# Transformers May Learn to Classify In-Context by Context-Adaptive Kernel Gradient Descent

**Sara Dragutinović** [1] **Andrew M. Saxe** [*2] **Aaditya K. Singh** [*2]

## Abstract

The remarkable ability of transformers to learn new concepts solely by reading examples within the input prompt, termed in-context learning (ICL), is a crucial aspect of intelligent behavior. Here, we focus on understanding the learning algorithm transformers use to learn from context. Existing theoretical work, often based on simplifying assumptions, has primarily focused on linear self-attention and continuous regression tasks, finding transformers can learn in-context by gradient descent. Given that transformers are typically trained on discrete and complex tasks, we bridge the gap from this existing work to the setting of *classification*, with *non-linear* (importantly, *softmax*) activation. We find that transformers still learn to do gradient descent in-context, though on functionals in the kernel feature space and with a context-adaptive learning rate in the case of softmax transformer.

## 1. Introduction

In-context-learning (ICL) is a critical ability often exhibited by transformers whereby they can adapt to context (i.e. input prompt) and solve tasks not present during training. After its initial emergence when training transformers at scale on natural language (Brown et al., 2020), much empirical and theoretical work has gone into studying this phenomena. Following prior work (Garg et al., 2022; Akyürek et al., 2022; Von Oswald et al., 2023; Ahn et al., 2023), this paper aims to contribute to answering the question: 'What learning algorithms do transformers use to perform ICL?'

Prior theoretical work suggests that transformers may implement in-context learning by taking a step of (preconditioned) gradient descent (GD) (Von Oswald et al., 2023; Garg et al., 2022; Akyürek et al., 2022; Ahn et al., 2023; Mahankali et al., 2023). However, much of this work has been restricted to linear regression tasks and transformers with a linear activation function, both of which are departures from standard settings (Vaswani et al., 2017). Here, we emphasize our key departures from prior work, and how they are bridging the gaps between theoretical results and the transformers used in practice:

1. **Switching to classification task**: Learning a continuous task with mean squared error (MSE) loss and learning a discrete task with cross-entropy (CE) loss are fundamentally different. Minimizing MSE corresponds to maximizing likelihood *under a Gaussian noise assumption*, whereas minimizing CE loss requires only the assumption of i.i.d. samples, without any additional noise model. From a practical standpoint, many transformers are trained to predict a probability distribution over a discrete set of tokens—most notably, large language models (LLMs) predicting the next token in a sequence—making the classification setup more representative of real-world applications. We show that transformers are expressive enough to implement a step of GD, even in *classification setup*, with *CE loss*.
2. **Using softmax attention**: From the original introduction of transformers (Vaswani et al., 2017) to today's LLMs and vision transformers, the softmax activation in self-attention has remained the standard choice. On the other hand, the mathematical convenience of linear attention (i.e., attention without any activation function) has attracted much of the existing theoretical analysis. We show that, while linear self-attention still implements the usual step of GD on context data, softmax self-attention implements a step of functional gradient descent in the Radial Basis Function (RBF) kernel feature space (kernel GD); furthermore, it benefits from having a context-adaptive learning rate.

[*]Equal contribution [1]University of Oxford, United Kingdom [2]Gatsby Computational Neuroscience Unit, UCL, United Kingdom. Correspondence to: Sara Dragutinović <sara.dragutinovic@cs.ox.ac.uk>.

*Accepted at Methods and Opportunities at Small Scale (MOSS), ICML 2025*, Vancouver, Canada.

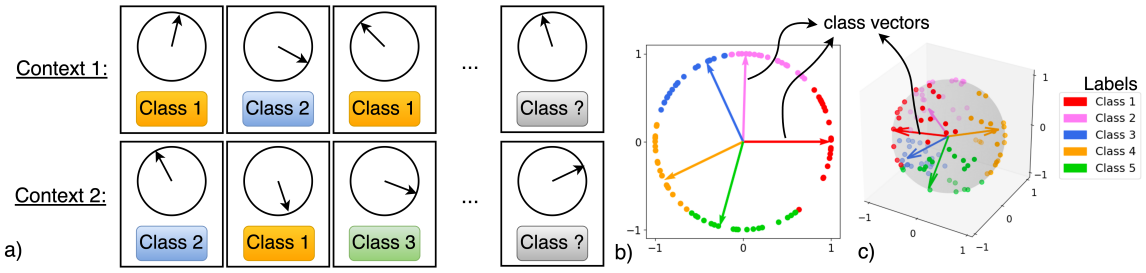


*Figure 1.* a) Two example contexts transformer gets from our classification task (see Section 2.1). For Context 1, the correct class for query is Class 1, as $\mathbf{x}_{\text{query}}$ lays between the first and the third context vector. With Context 2, we emphasize how: 1) context vectors $\mathbf{x}_i$ and query input $\mathbf{x}_{\text{query}}$ differ between contexts; 2) class assignment differs between contexts—if the first context vector was in Context 1, its label would be Class 1, not Class 2. *Right:* Two full contexts, with $C = 5, n = 100$ and b) $d = 2$, c) $d = 3$. The arrows are representing class vectors, and the points context vectors; different colors correspond to different class labels.

## 2. Setup

### 2.1. Classification Task

We follow the usual, controlled setting in theoretical ICL work: given a context of $n$ input-label pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ and a query input $\mathbf{x}_{\text{query}}$, the model is expected to predict its label $y_{\text{query}}$. In our classification task, we first sample $C$ prototype class vectors $\mathbf{z}_1, \ldots, \mathbf{z}_C$ uniformly from the unit sphere $S^{d-1}$. These class vectors define the labeling rule but are hidden from the model. Each context vector $\mathbf{x}_i \in S^{d-1}$ is assigned the label of its nearest class vector, and we sample $\frac{n}{C}$ such vectors per class using rejection sampling. The query is sampled from the same distribution, by first choosing a class $y_{\text{query}}$ uniformly and then sampling $\mathbf{x}_{\text{query}}$ uniformly from its region.[1] The illustrations of the task are shown in Figure 1.

### 2.2. Model Setup

Following the setup of (Von Oswald et al., 2023), we use a simplified one layer transformer with a single head, and concatenated input-output pairs as tokens: $[\mathbf{x}_i, \mathbf{y}_i] \in \mathbb{R}^{d+C}$, where $\mathbf{y}_i$ is one-hot encoding of $y_i$. The query token is concatenated with $\mathbf{0} \in \mathbb{R}^C$ vector as its $y$-entry. These tokens form rows of the input matrix $X$ (Figure 2). Let $W_Q, W_K, W_V$ and $W_O$ are the $(d+C) \times (d+C)$ query, key, value and projection/output weight matrices. We theoretically analyze three different models, differing in activation function on the attention: linear, kernel, and softmax self-attention.

**Linear self-attention.** Given $\mathbf{x}_{\text{query}}$, linear self-attention makes the following prediction for $\mathbf{y}_{\text{query}}$:

$$\hat{\mathbf{y}}_{\text{query}} = \text{softmax}\left\{ \left[ \left( [\mathbf{x}_{\text{query}}, \mathbf{0}] W_Q^\top W_K X^\top \right) X W_V^\top W_O^\top \right]_y \right\}.^{2}$$

Subscript $y$ refers to taking the $y$-entry of a token, as seen is Figure 2.

**Kernel self-attention.** In the case of kernel self-attention, the activation function on the attention originates from a kernel $k : \mathbb{R}^{d+C} \times \mathbb{R}^{d+C} \to \mathbb{R}$ (symmetric and positive semi-definite). Specifically, those activation functions $f = \text{act}_k$ satisfy the following: for two matrices $A = [\mathbf{a}_1, \ldots \mathbf{a}_a], B = [\mathbf{b}_1, \ldots, \mathbf{b}_b]$, applying kernel activation gives a $a \times b$-sized matrix $\text{act}_k([\mathbf{a}_1, \ldots, \mathbf{a}_a], [\mathbf{b}_1, \ldots, \mathbf{b}_b]) = \{k(\mathbf{a}_i, \mathbf{b}_j)\}_{i=1,\ldots,a; j=1,\ldots,b}$. With this, the prediction for $\mathbf{y}_{\text{query}}$ is

$$\hat{\mathbf{y}}_{\text{query}} = \text{softmax}\left\{ \left[ \text{act}_k \left( W_Q [\mathbf{x}_{\text{query}}, \mathbf{0}]^\top, W_K X^\top \right) X W_V^\top W_O^\top \right]_y \right\}.$$

**Softmax self-attention.** The most commonly used activation function is softmax, giving the prediction

$$\hat{\mathbf{y}}_{\text{query}} = \text{softmax}\left\{ \left[ \text{softmax}\left( \frac{[\mathbf{x}_{\text{query}}, \mathbf{0}] W_Q^\top W_K X^\top}{\sqrt{d+C}} \right) X W_V^\top W_O^\top \right]_y \right\}.^{3}$$

[1] We make this choice to avoid models learning pathological strategies such as picking the most common label from context.

