Decoding In-Context Learning: The Role of Preconditioned Gradient Descent in Transformers

Jan Niklas Nertinger

Abstract

We review recent theoretical results and present empirical evidence on linear regression tasks showing that, while Transformers can be configured to mimic preconditioned gradient descent [1, 15], they do not generally implement gradient descent exactly [12]. Our findings suggest that in-context learning exhibits a deep connection to, yet remains distinct from, classical optimization methods.

1 Introduction

Transformers have emerged as a cornerstone in modern machine learning, achieving remarkable success in natural language processing and beyond. A key phenomenon that has attracted significant attention is *in-context learning* - the ability of a model to infer and execute a task solely based on the input prompt, without any parameter updates. In other words, the model "learns" to solve a task by leveraging the relationships present within the prompt itself.

In this work, we study how both linear [1, 15, 3, 9, 11] and nonlinear [5, 6] Transformer architectures can mimic preconditioned gradient descent (PGD) or similar methods on linear regression tasks. Our investigation includes theoretical results showing that linear Transformers can implement PGD and empirical comparisons of nonlinear Transformers (using ReLU, LeakyReLU, and softmax activations) against a PGD baseline.

2 Preliminaries

Data setup We consider linear regression with tokenized data following von Oswald et al. [15], Ahn et al. [1]. Each training example is a pair $(x^{(i)}, y^{(i)})$, where the covariate $x^{(i)} \in \mathbb{R}^d$ is drawn from $\mathcal{N}(0, \Sigma)$ and the target is given by $y^{(i)} = w_*^T x^{(i)}$ with $w_* \sim \mathcal{N}(0, I_d)$. The covariates are organized in the matrix $X = (x^{(1)}, \dots, x^{(n+1)}) \in \mathbb{R}^{d \times (n+1)}$, and we form the task matrix

$$Z = \begin{pmatrix} z^{(1)} & \cdots & z^{(n)} & z^{(n+1)} \end{pmatrix} = \begin{pmatrix} x^{(1)} & \cdots & x^{(n)} & x^{(n+1)} \\ y^{(1)} & \cdots & y^{(n)} & 0 \end{pmatrix} \in \mathbb{R}^{(d+1) \times (n+1)},$$

where the zero indicates the unknown target for the test token $x^{(n+1)}$. Thus, the training data consists of pairs $(Z, w_*^T x^{(n+1)})$, with in-context learning inferring the missing target from the preceding tokens.

Preconditioned gradient descent Preconditioning enhances gradient descent (GD) by incorporating second-order or geometry-aware information, as seen in Natural Gradient Descent [4], AdaGrad [7], or Adam [10]. Instead of the standard GD update $w_{k+1} = w_k + \eta \nabla L(w_k)$, PGD uses a preconditioning matrix A to obtain

$$w_{k+1} = w_k + \eta A \nabla L(w_k),$$

which often leads to faster convergence.

Transformer architecture We use a simplified Transformer based on Ahn et al. [1] that adapts the original attention mechanism of Vaswani et al. [13]. Let $Q, P \in \mathbb{R}^{(d+1)\times(d+1)}$) and $\sigma : \mathbb{R}^{n+1} \to \mathbb{R}^{n+1}$ be a general column-wise activation (here, for f we write f_m for its column-wise application to m columns). To enforce causality - preventing the test token $z^{(n+1)} = (x^{(n+1)}, 0)^T$ from influencing the computation - we introduce a masking matrix M and our attention becomes

$$\mathrm{Attn}^{\sigma}_{P,Q}(Z) = P\,Z\,M \cdot \sigma_{n+1}\big(Z^T Q Z\big), \quad M = \begin{pmatrix} I_n & 0 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}. \tag{1}$$

A 1-headed L-layer σ-Transformer, denoted TF^{σ}_{L}, is parameterized by matrices $\{P_\ell, Q_\ell\}_{\ell=0}^{L-1}$. With $Z_0 := Z$, we iteratively compute

$$Z_{\ell+1} = Z_\ell + \tfrac{1}{n}\,\mathrm{Attn}_{P_\ell,Q_\ell}(Z_\ell) \quad \text{for } \ell = 0, \ldots, L-1,$$

and the final output is given by $\mathrm{TF}^{\sigma}_{L}(Z) := -\,(Z_L)_{(d+1),(n+1)}$, where the minus sign is for technical reasons. In the linear case we omit σ and write TF_L and $\mathrm{Attn}_{P,Q}(Z)$.

