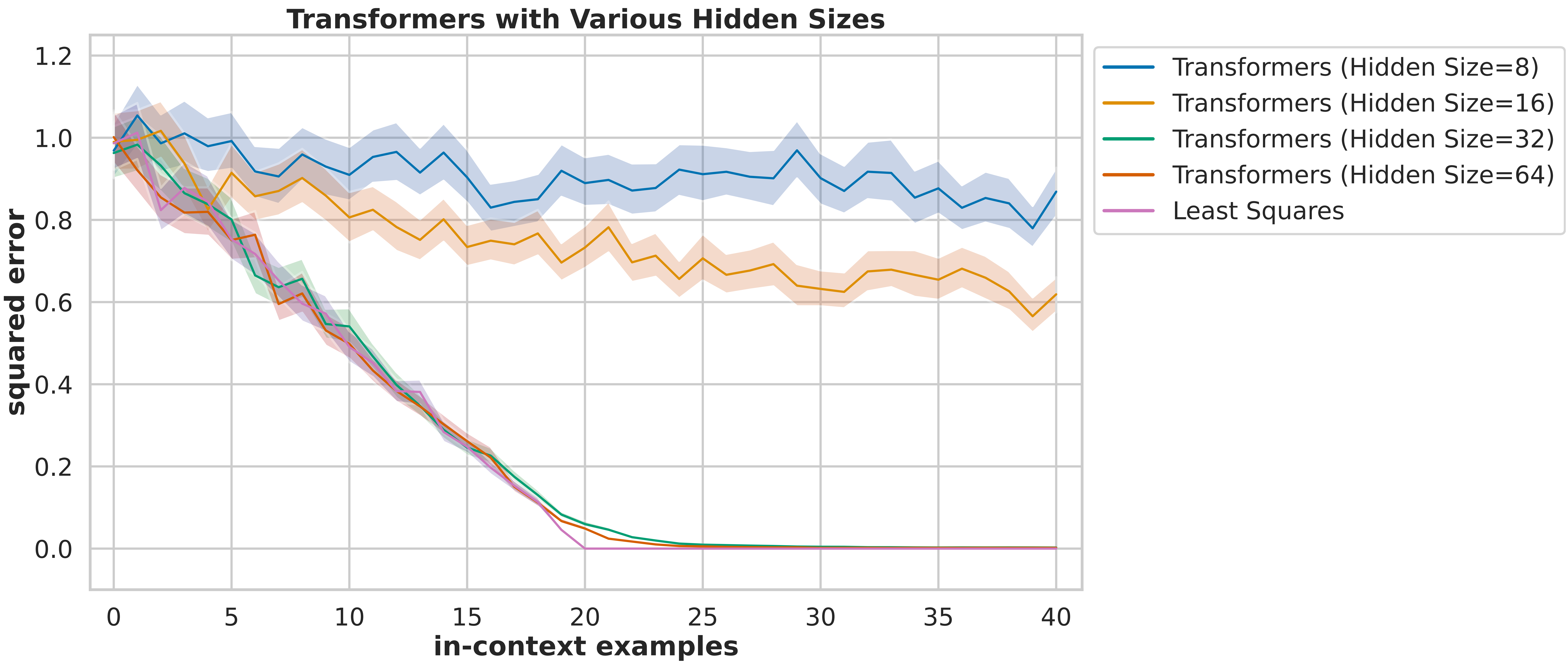
Claim 3: Transformer can still match Newton on Ill-Conditioned Case



Rate of Convergence

Algorithm	**Steps Required for Convergence**	**Algorithm Category**
Gradient Descent	$GD = O(\kappa \log(1/\epsilon))$	First-Order
Iterative Newton	$IN = O(\log \kappa + \log \log(1/\epsilon)) = \log (GD)$	Second-Order
Transformers	$TF \approx IN = \log (GD)$	Second-Order

Claim 4: Transformers Require $\mathcal{O}(d)$ Hidden Size



Theoretical Justification

Can Transformers actually represent as complicated of a method as Iterative Newton with only polynomially many layers?

Theoretical Justification

- **Theorem (Transformer as Newton's Method)**

There exist Transformer weights such that on any set of in-context examples $\{x_i, y_i\}_{i=1}^{n}$ and test point x_{test}, the Transformer predicts on x_{test} using $x_{\text{test}}^\top \hat{w}_k^{\text{Newton}}$. Here $\hat{w}_k^{\text{Newton}}$ are the Newton updates given by $\hat{w}_k^{\text{Newton}} = M_k X^\top y$ where M_j is updated as

$$M_j = 2M_{j-1} - M_{j-1} S M_{j-1}, 1 \leq j \leq k, \quad M_0 = \alpha S$$

for some $\alpha > 0$ and $S = X^\top X$. The number of layers of the transformer is $k + 8$ and the dimensionality of the hidden layers is $\mathcal{O}(d)$.

One transformer layer computes one Newton iteration. 3 initial transformer layers are needed for initializing M_0 and 5 layers at the end are needed to read out predictions from the computed pseudo-inverse M_k.

More in the Paper

- Transformers also achieve *second-order* convergence rates on *noisy* linear regression.
- LSTMs cannot improve over layers, and they behave more like Online GD.
- How Transformers deal with more complicated function classes, such as 2-layer Neural Networks, remains a mystery
- Transformers are also similar to other *second-order* algorithms, such as BFGS, but Transformers do better than Conjugate Gradient methods and L-BFGS.

Thanks!