[2] Note that the softmax in the formula has nothing to do with activation on the attention; it is used as we're training on a classification task, and matches what is commonly done in practice with cross-entropy loss.

[3] Softmax self-attention is *not* a form of kernel SA, due to the normalization inherent to the softmax function.

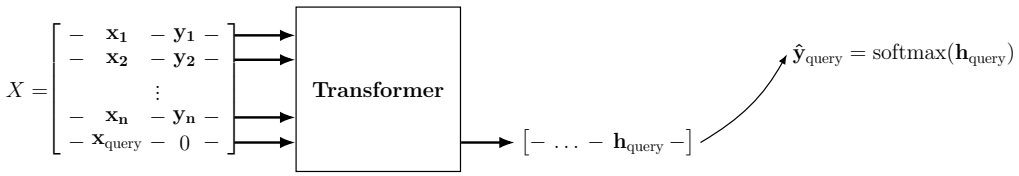


*Figure 2.* Illustration of a transformer forward pass. We only make use of the update of the last token to get the final prediction $\hat{\mathbf{y}}_{\text{query}}$.

## 3. Transformers *can* implement kernel gradient descent on classification tasks

To build up to our main theoretical result, namely that *softmax* transformers can implement *adaptive learning rate kernel gradient descent*, we iteratively relax assumptions made in prior work (Von Oswald et al., 2023), which showed that linear transformers can implement GD on linear regression tasks.

### 3.1. Linear transformer on linear classification task

First, we bridge the gap from regression tasks with MSE loss to classification tasks with CE loss; see the proof in Appendix B.

**Linear classification task:** For consistency, here we introduce what we call *linear classification*, and is also referred to by names softmax regression or multinomial logistic regression. We are given data $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, where $\mathbf{y}_i \in \mathbb{R}^C$ are one-hot encoded labels. For a new data point $\mathbf{x}_{\text{query}} \in \mathbb{R}^d$, we give prediction $\mathbb{P}[\mathbf{x}_{\text{query}} \text{ is in class } j] = (\text{softmax}(W^\top \mathbf{x}_{\text{query}}))_j$ (for $j \in \{1, ..., C\}$), where $W \in \mathbb{R}^{d \times C}$ are the parameters we optimize for, using the data.

**Proposition 3.1.** *Linear self-attention is expressive enough to implement one step of gradient descent on cross-entropy loss in the linear classification setup, using the samples from context and assuming we start from* $W_0 = \mathbf{0}$.[4]

### 3.2. Any kernel activation transformer on linear classification task

Next, we consider a kernel activation transformer, to bridge the gap between linear and softmax SA. We build upon Cheng et al. (2023) to show that kernel SA can implement one step of GD in the kernel feature space on this *classification* task; for the full proof, see Appendix C.

**Proposition 3.2.** *Kernel activation self-attention is expressive enough to implement one step of gradient descent in the kernel feature space, on cross-entropy loss in the linear classification setup, assuming we start from* $W_0 = \mathbf{0}$.[5]

### 3.3. Softmax transformer on linear classification task

Finally, we show that softmax self-attention implements a step of kernel gradient descent, using the RBF kernel, but with a *context-adaptive learning rate*. We assume that all input vectors have norm one,[6] that is $\|\mathbf{x}_i\| = 1$. In the case of RBF kernel $k(\mathbf{x}, \mathbf{x}') = e^{-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}}$ with kernel width $\sigma$, the equation for one step of kernel GD with learning rate $\eta$ (Equation 3) gives $\hat{\mathbf{y}}_{\text{query}} = \text{softmax}\left\{ \frac{\eta}{n e^{\frac{1}{\sigma^2}}} \sum_{i=1}^n \mathbf{y}_i e^{\frac{\mathbf{x}_i^\top \mathbf{x}_{\text{query}}}{\sigma^2}} \right\}$. If we let $\sigma^2 = \frac{\sqrt{d+C}}{c_\sigma}$ and define an adaptive learning rate (as a function of $X$ - sample points): $\eta(X) = \frac{c_\eta e^{1/\sigma^2} n}{\sum_{i=1}^n e^{\mathbf{x}_i^\top \mathbf{x}_{\text{query}}/\sigma^2}}$, for some constants $c_\sigma, c_\eta > 0$, we get

$$\hat{\mathbf{y}}_{\text{query}} = \text{softmax}\left\{ c_\eta \sum_{i=1}^n \mathbf{y}_i \frac{e^{\frac{c_\sigma \mathbf{x}_i^\top \mathbf{x}_{\text{query}}}{\sqrt{d+C}}}}{\sum_{i=1}^n e^{\frac{c_\sigma \mathbf{x}_i^\top \mathbf{x}_{\text{query}}}{\sqrt{d+C}}}} \right\}. \tag{1}$$

Equation 1 can be implemented by softmax self-attention; for details, see Appendix D. Our main departure here from standard kernel gradient descent is variable step size $\eta(X)$, which we term *context-adaptive learning rate*. An important property is that $\eta(X)$ depends on the points in the dataset/context, and also on the position of $\mathbf{x}_{\text{query}}$ relative to them. If there are many points near $\mathbf{x}_{\text{query}}$, $\eta(X)$ will be relatively *smaller*, evoking some of the flavor of non-parametric algorithms.

[4] Note that the realistic assumption of starting from $W_0 = \mathbf{0}$ corresponds to no prior knowledge on the classes.

[5] For infinite dimensional kernel feature spaces, what we mean by $W_0 = \mathbf{0}$ is that we're starting a step of functional GD from the zero functional (see Appendix C.1).

[6] Such an assumption is reasonable given the use of LayerNorm (Ba et al., 2016) in most transformers.

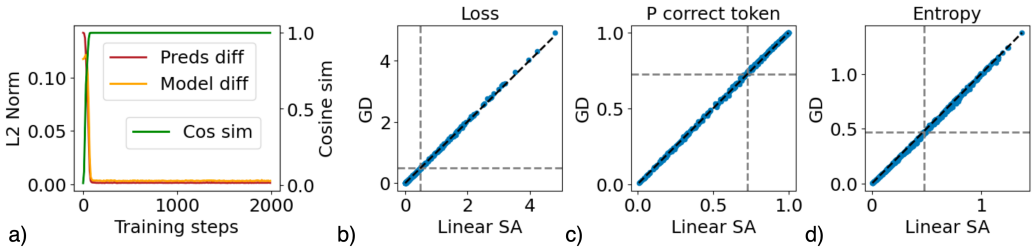


*Figure 3.* Similarity between a trained linear SA and a step of GD, in the setup $C = 5, n = 100, d = 5$. a) Models alignment metrics through transformer training. *Right:* Metrics similarity per context sample: b) loss, c) probability of the correct class and d) entropy of $\hat{\mathbf{y}}_{\text{query}}$: each point represents the value on one context. Dotted line corresponds to the mean value of a metric.

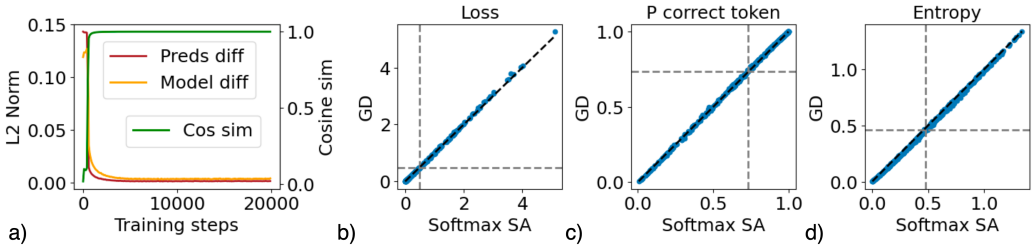


*Figure 4.* Similarity between a trained softmax SA and a context-adaptive step of kernel GD, in the setup $C = 5, n = 100, d = 5$. a) Models alignment metrics through transformer training. *Right:* Metrics similarity per context sample: b) loss, c) probability of the correct class and d) entropy achieved by both algorithms.