3 Gradient descent methods in Transformers

3.1 Linear Transformers are expressive

First, we focus on how Linear Transformers *can* perform GD in-context given a suitable choice of weights. For the linear regression loss $L(w) = \frac{1}{2n}\sum_{i=1}^{n}\big(w^T x^{(i)} - y^{(i)}\big)^2$, GD yields the update $\Delta w = -\eta \nabla_w L(w) = -\frac{\eta}{n}\sum_{i=1}^{n}\big(w^T x^{(i)} - y^{(i)}\big)x^{(i)}$. von Oswald et al. [15] reformulate the loss after the update $\tilde{w} = w + \Delta w$ as

$$L(w + \Delta w) = \frac{1}{2n}\sum_{i=1}^{n}\big((w^T + \Delta w^T)x^{(i)} - y^{(i)}\big)^2 = \frac{1}{2n}\sum_{i=1}^{n}\big(w^T x^{(i)} - \big(y^{(i)} - \Delta w^T x^{(i)}\big)\big)^2,$$

showing that we can view the loss after GD as updating the targets to $\tilde{y}^{(i)} = y^{(i)} - \Delta w^T x^{(i)}$ rather than updating the wights. Using this idea, they show that parameters for a 1-layer linear Transformer can be chosen such that $z^{(i)}_{(1)} = z^{(i)}_{(0)} + \big(0, -\Delta w^T x^{(i)}_{(0)}\big)^T$ for all $i = 1, \ldots, n+1$, yielding

$$\mathrm{TF}_1(Z) = \Delta w^T x^{(n+1)}_{(0)}.$$

Thus, the test token output after one layer equals one GD step starting from $w = 0$ (here, the subscript (ℓ) denotes values from layer Z_ℓ).

Ahn et al. [1] further build on this idea and show that L-layer linear Transformers can implement a wide range of optimization techniques:

Theorem 1. *Let TF_L be an L-layer linear Transformer with parameters*

$$P_0 = \begin{pmatrix} 0_{d\times d} & 0 \\ 0 & 1 \end{pmatrix}, \quad Q_0 = -\begin{pmatrix} A_\ell & 0 \\ 0 & 0 \end{pmatrix}, \text{ where } A_\ell \in \mathbb{R}^{d\times d} \text{ for } \ell = 0, \ldots, L-1.$$

Then $y^{(n+1)}_{(\ell)} = -(w^{gd}_\ell)^T x^{(n+1)}$, where $w^{gd}_0 = 0$ and $w^{gd}_{\ell+1} = w^{gd}_\ell - A_\ell \nabla L(w^{gd}_\ell)$ for $1 \le \ell \le L-1$.

Therefore, $\mathrm{TF}_L(Z) = (w^{\mathrm{gd}}_L)^T x^{(n+1)}$ is the output of L steps of PGD. In other words, Transformers can express many (even adaptive) PGD algorithms. Moreover, Akyürek et al. [3] prove that a linear Transformer can recover the closed-form ridge regression solution $w_* = (X^T X + \lambda I_d)^{-1} X^T y$ via a single Sherman-Morrison update. While these results showcase the high in-context expressivity of linear Transformers, they remain primarily of theoretical interest - constructed weights yielding a specific output may not naturally arise in practice.

3.2 Linear Transformers learn PGD

To determine if a Transformer learns an algorithm in-context, we analyze its optimizer with respect to the in-context loss

$$f(\{P_\ell, Q_\ell\}_{\ell=0}^{L-1}) := \mathbb{E}_{(Z, w_*)}\Big[\Big(\mathrm{TF}^{\sigma}_{L}(Z) - w_*^T x^{(n+1)}\Big)^2\Big]. \tag{2}$$

For a 1-layer linear Transformer with $x^{(i)} \sim \mathcal{N}(0, \Sigma)$ and $w_* \sim \mathcal{N}(0, I_d)$, Ahn et al. [1] show that parameters approximating PGD minimize this loss. In particular, they prove:

