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]

Abstract

At present, the mechanisms of in-context learning in Transformers are not well understood and remain mostly an intuition. In this paper, we suggest that training Transformers on auto-regressive objectives is closely related to gradient-based meta-learning formulations. We start by providing a simple weight construction that shows the equivalence of data transformations induced by 1) a single linear self-attention layer and by 2) gradient-descent (GD) on a regression loss. Motivated by that construction, we show empirically that when training self-attention-only Transformers on simple regression tasks either the models learned by GD and Transformers show great similarity or, remarkably, the weights found by optimization match the construction. Thus we show how trained Transformers become mesa-optimizers i.e. learn models by gradient descent in their forward pass. This allows us, at least in the domain of regression problems, to mechanistically understand the inner workings of in-context learning in optimized Transformers. Building on this insight, we furthermore identify how Transformers surpass the performance of plain gradient descent by learning an iterative curvature correction and learn linear models on deep data representations to solve non-linear regression tasks. Finally, we discuss intriguing parallels to a mechanism identified to be crucial for in-context learning termed *induction-head* (Olsson et al., 2022) and show how it could be understood as a specific case of in-context learning by gradient descent learning within Transformers.

[1]Department of Computer Science, ETH Zürich, Zürich, Switzerland [2]Google Research. Correspondence to: Johannes von Oswald <voswaldj@ethz.ch>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

1. Introduction

In recent years Transformers (TFs; Vaswani et al., 2017) have demonstrated their superiority in numerous benchmarks and various fields of modern machine learning, and have emerged as the *de-facto* neural network architecture used for modern AI (Dosovitskiy et al., 2021; Yun et al., 2019; Carion et al., 2020; Gulati et al., 2020). It has been hypothesised that their success is due in part to a phenomenon called *in-context learning* (Brown et al., 2020; Liu et al., 2021): an ability to flexibly adjust their prediction based on additional data given *in context* (i.e. in the input sequence itself). In-context learning offers a seemingly different approach to few-shot and meta-learning (Brown et al., 2020), but as of today the exact mechanisms of how it works are not fully understood. It is thus of great interest to understand what makes Transformers pay attention to their context, what the mechanisms are, and under which circumstances, they come into play (Chan et al., 2022b; Olsson et al., 2022).

In this paper, we aim to bridge the gap between in-context and meta-learning, and show that in-context learning in Transformers can be an emergent property approximating gradient-based few-shot learning within its forward pass, see Figure 1. For this to be realized, we show how Transformers **(1)** construct a loss function dependent on the data given in sequence and **(2)** learn based on gradients of that loss. We will first focus on the latter, the more elaborate learning task, in sections 2 and 3, after which we provide evidence for the former in section 4.

We summarize our contributions as follows[1]:

- We construct explicit weights for a linear self-attention layer that induces an update identical to a single step of gradient descent (GD) on a mean squared error loss. Additionally, we show how several self-attention layers can iteratively perform curvature correction improving on plain gradient descent.

- When optimized on linear regression datasets, we demonstrate that linear self-attention-only Transform-

[1]Main experiments can be reproduced with notebooks provided under the following link: https://github.com/google-research/self-organising-systems/tree/master/transformers_learn_icl_by_gd

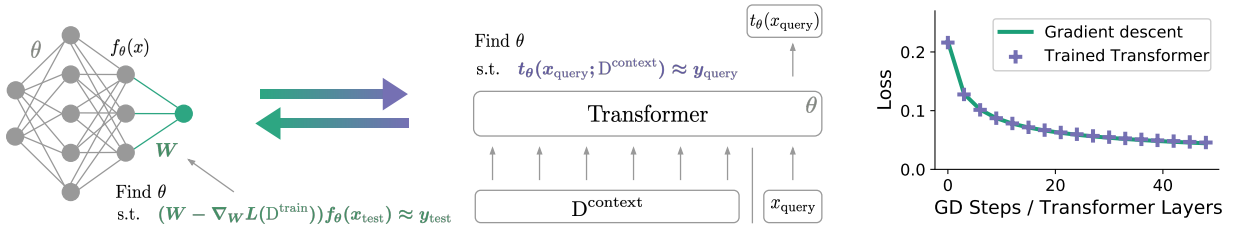


Figure 1. **Illustration of our hypothesis: gradient-based optimization and attention-based in-context learning are equivalent.** *Left*: Learning a neural network output layer by gradient descent on a dataset $\mathrm{D}^{\text{train}}$. The task-shared meta-parameters θ are obtained by meta-learning with the goal that after adjusting the neural network output layer, the model generalizes well on unseen data. *Center*: Illustration of a Transformer that adjusts its query prediction on the data given in-context i.e. $t_\theta(x_{\text{query}}; \mathrm{D}^{\text{context}})$. The weights of the Transformer are optimized to predict the next token y_{query}. *Right*: Our results confirm the hypothesis that learning with K steps of gradient descent on a dataset $\mathrm{D}^{\text{train}}$ (*green part of the left plot*) matches trained Transformers with K linear self-attention layers (*central plot*) when given $\mathrm{D}^{\text{train}}$ as in-context data $\mathrm{D}^{\text{context}}$.

ers either converge to our weight construction and therefore implement gradient descent, or generate linear models that closely align with models trained by GD, both in in- and out-of-distribution validation tasks.