## 4. Transformers *do* learn kernel gradient descent to solve classification tasks

Knowing theoretically that self-attention is expressive enough to do an (adaptive) step of (kernel) gradient descent in the classification setup, the key question becomes if this solution is present in trained transformers, i.e. does a trained transformer actually implement GD on the data in context? In this section, we present empirical evidence towards a positive answer by training transformers using the setup from Section 2. To show the similarity between the two algorithms—trained SA and a vanilla step of GD—we make use of two different approaches. All implementation details can be found in Appendix G.

**Metrics similarity per context sample.** Firstly, we observe per-sequence metrics in both models. Concretely, metrics we observe are loss, entropy, and probability of the correct class. For each metric, we plot different contexts as points: $x$-axis being the metric value (of predicting $\mathbf{y}_{\text{query}}$) obtained from a trained transformer, and the $y$-axis is metric value obtained with a step of gradient descent. The indicator two algorithms are similar is all the points concentrating around the $y = x$ line.

**Models alignment.** Following the alignment metrics introduced by Von Oswald et al. (2023), we investigate the similarity of the two models by capturing three quantities throughout training, for each averaging the values over $N = 100$ contexts:

- Prediction differences (Preds diff): the norm of the differences in the predictions by the two models $\|\hat{\mathbf{y}}^{\text{TR}}_{\text{query}} - \hat{\mathbf{y}}^{\text{GD}}_{\text{query}}\|$
- Cosine similarity (Cos sim): the cosine similarity between the sensitivities of the two models $\frac{\partial(\hat{\mathbf{y}}^{\text{TR}}_{\text{query}})_j}{\partial \mathbf{x}_{\text{query}}}$ and $\frac{\partial(\hat{\mathbf{y}}^{\text{GD}}_{\text{query}})_j}{\partial \mathbf{x}_{\text{query}}}$, averaged over classes $j = 1, ..., C$
- Sensitivity differences (Model diff): the difference in sensitivities of the two models $\frac{1}{C}\sum_{j=1}^{C} \| \frac{\partial(\hat{\mathbf{y}}^{\text{TR}}_{\text{query}})_j}{\partial \mathbf{x}_{\text{query}}} - \frac{\partial(\hat{\mathbf{y}}^{\text{GD}}_{\text{query}})_j}{\partial \mathbf{x}_{\text{query}}} \|$

The results show high similarity between a trained linear SA and a GD step with $\eta$ found through linear search, as illustrated by Figure 3 in the setting $n = 100, C = 5, d = 5$. For more results on linear attention, see Appendix E). Furthermore, there is a significant similarity between a trained softmax SA and a context adaptive kernel GD step (Equation 1), where $c_\eta, c_\sigma$ are found in a grid-search. The setting $n = 100, C = 5, d = 5$ results in Figure 4, and the other settings can be found in Appendix F. However, there was another algorithm softmax transformer learns (less common but present), that we term 'elimination'. For further details about it, refer to Appendix J.

## 5. Conclusion

The remarkable ICL ability of transformers is a key reason behind their success in generalizing to never-before-seen sequences, both in controlled small-scale settings like ours and at a much larger scales of models such as LLMs. Approaching ICL from a theoretical perspective, prior theoretical work (Garg et al., 2022; Von Oswald et al., 2023) demonstrated that self-attention can implement a step of gradient descent, but under simplifying assumptions—specifically, using linear attention and MSE loss on a regression task. We bridge these gaps to more practical transformer settings by investigating softmax self-attention on a classification task with CE loss. we demonstrate that softmax self-attention is expressive enough to implement a single context-adaptive step of kernel gradient descent. Furthermore, we design a synthetic classification task and use it to show that softmax self-attention learns to implement this theoretical solution in practice.

Altogether, we hope this paper lays theoretical foundations for exploring ICL in settings that are closer to practical applications. While important challenges remain—such as understanding multi-layer architectures and the role of MLPs—we believe progress will come by systematically relaxing simplifying assumptions, one step at a time. Foundational understanding is not only important for theoretical clarity, but also essential for driving meaningful, lasting progress.

## Acknowledgements

We thank Basile Confavreaux, Jin Hwa Lee, Yedi Zhang, Stephanie C. Y. Chan and Andrew K. Lampinen for useful discussions. This work was supported by a Sir Henry Dale Fellowship from the Wellcome Trust and Royal Society (216386/Z/19/Z) to A.M.S, and the Sainsbury Wellcome Centre Core Grant from Wellcome (219627/Z/19/Z) and the Gatsby Charitable Foundation (GAT3755) to S.D, A.K.S and A.M.S.

## References

Ahn, K., Cheng, X., Daneshmand, H., and Sra, S. Transformers learn to implement preconditioned gradient descent for in-context learning. *Advances in Neural Information Processing Systems*, 36:45614–45650, 2023.

Akyürek, E., Schuurmans, D., Andreas, J., Ma, T., and Zhou, D. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*, 2022.

Anand, S., Lepori, M. A., Merullo, J., and Pavlick, E. Dual process learning: Controlling use of in-context vs. in-weights strategies with weight forgetting. *arXiv preprint arXiv:2406.00053*, 2024.

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and De Freitas, N. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Bai, Y., Chen, F., Wang, H., Xiong, C., and Mei, S. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. *Advances in neural information processing systems*, 36:57125–57211, 2023.

Bhattamishra, S., Patel, A., Blunsom, P., and Kanade, V. Understanding in-context learning in transformers and llms by learning to learn discrete functions. *arXiv preprint arXiv:2310.03016*, 2023.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Carroll, L., Hoogland, J., Farrugia-Roberts, M., and Murfet, D. Dynamics of transient structure in in-context linear regression transformers. *arXiv preprint arXiv:2501.17745*, 2025.

Chan, B., Chen, X., György, A., and Schuurmans, D. Toward understanding in-context vs. in-weight learning. *arXiv preprint arXiv:2410.23042*, 2024.

Chan, S., Santoro, A., Lampinen, A., Wang, J., Singh, A., Richemond, P., McClelland, J., and Hill, F. Data distributional properties drive emergent in-context learning in transformers. *Advances in neural information processing systems*, 35: 18878–18891, 2022.

Chen, S., Sheen, H., Wang, T., and Yang, Z. Training dynamics of multi-head softmax attention for in-context learning: Emergence, convergence, and optimality. *arXiv preprint arXiv:2402.19442*, 2024.

Cheng, X., Chen, Y., and Sra, S. Transformers implement functional gradient descent to learn non-linear functions in context. *arXiv preprint arXiv:2312.06528*, 2023.

Collins, L., Parulekar, A., Mokhtari, A., Sanghavi, S., and Shakkottai, S. In-context learning with transformers: Softmax attention adapts to function lipschitzness. *Advances in Neural Information Processing Systems*, 37:92638–92696, 2024.

Edelman, E., Tsilivis, N., Edelman, B., Malach, E., and Goel, S. The evolution of statistical induction heads: In-context learning markov chains. *Advances in Neural Information Processing Systems*, 37:64273–64311, 2024.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

Garg, S., Tsipras, D., Liang, P. S., and Valiant, G. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598, 2022.

He, T., Doshi, D., Das, A., and Gromov, A. Learning to grok: Emergence of in-context learning and skill composition in modular arithmetic tasks. *Advances in Neural Information Processing Systems*, 37:13244–13273, 2024.

Huang, Y., Cheng, Y., and Liang, Y. In-context convergence of transformers. *arXiv preprint arXiv:2310.05249*, 2023.

Hubinger, E., van Merwijk, C., Mikulik, V., Skalse, J., and Garrabrant, S. Risks from learned optimization in advanced machine learning systems. *arXiv preprint arXiv:1906.01820*, 2019.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Li, H., Wang, M., Lu, S., Cui, X., and Chen, P.-Y. How do nonlinear transformers learn and generalize in in-context learning? *arXiv preprint arXiv:2402.15607*, 2024a.

Li, T., Zhang, G., Do, Q. D., Yue, X., and Chen, W. Long-context llms struggle with long in-context learning. *arXiv preprint arXiv:2404.02060*, 2024b.