Theorem 2. *Let $\Sigma = U\Lambda U^T$ with $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_d)$. Define*

$$P_0 = \begin{pmatrix} 0_{d\times d} & 0 \\ 0 & 1 \end{pmatrix}, \quad Q_0 = -\begin{pmatrix} U\ \mathrm{diag}\Big(\Big\{\Big(\frac{n+1}{n}\lambda_i + \frac{1}{n}\sum_{k=1}^d \lambda_k\Big)^{-1}\Big\}_{i=1}^d\Big)U^T & 0 \\ 0 & 0 \end{pmatrix}.$$

Then these parameters globally minimize $f(\{P, Q\})$ up to scaling (i.e. $P_0 \leftarrow \gamma P_0$ and $Q_0 \leftarrow \gamma^{-1} Q_0$ for some scalar γ).

For large n, the upper-left block of Q_0 approximates Σ^{-1}, corresponding to natural gradient preconditioning in PGD. Thus, 1-layer linear Transformers effectively perform PGD—even incorporating the regularizer $\frac{1}{n}\sum_k \lambda_k$ that becomes significant when n is small relative to d or when Σ is ill-conditioned.

For multilayer ($L \geq 2$) linear Transformers, analysis is more complex. Empirical studies [15] on isotropic data ($\Sigma = I_d$) show that such models often outperform L steps of GD by aligning with a variant called GD^{++}, where inputs are pre-transformed as $x^{(i)} \leftarrow (I_d - \gamma XX^T)x^{(i)}$, with a tuned parameter γ. Moreover, Ahn et al. [1] demonstrate that relaxing sparsity constraints (as in Theorem 1) allows Transformers to learn more sophisticated optimization methods in line with GD^{++}. Finally, Gatmiry et al. [9] show that linear looped Transformers (with weight sharing) minimize the in-context loss via multistep PGD, with gradient flow converging even on the nonconvex landscape.

3.3 Limitations of the GD analogy

Several works [1, 15, 9, 14] argue that Transformers inherently perform PGD or a PGD-like algorithm when trained on structured problems. Ahn et al. [2] further stress that simple linear Transformers, as explored above, offer a useful abstraction for understanding more complex Transformers used with real language data. Shen et al. [12] take a different viewpoint and argue that in-context learning in real world pre-trained Transformers and GD are fundamentally different. They highlight a key limitation based on order sensitivity. In our setup, they propose the following:

Theorem 3. *Let TF_L^σ be an L-layer σ-Transformer and Z a task matrix. Let GD_L denote the output after L steps of GD on the linear regression loss L defined above, and let π_A and π_B be two different permutations of the columns of Z (with the $(n+1)$th column fixed). If in-context learning in TF_L^σ were equivalent to GD, then the predictions must satisfy*

$$\mathrm{TF}_L^\sigma(\pi_A(Z)) - \mathrm{TF}_L^\sigma(\pi_B(Z)) = \mathrm{GD}_L(\pi_A(Z)) - \mathrm{GD}_L(\pi_B(Z)).$$

They argue that, while GD is invariant to the order of training examples, real Transformers are sensitive to this order, implying that in-context learning and GD are fundamentally different. Additionally, they criticize that the parameter matrices used to implement various GD-based algorithms in works such as Akyürek et al. [3], von Oswald et al. [15] are highly contrived and unlikely to emerge naturally. This criticism is countered - at least for our specific data distributions and model architectures - by the results in Ahn et al. [1] (e.g., Theorem 2) or [11], which indicate that such parameters can indeed be learned. Finally, they point out that real-world Transformers, such as LLaMa-7B, do not exhibit GD-like behavior, possibly because natural language data does not organize examples in a way that supports explicit optimization updates.

These contrasting viewpoints raise critical questions about the relevance of GD-like analogies in in-context learning. In the next chapter, we examine the alignment between PGD and the in-context predictions of various nonlinear Transformers on linear regression tasks. First, however, we review key results obtained with nonlinear Transformers.