Li, Z., Cao, Y., Gao, C., He, Y., Liu, H., Klusowski, J., Fan, J., and Wang, M. One-layer transformer provably learns one-nearest neighbor in context. *Advances in Neural Information Processing Systems*, 37:82166–82204, 2024c.

Liu, J., Shen, D., Zhang, Y., Dolan, B., Carin, L., and Chen, W. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*, 2021.

Lu, Y. M., Letey, M. I., Zavatone-Veth, J. A., Maiti, A., and Pehlevan, C. Asymptotic theory of in-context learning by linear attention. *arXiv preprint arXiv:2405.11751*, 2024.

Mahankali, A., Hashimoto, T. B., and Ma, T. One step of gradient descent is provably the optimal in-context learner with one layer of linear self-attention. *arXiv preprint arXiv:2307.03576*, 2023.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Park, C. F., Lubana, E. S., Pres, I., and Tanaka, H. Competition dynamics shape algorithmic phases of in-context learning. *arXiv preprint arXiv:2412.01003*, 2024.

Reddy, G. The mechanistic basis of data dependence and abrupt learning in an in-context classification task. *arXiv preprint arXiv:2312.03002*, 2023.

Singh, A., Chan, S., Moskovitz, T., Grant, E., Saxe, A., and Hill, F. The transient nature of emergent in-context learning in transformers. *Advances in Neural Information Processing Systems*, 36:27801–27819, 2023.

Singh, A. K., Moskovitz, T., Hill, F., Chan, S. C., and Saxe, A. M. What needs to go right for an induction head? a mechanistic study of in-context learning circuits and their formation. *arXiv preprint arXiv:2404.07129*, 2024.

Singh, A. K., Moskovitz, T., Dragutinovic, S., Hill, F., Chan, S. C., and Saxe, A. M. Strategy coopetition explains the emergence and transience of in-context learning. *arXiv preprint arXiv:2503.05631*, 2025.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Vladymyrov, M., Von Oswald, J., Sandler, M., and Ge, R. Linear transformers are versatile in-context learners. *Advances in Neural Information Processing Systems*, 37:48784–48809, 2024.

Von Oswald, J., Niklasson, E., Randazzo, E., Sacramento, J., Mordvintsev, A., Zhmoginov, A., and Vladymyrov, M. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pp. 35151–35174. PMLR, 2023.

Wang, A. T., Henao, R., and Carin, L. Transformer in-context learning for categorical data. *arXiv preprint arXiv:2405.17248*, 2024a.

Wang, Z., Jiang, B., and Li, S. In-context learning on function classes unveiled for transformers. In *Forty-first International Conference on Machine Learning*, 2024b.

Wei, J., Wei, J., Tay, Y., Tran, D., Webson, A., Lu, Y., Chen, X., Liu, H., Huang, D., Zhou, D., et al. Larger language models do in-context learning differently. *arXiv preprint arXiv:2303.03846*, 2023.

Wu, J., Zou, D., Chen, Z., Braverman, V., Gu, Q., and Bartlett, P. L. How many pretraining tasks are needed for in-context learning of linear regression? *arXiv preprint arXiv:2310.08391*, 2023.

Yang, T., Huang, Y., Liang, Y., and Chi, Y. In-context learning with representations: Contextual generalization of trained transformers. *arXiv preprint arXiv:2408.10147*, 2024.

Yin, K. and Steinhardt, J. Which attention heads matter for in-context learning? *arXiv preprint arXiv:2502.14010*, 2025.

Zhang, R., Frei, S., and Bartlett, P. L. Trained transformers learn linear models in-context. *Journal of Machine Learning Research*, 25(49):1–55, 2024a.

Zhang, R., Wu, J., and Bartlett, P. In-context learning of a linear transformer block: benefits of the mlp component and one-step gd initialization. *Advances in Neural Information Processing Systems*, 37:18310–18361, 2024b.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Zhang, Y., Singh, A. K., Latham, P. E., and Saxe, A. Training dynamics of in-context learning in linear attention. *arXiv preprint arXiv:2501.16265*, 2025.

Zhao, S., Nguyen, T., and Grover, A. Probing the decision boundaries of in-context learning in large language models. *Advances in Neural Information Processing Systems*, 37:130408–130432, 2024.

## A. Extended Related Work

The concept of emergent in-context learning was first observed in (Brown et al., 2020), where it was noted that the trained GPT-3 language model could perform well on few-shot tasks presented in the prompt, without any parameter updates. Since then, ICL has been compared to meta-learning (Andrychowicz et al., 2016; Finn et al., 2017) and so-called mesa-learning (Hubinger et al., 2019), where trained models are interpreted as performing optimization at inference time.

Staying close to practice, there has been extensive empirical analysis of ICL, with several works investigating how well LLMs perform in in-context learning settings, as well as their capabilities and limitations (Liu et al., 2021; Zhang et al., 2022; Wei et al., 2023; Bhattamishra et al., 2023; Li et al., 2024b; Zhao et al., 2024). To further investigate the phenomenon,

researchers have begun designing carefully crafted experimental setups, usually involving smaller transformers and non-language datasets, such as Omniglot (Chan et al., 2022; Singh et al., 2024), boolean function classes (Bhattamishra et al., 2023), modular arithmetic (He et al., 2024) or Markov chains (Edelman et al., 2024; Park et al., 2024). An important component of transformers' ability to perform ICL has been attributed to the formation of so-called induction heads (Olsson et al., 2022; Reddy, 2023; Singh et al., 2024; Edelman et al., 2024). Additionally, ICL has been shown to be a transient phenomenon (Singh et al., 2023; Anand et al., 2024; Chan et al., 2024; Yin & Steinhardt, 2025; Carroll et al., 2025; Singh et al., 2025), where, if a task can be solved through both in-context and in-weights learning, the model eventually transitions to solving it via in-weights learning—that is, by memorizing the example-label pairs in its parameters. In this case, data properties also play a significant role in the emergence and persistence of ICL (Chan et al., 2022).

From a more theoretical perspective, prior work has typically focused on small transformers (1–2 layers), often combined with additional simplifications for mathematical convenience. The prior work referenced in this paragraph adopts the controlled ICL setting, where the context consists of input-output pairs drawn from a task—each context corresponding to a different task.

**Linear self-attention.** Another commonly used simplification of self-attention is the variant without the softmax activation, often referred to as linear self-attention. Linear self-attention is expressive enough to solve linear regression task by performing one step of GD, and it also does learn it when trained on ICL linear regression (each context corresponds to different linear regression task), as shown by prior work (Von Oswald et al., 2023; Garg et al., 2022; Akyürek et al., 2022; Ahn et al., 2023; Mahankali et al., 2023). The mathematical simplicity of linear attention has facilitated further theoretical research on ICL, revealing additional properties and learning dynamics (Wu et al., 2023; Zhang et al., 2024a; Lu et al., 2024; Vladymyrov et al., 2024; Zhang et al., 2024b; 2025).

**Self-attention with non-linear activation.** Due to the analytical complexity, there haven't been as many theoretical results on attention mechanisms with non-linearities. Ahn et al. (2023); Bai et al. (2023) have provided expressivity results (weight constructions) for ReLU-activated self-attention, demonstrating its ability to implement various algorithms for solving various tasks. Softmax attention has been explored in combination with continuous tasks, using MSE loss (Cheng et al., 2023; Huang et al., 2023; Collins et al., 2024; Chen et al., 2024; Yang et al., 2024; Wang et al., 2024b). Collins et al. (2024) shows that softmax self-attention adapts to function Lipschitzness; however, this result isn't applicable in our setting, as classification tasks aren't continuous (i.e. have infinite Lipschitzness). Cheng et al. (2023) mention the similarity between softmax transformer and a step of gradient descent in the kernel space; some preliminary results left indicate that there are cases of our classification setup where the two algorithms perform differently. Moving closer to the matter in this paper, there have been studies on in-context binary *classification* using softmax self-attention, but together with hinge loss (Li et al., 2024a), and MSE loss (Li et al., 2024c). Our goal is to take a step further to understanding softmax self-attention in the ICL classification setting with *cross-entropy* loss, as this is the setup used with transformers in practice (e.g., in LLMs).

Importantly, we emphasize how our work differs from the concurrent study of Wang et al. (2024a). While they also aim to bridge toward the ICL classification setting, they introduce a distinct synthetic task and construct a specific transformer block—featuring two softmax attention layers—that performs functional gradient descent on a parameterized prediction model. In contrast, our setting is simpler and arguably more natural. We provide a rigorous proof, grounded in RKHS theory, that transformers can implement functional GD. Moreover, we offer stronger, per-context empirical evidence that trained transformers actually learn this behavior. Finally, we introduce the notion of a context-adaptive learning rate, whose impact we leave to future work.