3.4 Nonlinear activations

While linear Transformers are allow for easier analysis, most deep models employ non-linearities. Bai et al. [5] show that advanced ReLU-based Transformers - often with multiple attention heads and interleaved MLP layers - can approximate a variety of optimization algorithms (e.g., ridge regression, Lasso, GD) in-context, even adaptively selecting among methods based on input.

In the more manageable case of a 1-layer ReLU-Transformer, Ahn et al. [1] derived the optimal parameters in closed form. For consistency in notation, let $\mathrm{ReLU} : \mathbb{R}^{n+1} \to \mathbb{R}^{n+1}$ denote the elementwise ReLU function.

Theorem 4. *Let* $\mathrm{TF}_1^{\mathrm{ReLU}}$ *be a 1-layer ReLU-Transformer with both* $x^{(i)}$ *and* w_* *sampled from* $\mathcal{N}(0, I_d)$*. Then the parameters*

$$P_0 = \begin{pmatrix} 0_{d\times d} & 0 \\ 0 & 1 \end{pmatrix}, \quad Q_0 = \left(\tfrac{1}{2}\cdot\tfrac{n-1}{n} + (d+2)\cdot\tfrac{1}{n}\right)^{-1}\begin{pmatrix} I_d & 0 \\ 0 & 0 \end{pmatrix}$$

globally optimize the in-context loss $f(\{P, Q\})$ *defined in* (2)*.*

This parameter choice closely resembles that in Theorem 2 - with diagonal entries $\frac{n-1}{n} + (d+2)\cdot\frac{1}{n}$ in the isotropic case - yet it does not imply that a ReLU-Transformer behaves identically to its linear counterpart (Theorem 1 assumes linearity). Nonetheless, this similarity further motivates our empirical investigation into whether PGD-like behavior emerges in nonlinear Transformers.

4 Experiments

Our experiments address two main questions: (i) Does the correspondence between in-context learning and PGD extend to non-linear activations, as suggested by the structural similarity between the optimizers in Theorems 4 and 2? (ii) Does this similarity persist in more realistic settings than the highly constructed cases critiqued by Shen et al. [12]? Below, we outline our experimental approach.

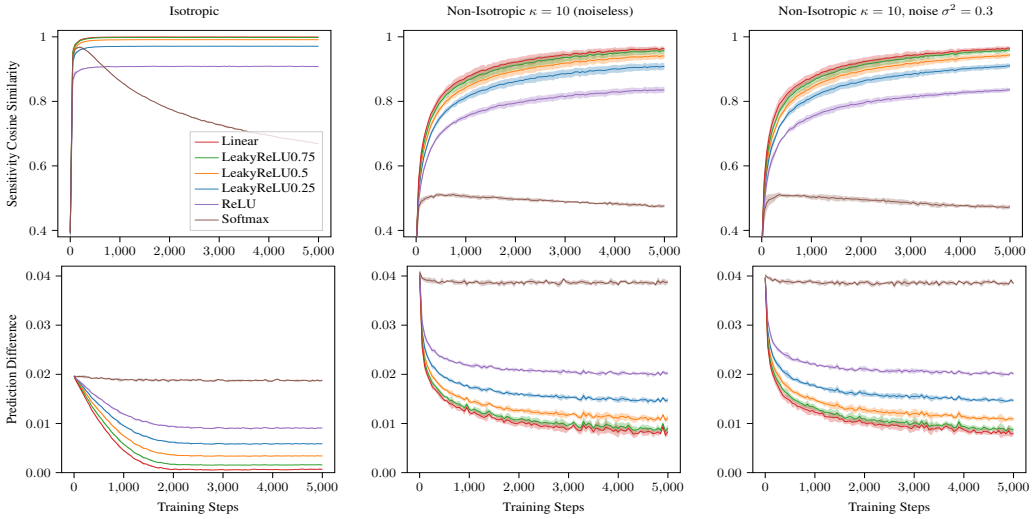

Figure 1: **Comparison with PGD** top: cosine similarity of sensitivites; bottom: prediction difference.

4.1 Experimental setup

We train 1-layer σ-Transformers (TF_1^σ) on the in-context loss (2) using $n = 20$ tokens and input dimension $d = 5$. These models are compared to a PGD baseline that performs one step: $\mathrm{PGD}_1^\eta(Z) := \Sigma^{-1}\frac{\eta}{n}\sum_{i=1}^n y^{(i)}(x^{(i)})^T x^{(n+1)}$, derived from the linear regression loss (with $w_0 = 0$). Data is generated with $w_* \sim \mathcal{N}(0, I_d)$ and $x^{(i)} \sim \mathcal{N}(0, \Sigma)$ for various condition numbers κ, where $\Sigma = U\,\mathrm{diag}(1, \kappa^{\frac{1}{d-1}}, \kappa^{\frac{2}{d-1}}, \ldots, \kappa)U^T$, and U is a uniformly random orthogonal matrix. Optionally, Gaussian noise with variance ν^2 is added to the labels $y^{(i)} = w_*^T x^{(i)}$.

We determine an optimal step-size η_* for PGD via line search over 10^4 validation tasks. Inspired by von Oswald et al. [15], we assess similarity between the Transformer and PGD by measuring (i) the average absolute prediction difference $|\mathrm{TF}_1^\sigma(Z) - \mathrm{PGD}_1^{\eta_*}(Z)|$ and (ii) the average cosine similarity between their sensitivities $\frac{\partial}{\partial x^{(n+1)}}\mathrm{TF}_1^\sigma(Z)$ and $\frac{\partial}{\partial x^{(n+1)}}\mathrm{PGD}_1^{\eta_*}(Z)$ indicating whether the learned model aligns in direction (or structure) independently of the (implicit) step-size. We also track the in-context loss for both methods, repeating each experiment 5 times per parameter combination and averaging the results. As suggested by Theorem 4 as well as Wortsman et al. [16], we consider ReLU activation, along with several variants of $\mathrm{LeakyReLU}_\alpha$ (defined as $\mathrm{LeakyReLU}_\alpha(x) = \max(x, \alpha x)$ for $\alpha \in (0, 1)$) and the commonly used softmax activation.

Table 1: Best achieved in-context loss across various data distributions

Conditioning	$\kappa = 1$			$\kappa = 10$			$\kappa = 100$		
Noise ν^2	0.0	0.1	0.3	0.0	0.1	0.3	0.0	0.1	0.3
PGD	**1.13**	**1.15**	**1.17**	4.85	4.89	4.92	33.02	32.83	32.93
Linear	1.13	1.15	1.18	**4.52**	**4.57**	**4.62**	**26.94**	**27.00**	**26.76**
LeakyReLU0.75	1.15	1.17	1.20	4.60	4.66	4.69	27.06	27.11	27.37
LeakyReLU0.5	1.24	1.26	1.29	4.97	5.01	5.03	28.82	29.01	29.17
LeakyReLU0.25	1.46	1.48	1.52	5.85	5.90	5.93	34.10	34.24	34.24
ReLU	1.93	1.95	1.99	7.69	7.74	7.79	45.20	44.91	45.32
Softmax	4.57	4.54	4.54	19.13	19.31	19.26	125.45	127.22	126.82

4.2 Discussion of results

We now examine the empirical findings in Figure 1 and Table 1 to address our research questions.

Transformers with (Leaky)ReLU activations exhibit a strong correspondence with PGD, achieving high cosine similarity and low prediction differences across various data setups (see Figure 1). In particular, the LeakyReLU variants closely mimic the behavior of linear Transformers and align well with PGD. As expected, there is a noticeable drop in similarity when moving from LeakyReLU0.25 to strict ReLU, likely due to the abrupt removal of negative activations. Despite this, ReLU-Transformers still show surprisingly strong alignment - even though the computation in the attention layer (1) does not explicitly suggest such behavior. Additionally, for all (Leaky)ReLU-based Transformers, cosine similarity saturates rapidly in the isotropic case, while both prediction differences and cosine similarity take longer to stabilize in non-isotropic setups. This indicates that the models quickly capture the gradient direction but require more iterations to fine-tune step-sizes and preconditioning. Notably, as shown in Table 1, the linear and even the LeakyReLU0.75 configuration outperform PGD in ill-conditioned setups, despite PGD's learning rate η being optimized for each validation task.