## B. Proof of Proposition 3.1

In this section we provide the proof of Proposition 3.1, which demonstrates that a linear transformer can perform one step of gradient descent in a linear classification setting.

*Proof outline.* Cross-entropy loss on one sample $(\mathbf{x}, \mathbf{y})$ is given by formula

$$L(W) = -\mathbf{y}^\top \log \text{softmax}(\mathbf{z}), \text{ where } \mathbf{z} = W^\top \mathbf{x}.$$

To calculate $\nabla_W L(\mathbf{x})$, let $W = [\mathbf{w}_1, ..., \mathbf{w}_C]$ and $z_j = \mathbf{w}_j^\top \mathbf{x}$. Then $\nabla_\mathbf{z} L = \text{sfmx}(\mathbf{z}) - \mathbf{y}$. As $\nabla_W z_1 = [\mathbf{x}, \mathbf{0}, \dots, \mathbf{0}]$, we have

$$\nabla_W L(\mathbf{x}) = \sum_{i=1}^{C} \frac{\partial L}{\partial z_i} \nabla_W z_i = \sum_{i=1}^{C} (p_i - y_i)[\mathbf{0}, ..., \mathbf{0}, \mathbf{x}, \mathbf{0}, ...], \text{ where } \mathbf{p} = \text{softmax}(\mathbf{z}).$$

Recognizing the outer product, this leads us to

$$\nabla_W L(\mathbf{x}) = \mathbf{x}(\text{softmax}(\mathbf{z}) - \mathbf{y})^\top.$$

This means one step of gradient descent on the whole dataset gives:

$$W_{\text{new}} = W_0 - \frac{\eta}{n}\sum_{i=1}^{n} \mathbf{x}_i(\text{softmax}(W_0^\top \mathbf{x}_i) - \mathbf{y}_i)^\top,$$

and assuming $W_0 = \mathbf{0}$, this makes a new prediction on $\mathbf{x}_{\text{query}}$:

$$\begin{aligned}\hat{\mathbf{y}}_{\text{query}} &= \text{softmax}\{W_{\text{new}}^\top \mathbf{x}_{\text{query}}\} \\ &= \text{softmax}\left\{ -\frac{\eta}{n}\sum_{i=1}^{n}\left(\frac{1}{C}\mathbf{1} - \mathbf{y}_i\right)\mathbf{x}_i^\top \mathbf{x}_{\text{query}}\right\} \\ &= \text{softmax}\left\{\frac{\eta}{n}\sum_{i=1}^{n}\mathbf{y}_i\mathbf{x}_i^\top \mathbf{x}_{\text{query}}\right\}, \end{aligned} \tag{2}$$

where crucially we use the fact that softmax is shift invariant. Intuitively, Equation 2 can be implemented with linear self-attention because $\mathbf{x}_i^\top \mathbf{x}_{\text{query}}$ can be obtained in the product of key and query matrices, and multiplier $\mathbf{y}_i$ can be extracted using the value matrix. We provide a simple set of weights that achieve this. Namely, setting

$$W_Q^\top W_K = \begin{bmatrix} I_d & 0_{d\times C} \\ 0_{C\times d} & 0_C \end{bmatrix}, W_V^\top W_O^\top = \begin{bmatrix} 0_d & 0_{d\times C} \\ 0_{C\times d} & \frac{\eta}{N} I_C \end{bmatrix}$$

enables extracting $\mathbf{x}_i$s from the merged tokens in the attention, and $\mathbf{y}_i$s in the value and projection matrix. □

## C. Proof of Proposition 3.2

Here we provide the proof of Proposition 3.2, showing that kernel self-attention is expressive enough to implement a step of gradient descent in kernel feature space. We first provide a sketch of the proof for kernels with finite dimensional feature spaces, and then separately showing that Equation 3 also holds for general kernels.

*Proof sketch for finite dimensional feature spaces.* We provide intuition here for finite dimensional kernel feature spaces, with a fully general proof (extending to functionals in RKHS) in Section C.1. Let $k$ be a kernel with finite dimensional kernel space and $\varphi$ be its feature map, i.e. $k(\mathbf{x}, \mathbf{x}') = \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}')\rangle$. Using kernel feature expansion, on input $\mathbf{x}$, we predict $\hat{\mathbf{y}} = \text{softmax}(W^\top \varphi(\mathbf{x}))$. All the equations from Section 3.1 hold through, with $\mathbf{x}$ replaced by $\varphi(\mathbf{x})$. Hence we end up with

$$\hat{\mathbf{y}}_{\text{query}} = \text{softmax}\left\{\frac{\eta}{n}\sum_{i=1}^{n}\mathbf{y}_i\varphi(\mathbf{x}_i)^\top\varphi(\mathbf{x}_{\text{query}})\right\} = \text{softmax}\left\{\frac{\eta}{n}\sum_{i=1}^{n}\mathbf{y}_i k(\mathbf{x}_i, \mathbf{x}_{\text{query}})\right\}. \tag{3}$$

Equation 3 holds for any kernel, even if the kernel feature space is infinite-dimensional, see the full proof in Appendix C.1. Similarly to the linear transformer case, Equation 3 can be implemented with a kernel activation SA, as $k(\mathbf{x}_i, \mathbf{x}_{\text{query}})$ can be obtained from the attention with activation $\text{act}_k$, and $\mathbf{y}_i$ can be extracted using the value matrix. A simple set of weights implementing it is given by:

$$W_Q = \begin{bmatrix} I_d & 0_{d\times C} \\ 0_{C\times d} & 0_C \end{bmatrix} W_K = \begin{bmatrix} I_d & 0_{d\times C} \\ 0_{C\times d} & 0_C \end{bmatrix}$$
$$W_V = \begin{bmatrix} 0_d & 0_{d\times C} \\ 0_{C\times d} & I_C \end{bmatrix} W_O = \begin{bmatrix} 0_d & 0_{d\times C} \\ 0_{C\times d} & \frac{\eta}{N} I_C \end{bmatrix},$$

which recover the Equation 3 in the transformer's forward pass. □

### C.1. Functional gradient descent leads to Equation 3

Let $X = \mathbb{R}^d$ be the space of inputs, and $k : X \times X \to \mathbb{R}$ a kernel. Let $\mathcal{H}$ be the reproducing kernel Hilbert space (RKHS) associated with $k$. We do a functional gradient descent in $\mathcal{H}$, making use of chain rule. Corresponding to the $W_0 = \mathbf{0}$

assumption of Proposition 3.2, we start from $f_1, ..., f_C \in \mathcal{H}$ being the zero functions, where $\text{softmax}(f_j(\mathbf{x}))_{j=1}^C$ gives the predicted probabilities of $\mathbf{x}$ being of classes $j = 1, ..., C$. Now, we lay out similar equations as in the case of normal gradient descent:

$$\begin{aligned}
L(f_1, ..., f_C) &= -\mathbf{y}^\top \log(\text{softmax}(\mathbf{z})) && [\text{where } \mathbf{z} = (f_1(\mathbf{x}), ..., f_C(\mathbf{x}))^\top] \\
\nabla_{f_1} L &= \sum_{i=1}^{C} \frac{\partial L}{\partial z_i} \nabla_{f_1} z_i && [\text{by chain rule}] \\
&= \frac{\partial L}{\partial z_1} \nabla_{f_1} z_1 \\
&= \frac{\partial L}{\partial z_1} \nabla_{f_1} E_{\mathbf{x}}(f_1) && [\text{where } E_{\mathbf{x}} : \mathcal{H} \to \mathbb{R} \text{ is the evaluation functional}^7] \\
&= \frac{\partial L}{\partial z_1} k(\mathbf{x}, \cdot) && [\text{as } \nabla E_{\mathbf{x}} = k(\mathbf{x}, \cdot)] \\
&= (p_1 - y_1) k(\mathbf{x}, \cdot) && [\text{where } \mathbf{p} = \text{softmax}(\mathbf{z}), \text{ using } \nabla_{\mathbf{z}}(L) = \text{softmax}(\mathbf{z}) - \mathbf{y}]
\end{aligned}$$

where the functional gradient with respect to other functionals $f_j, j = 2, ..., C$ is computed similarly. Given a sample set of points $(\mathbf{x}_i, \mathbf{y}_i)_{i=1}^n$, the updated $f_1, ..., f_C$ become

$$\begin{bmatrix} f_{1,\text{new}} \\ f_{2,\text{new}} \\ ... \\ f_{C,\text{new}} \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ ... \\ f_C \end{bmatrix} - \frac{\eta}{n} \sum_{i=1}^{n} (\text{softmax}(\mathbf{z}_i) - \mathbf{y}_i) k(\mathbf{x}_i, \cdot)$$

Now assuming $f_i = 0$, and applying these updated functions to $\mathbf{x}_{\text{query}}$ gives:

$$\begin{aligned}
\hat{\mathbf{y}}_{\text{query}} &= \text{softmax} \begin{bmatrix} f_{1,\text{new}}(\mathbf{x}_{\text{query}}) \\ f_{2,\text{new}}(\mathbf{x}_{\text{query}}) \\ ... \\ f_{C,\text{new}}(\mathbf{x}_{\text{query}}) \end{bmatrix} \\
&= \text{softmax}\left\{ -\frac{\eta}{n} \sum_{i=1}^{n} (\frac{1}{C}\mathbf{1} - \mathbf{y}_i) k(\mathbf{x}_i, \mathbf{x}_{\text{query}}) \right\} \\
&= \text{softmax}\left\{ \frac{\eta}{n} \sum_{i=1}^{n} \mathbf{y}_i k(\mathbf{x}_i, \mathbf{x}_{\text{query}}) \right\}.
\end{aligned}$$

This concludes the proof that Equation 3 holds for all kernels. □

## D. Softmax self-attention can do one context-adaptive step of kernel GD on classification task

First, we give a more detailed derivation of Equation 1, from Equation 3 with RBF kernel:

$$\begin{aligned}
\hat{\mathbf{y}}_{\text{query}} &= \text{softmax}\left\{ \frac{\eta}{n} \sum_{i=1}^{n} \mathbf{y}_i e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_{\text{query}}\|^2}{2\sigma^2}} \right\} \\
&= \text{softmax}\left\{ \frac{\eta}{n} \sum_{i=1}^{n} \mathbf{y}_i e^{\frac{2\mathbf{x}_i^\top \mathbf{x}_{\text{query}} - 2}{2\sigma^2}} \right\} \\
&= \text{softmax}\left\{ \frac{\eta}{n e^{\frac{1}{\sigma^2}}} \sum_{i=1}^{n} \mathbf{y}_i e^{\frac{\mathbf{x}_i^\top \mathbf{x}_{\text{query}}}{\sigma^2}} \right\}.
\end{aligned}$$

Now setting $\sigma$ and introducing $\eta(X)$ as stated in the main paper leads to Equation 1.

[6] I.e. $E_{\mathbf{x}}(f) = f(\mathbf{x})$, for all $f \in \mathcal{H}$.

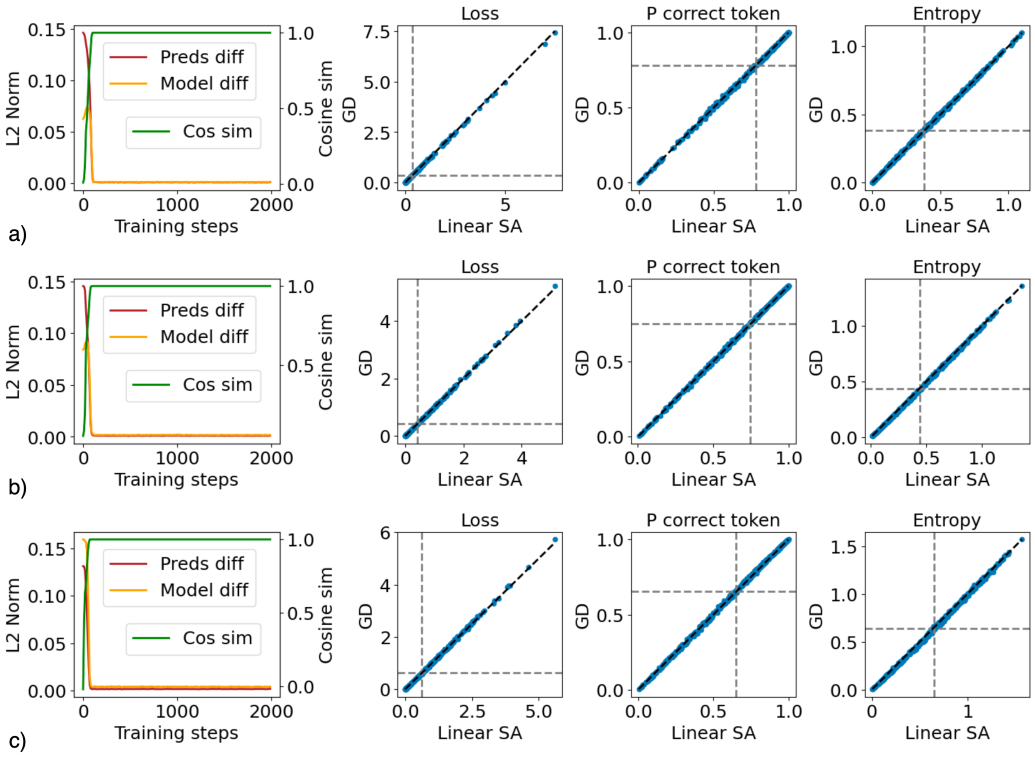


*Figure 5.* Similarity between the two algorithms—trained linear SA and a GD step—in the setup with $C = 5, n = 100$ and a) $d = 2$, b) $d = 3$, c) $d = 10$. *Left:* Alignment metrics through transformer training. *Right:* Similarity per context sample of loss, probability of the correct class and entropy of $\hat{\mathbf{y}}_{\text{query}}$, respectively.

Here is a simple set of self-attention weights giving raise to the Equation 1 in the transformer's forward pass, hence showing that softmax transformer is expressive enough to implement one context-adaptive step of GD:

$$W_Q^\top W_K = \begin{bmatrix} c_\sigma I_d & 0_{d\times C} \\ 0_{C\times d} & 0_C \end{bmatrix}, W_V^\top W_O^\top = \begin{bmatrix} 0_d & 0_{d\times C} \\ 0_{C\times d} & c_\eta I_C \end{bmatrix}.$$

## E. Linear transformer does learn to do GD

To test whether trained linear SA actually learns the solution provided in Proposition 3.1, we compare its performance with one step gradient descent (starting from $W = \mathbf{0}$), whose learning rate value is found through a linear search. In Figure 5, we show the results for $C = 5, n = 100$ and varying $d$. The low values of the prediction and sensitivity differences, the cosine similarity converging to 1, and the similar per-sample metrics suggest that a single-layer linear transformer does learn to solve linear classification task by doing one step of GD.

## F. Softmax transformer does learn to do context-adaptive step of kernel GD

Here, we provide more results analogous to Figure 4, varying different dimensions of the space $d$. For the vanilla context-adaptive step of kernel GD (Equation 1), we pick $c_\eta$ and $c_\sigma$ that minimize CE loss in a grid-search. We compare this to a softmax SA trained on our toy classification task contexts. Figure 6 shows the results for $d = 2, 3, 10$. We note that for $d = 2$, the gradient $\frac{\partial(\hat{\mathbf{y}}^{\text{TR}}_{\text{query}})_j}{\partial \mathbf{x}_{\text{query}}}$ is numerically unstable to compute in our setup, so we couldn't provide the results on the cosine similarity and sensitivity differences. The numerical instability comes from highly saturated softmax attention. In the case of $d = 10$, softmax attention and context-adaptive kernel GD are slightly different, and we believe that is due to the fact that the training of the transformer, in terms of the values of $c_\eta$ and $c_\sigma$, hasn't converged yet (see Appendix I).