Although a resemblance to PGD is still evident in ill-conditioned data (Figure 1), the alignment is significantly reduced even with this rather mild ill-conditioning, supporting the concerns raised by Shen et al. [12]. Moreover, while similarity decreases in ill-conditioned setups, some Transformer-based models outperform PGD (see Table 1), suggesting that in-context learning in Transformers involves more sophisticated computations as discussed in Fu et al. [8] claiming that Transformers can even achieve second-order convergence in-context. Importantly, the addition of Gaussian noise has minimal impact on both the alignment with PGD and overall performance, demonstrating the robustness of these models.

Although not our primary focus, the softmax Transformer shows intriguing behavior. In the isotropic case, it displays a concentrated spike in cosine similarity that quickly diminishes, while the prediction difference remains largely unchanged. One possible explanation is that softmax prioritizes specific tokens, so that parameter changes do not translate into large prediction differences even when the partial derivatives vary significantly. The initial strong alignment might result from evenly distributed parameters at initialization, which lead to a more uniformly weighted sum similar to PGD.

5 Conclusion

We reviewed theoretical insights and conducted empirical experiments to investigate the connection between in-context learning in Transformers and PGD. Our findings demonstrate that while Transformers - especially in their linear and certain nonlinear forms - can be configured to approximate PGD-like behavior, they often don't implement GD exactly. In fact, recent work [12] argues that in-context learning is not inherently equivalent to GD.

In our view, recent research [1, 15, 9, 3] has not sought to claim a strict equivalence between in-context learning and GD but rather to reveal a deep underlying connection between the implicit optimization processes in Transformers and classical iterative algorithms. Our results support this perspective, highlighting that while the GD analogy is useful for understanding aspects of Transformer behavior, in-context learning remains a distinct phenomenon with its own complexities.

References

[1] K. Ahn, X. Cheng, H. Daneshmand, and S. Sra. Transformers learn to implement preconditioned gradient descent for in-context learning. In *Advances in Neural Information Processing Systems*, 2023.

[2] K. Ahn, X. Cheng, M. Song, C. Yun, A. Jadbabaie, and S. Sra. Linear attention is (maybe) all you need (to understand transformer optimization). In *International Conference on Learning Representations*, 2024.

[3] E. Akyürek, D. Schuurmans, J. Andreas, T. Ma, and D. Zhou. What learning algorithm is in-context learning? investigations with linear models. In *International Conference on Learning Representations*, 2023.

[4] S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2), 1998.

[5] Y. Bai, F. Chen, H. Wang, C. Xiong, and S. Mei. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. In *Advances in Neural Information Processing Systems*, 2023.

[6] D. Dai, Y. Sun, L. Dong, Y. Hao, S. Ma, Z. Sui, and F. Wei. Why can gpt learn in-context? language models implicitly perform gradient descent as meta-optimizers. In *ICLR Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2023.

[7] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[8] D. Fu, T.-q. Chen, R. Jia, and V. Sharan. Transformers learn to achieve second-order convergence rates for in-context linear regression. In *Advances in Neural Information Processing Systems*, 2024.

[9] K. Gatmiry, N. Saunshi, S. Reddi, S. Jegelka, and S. Kumar. Can looped transformers learn to implement multi-step gradient descent for in-context learning? In *Proceedings of Machine Learning Research*, volume 235, pages 15130–15152, 2024.

[10] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[11] A. V. Mahankali, T. Hashimoto, and T. Ma. One step of gradient descent is provably the optimal in-context learner with one layer of linear self-attention. In *International Conference on Learning Representations*, 2024.

[12] L. Shen, A. Mishra, and D. Khashabi. Do pretrained transformers learn in-context by gradient descent? *arXiv preprint arXiv:2310.08540*, 2024.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[14] M. Vladymyrov, J. von Oswald, M. Sandler, and R. Ge. Linear transformers are versatile in-context learners. In *ICML 2024 Workshop on In-Context Learning*, 2024.

[15] J. von Oswald, E. Niklasson, E. Randazzo, J. Sacramento, A. Mordvintsev, A. Zhmoginov, and M. Vladymyrov. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, 2023.

[16] M. Wortsman, J. Lee, J. Gilmer, and S. Kornblith. Replacing softmax with relu in vision transformers. *arXiv preprint arXiv:2309.08586*, 2023.