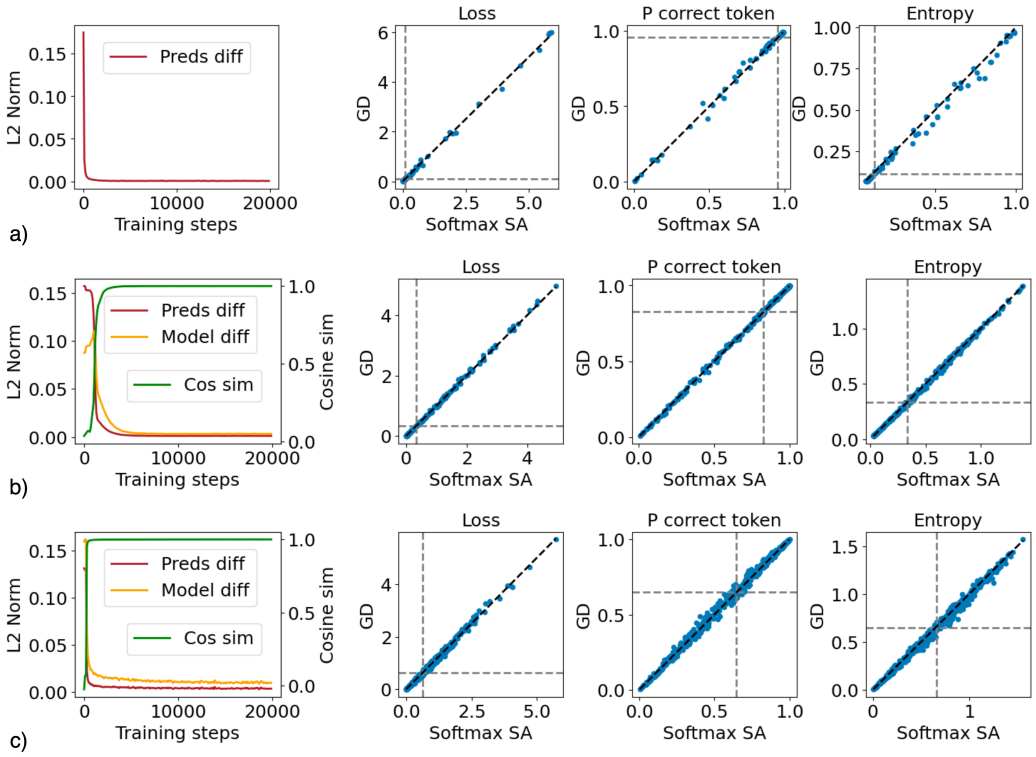


*Figure 6.* Similarity between a trained softmax self-attention and a context-adaptive step of kernel GD—in the setup with $C = 5, n = 100$ and: a) $d = 2$, b) $d = 3$, c) $d = 10$.

## G. Implementation details

**Training.** For our empirical experiments, we train transformers in JAX (Bradbury et al., 2018) using a modified version of the open-sourced code from (Singh et al., 2024). The use of JAX ensures full reproducibility of our experiments, while also enabling efficient training via just-in-time compilation and automatic vectorization. We use Adam optimizer (Kingma & Ba, 2014) with the default parameters. In all the experiments, we use single-headed self attention, without the causal mask. In all experiments, model training had 2048 batch size, meaning 2048 linear classification task contexts. All evaluation sets contained 512 context samples. We emphasize again that we don't use positional encoding, token embedding nor unembedding, MLPs nor layer norm[8].

**Linear transformer hyper-parameters.** For linear transformer, we mostly follow the setup in (Von Oswald et al., 2023), with learning rate $0.00005$. On initialization, we rescale the weights by $0.002$ and perform gradient clipping with value $0.001$. We train the transformer for 200,000 iterations (batches). We evaluate every 100 iterations, hence our results are showing training steps in the range 0-2,000.

**Softmax transformer hyper-parameters.** For softmax transformer, we also rescaled the initial weights by 0.002, and we set the gradient clipping value to $1.0$. Transformer training was done over 2,000,000 iterations and (as linear) evaluated every 100 iterations, therefore shown in range 0-20,000. Learning rate was dimension dependent: $d = 2$ - $0.0006$; $d = 3$ - $0.00003$; $d = 5$ - $0.0001$; $d = 10$ - $0.0005$.

**Gradient descent grid search.** For a GD step, we did a grid search for learning rate over $\eta \in [10^0, 10^{2.5}]$, trying out 100 examples chosen from a logarithmic range. For a kernel GD step, we did search for 2 hyper-parameter values, $\eta$ and $\sigma^2$. We tested 100 values of $\eta \in [10^0, 10^3]$ (log scale), times 100 values of $\sigma^2 \in [10^{-3}, 10^2]$ (log scale). In both cases, the search was done using $N = 10,000$ different contexts from the classification task described in 2.1.

**Grid search for $c_\sigma$ and $c_\eta$.** For context-adaptive kernel GD, we did grid search to find $c_\sigma$ and $c_\eta$. These are shown in Figures

[8] As we have only one layer, it doesn't really make a difference.

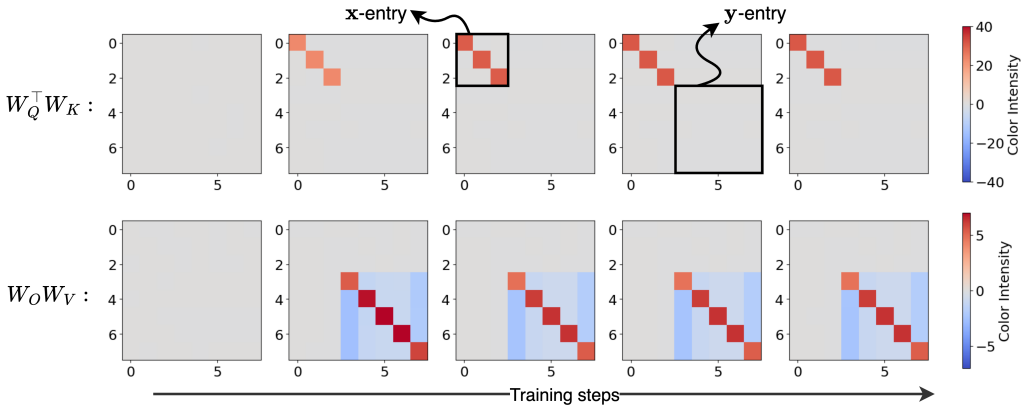


*Figure 7.* The weights through training: $W_Q^\top W_k$ (top row) and $W_O W_V$ (bottom row) for the setup $n = 100, C = 5, d = 3$. Observing they look similar to the construction in Section D, we found a way to extract the effective values $c_\eta, c_\sigma$ that softmax self-attention uses, described in Section H.

4 and 6. We used different scan ranges depending on the dimension. For $c_\sigma$, we tried 100 values from log range [vmin, vmax], where vmin and vmax are: $d = 2$ - vmin $= 10^0$, vmax $= 10^{3.5}$; $d = 3, 5$ - vmin $= 10^{-1}$, vmax $= 10^2$; $d = 10$ - vmin $= 10^{-3}$, vmax $= 10^1$. For $c_\eta$, the 100 values were from log range with: $d = 2, 3, 5$ - vmin $= 10^{-1}$, vmax $= 10^2$; $d = 10$ - vmin $= 10^1$, vmax $= 10^4$. To perform the search, we used $N = 10,000$ different contexts from the classification task in Section 2.1.

## H. Extracting $c_\sigma$ and $c_\eta$ from a trained transformer; Dynamics

As demonstrated empirically, a transformer learns to implement Equation 1, corresponding to a single step of GD in kernel space with a context-adaptive learning rate. This suggests that a trained softmax SA—with parameters $W_Q, W_K, W_V, W_O$—can be effectively characterized by just two parameters: $c_\eta$ and $c_\sigma$. Observing that the learned weights $W_Q^\top W_K$ and $W_V^\top W_O^\top$ resemble those from our construction in Appendix D, we found a method to extract the effective values of $c_\sigma$ and $c_\eta$ from the trained model. For example, the weight matrices $W_Q^\top W_K$ and $W_O W_V$ through training can be seen in Figure 7 for the case $n = 100, C = 5, d = 3$. To extract the effective $c_\sigma$, we take the average value at the diagonal of $\mathbf{x}$-entry submatrix of $W_Q^\top W_K$. Extracting $c_\eta$ goes as follows: we first note that adding a constant to a column in $\mathbf{y}$-entry submatrix of $W_O W_V$ mathematically doesn't change anything—due to softmax activation being shift invariant. Thus to get to the form of our construction in Appendix D, we add to each column of $\mathbf{y}$-entry the average value of the off-diagonal entries in that column. This results in the off-diagonal entries being approximately 0, so we take the average of the diagonal entries to be the effective value $c_\eta$. Our experiments show that this method indeed extracts the correct effective values—in Figure 8 we can see that in the setting $n = 100, C = 5$ and $d = 3$, transformer's similarity metrics align well with the implementation of the Equation 1 with the effective values of $c_\eta, c_\sigma$ we extracted. For reference, we also included an example with $c_\eta, c_\sigma$ slightly off from the extracted values—we see that differences are large, strengthening correctness of our extraction method.

Figure 9 shows the how the two effective values evolve through training, for fixed $n = 100, C = 5$, and varying $d$. General dynamics involves the initial spike—the time that weights of the transformer align to do kernel GD, followed by scaling of $c_\eta$ and $c_\sigma$. For $d = 2$ softmax transformer learns to pay attention only to points very close to the query point (high $c_\sigma$). However, this isn't the case in general, as we can see for higher dimensional spaces. As the dimension gets higher, less and less points are in a close neighborhood of $\mathbf{x}_{\text{query}}$, with many more being approximately orthogonal to it. This leads to larger attention window (made possible with lower $c_\sigma$), and higher saturation/confidence in the samples that are captured in that window (higher $c_\eta$). In the case of $d = 10$, we see that the values haven't converged yet—see Appendix I for more experiments and discussions on this setup.

## I. Finding $c_\eta$ and $c_\sigma$ in the case $d = 10$

After plotting the effective value evolution for the case $d = 10$ (and $C = 5, n = 100$), we've noticed that the growth of $c_\eta$ hasn't finished yet. Trying out different (model training) learning rates, we plot the dynamics over 60,000 evaluation

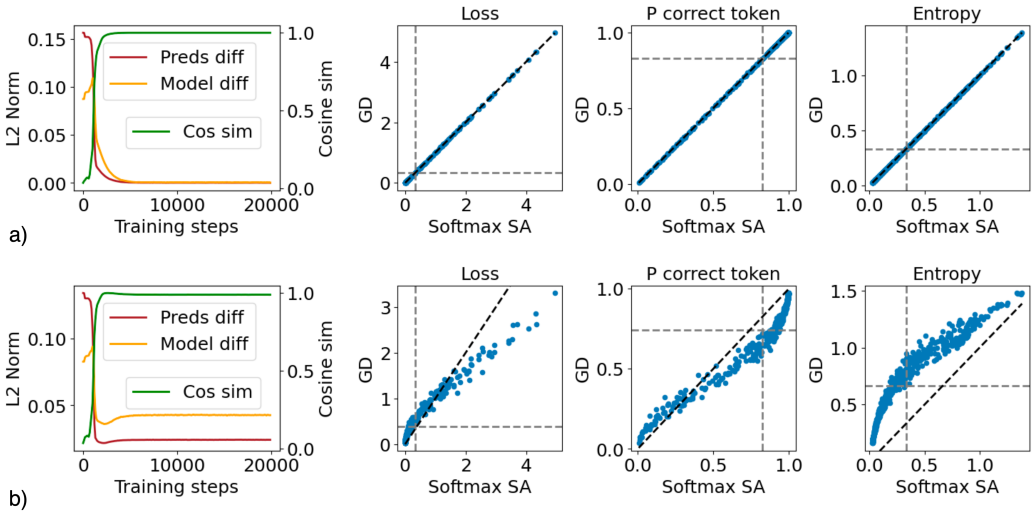


*Figure 8.* a) The differences between a trained self-attention and Equation 1 with $c_\eta, c_\sigma$ values extracted with our method from the transformer. The similarities observed indicate that our method works. b) The differences between a trained self-attention and Equation 1 with $c_\sigma, c_\eta$ being slightly off ($c_\eta = 5, c_\sigma = 25$) from the values extracted ($c_\eta = 7.15, c_\sigma = 31.07$). The observable differences further amplify the effectiveness of our method.

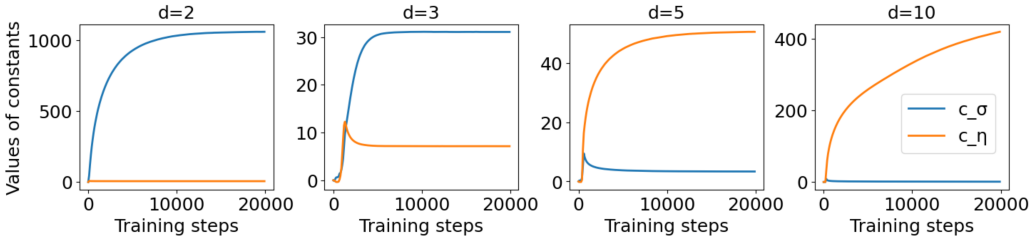


*Figure 9.* Dynamical evolution of $c_\sigma$ and $c_\eta$ through softmax self-attention training process. The figure depicts the setup $C = 5, n = 100$ and $d$ being 2,3,5 and 10, from left to right.

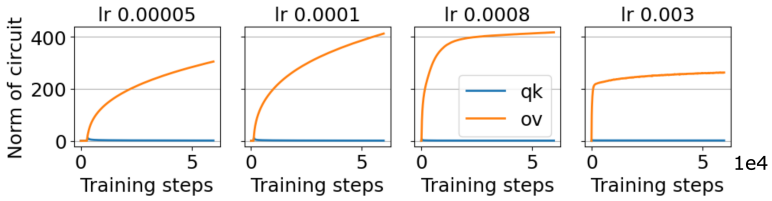


*Figure 10.* Evolution of the effective values of the constants $c_\eta$ and $c_\sigma$ through training, for different values of the (transformer's training) learning rate: $0.00005, 0.0001, 0.0008$ and $0.003$, respectively.

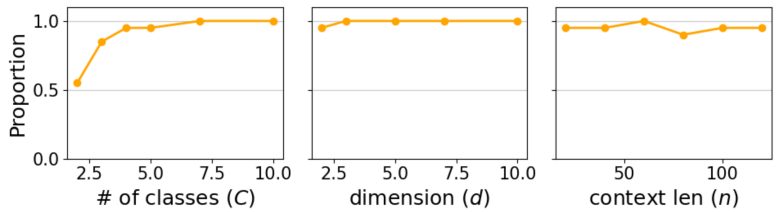


*Figure 11.* Proportion of runs where selection algorithm is learned, over 20 random seeds, varying number of classes, dimension and context length, from left to right.

iterations, seen in Figure 10. Interestingly, for higher learning rate, the growth of $c_\eta$ flattens earlier and around a smaller value, as seen from the right two plots. Observation that no value of $c_\eta$ converged yet, together with the fact that grid search finds the optimal $c_\eta$ to be ∼8,000, we realized that this case of $d = 10$ has a more complicated loss landscape. This evidence suggests that a learning rate scheduler may be needed to fully successfully train a softmax SA in this case. Since this is not the primary focus of the paper, as well as our compute power restriction, we leave further investigations for future work. A lesson learned from these experiments is that seemingly flat loss and accuracy curves can be misleading, and the model may still be developing.

## J. Softmax attention selecting vs eliminating

Softmax transformer learns (at least) two different algorithms through training. In the more common case, our results match the context-adaptive kernel GD solution (Figures 4, 6). This strategy can be viewed as predicting query class logits as a weighted average of $\mathbf{y}_i$s, weighted based on how close $\mathbf{x}_{\text{query}}$ is to different $\mathbf{x}_i$s. Intuitively, the model is *selecting* the context points from which to copy the labels, similar to the mechanism of induction heads (Olsson et al., 2022; Reddy, 2023; Singh et al., 2024), with the radius of selection controlled by the kernel width $\sigma^2 > 0$. More rarely, we observe a different algorithm—*elimination*—whereby softmax self-attention learns to attend to the tokens *furthest* from the query and then *subtract* those out.[9] Throughout this paper, we focused on the selection algorithm, because it is the one learned in majority of cases. However, especially when $C$ is low, the elimination algorithm is appearing as well. In Figure 11, we provide the proportion of times the selection algorithm is learned, across different seeds (20 different combinations in total). The base setting is $d = 2, C = 5, n = 100$. For each of the plots, we make number of classes, dimension and context length, respectively, vary and observe how does the percentage of selection algorithm leaned change. These experiments affirm that selection algorithm is the one leaned in majority of runs, especially in our default settings. We leave further studies of the elimination algorithm and the conditions it appears in for future work.

[9]Such a mechanism was more frequent when $C$ was small, indicating possible connections to the anti-induction heads noted by Singh et al. (2024).