- By incorporating multi-layer-perceptrons (MLPs) into the Transformer architecture, we enable solving *non*linear regression tasks within Transformers by showing its equivalence to learning a linear model on deep representations. We discuss connections to kernel regression as well as nonparametric kernel smoothing methods. Empirically, we compare meta-learned MLPs and a single step of GD on its output layer with trained Transformers and demonstrate striking similarities between the identified solutions.
- We resolve the dependency on the specific token construction by providing evidence that learned Transformers first encode incoming tokens into a format amenable to the in-context gradient descent learning that occurs in the later layers of the Transformer.

These findings allow us to connect *learning* Transformer weights and the concept of *meta-learning* a learning algorithm (Schmidhuber, 1987; Hinton & Plaut, 1987; Bengio et al., 1990; Chalmers, 1991; Schmidhuber, 1992; Thrun & Pratt, 1998; Hochreiter et al., 2001; Andrychowicz et al., 2016; Ba et al., 2016; Kirsch & Schmidhuber, 2021). In this extensive research field, meta-learning is typically regarded as learning that takes place on various time scales namely fast and slow. The slowly changing parameters control and prepare for fast adaptation reacting to sudden changes in the incoming data by e.g. a context switch. Notably, we build heavily on the concept of fast weights (Schmidhuber, 1992) which has shown to be equivalent to linear self-attention (Schlag et al., 2021) and show how optimized Transformers implement interpretable learning algorithms within their weights.

Another related meta-learning concept, termed MAML (Finn et al., 2017), aims to meta-learn a deep neural network initialization which allows for fast adaptation on novel tasks. It has been shown that in many circumstances, the solution found can be approximated well when only adapting the output layer i.e. learning a linear model on a meta-learned deep data representations (Finn et al., 2017; Finn & Levine, 2018; Gordon et al., 2019; Lee et al., 2019; Rusu et al., 2019; Raghu et al., 2020; von Oswald et al., 2021). In section 3, we show the equivalence of this framework to in-context learning implemented in a common Transformer block i.e. when combining self-attention layers with a multi-layer-perceptron.

In the light of meta-learning we show how optimizing Transformer weights can be regarded as learning on two time scales. More concretely, we find that solely through the pressure to predict correctly Transformers discover learning algorithms inside their forward computations, effectively meta-learning a learning algorithm. Recently, this concept of an emergent optimizer within a learned neural network, such as a Transformer, has been termed "mesa-optimization" (Hubinger et al., 2019). We find and describe one possible realization of this concept and hypothesize that the in-context learning capabilities of language models emerge through mechanisms similar to the ones we discuss here.

Transformers come in different "shapes and sizes", operate on vastly different domains, and exhibit varying forms of phase transitions of in-context learning (Kirsch et al., 2022; Chan et al., 2022a), suggesting variance and significant complexity of the underlying learning mechanisms. As a result, we expect our findings on linear self-attention-only Transformers to only explain a limited part of a complex process, and it may be one of many possible methods giving rise to in-context learning. Nevertheless, our approach provides an intriguing perspective on, and novel evidence for, an in-context learning mechanism that significantly differs from existing mechanisms based on associative memory (Ramsauer et al., 2020), or by the copying mechanism termed *induction heads* identified by (Olsson et al., 2022). We, therefore, state the following

Hypothesis 1 (Transformers learn in-context by gradient descent)**.** *When training Transformers on auto-regressive tasks, in-context learning in the Transformer forward pass is implemented by gradient-based optimization of an implicit auto-regressive inner loss constructed from its in-context data.*

We acknowledge work done in parallel, investigating the same hypothesis. Akyürek et al. (2023) puts forward a weight construction based on a chain of Transformer layers (including MLPs) that together implement a single step of gradient descent with weight decay. Similar to work done by Garg et al. (2022), they then show that trained Transformers match the performance of models obtained by gradient descent. Nevertheless, it is not clear that optimization finds Transformer weights that coincide with their construction.

Here, we present a much simpler construction that builds on Schlag et al. (2021) and *only requires a single linear self-attention layer* to implement a step of gradient descent. This allows us to (1) show that optimizing self-attention-only Transformers finds weights that match our weight construction (Proposition 1), demonstrating its practical relevance, and (2) explain in-context learning in shallow two layer Transformers intensively studied by Olsson et al. (2022). Therefore, although related work provides comprehensive empirical evidence that Transformers indeed seem to implement gradient descent based learning on the data given in-context, we will in the following present mechanistic verification of this hypothesis and provide compelling evidence that our construction, which implements GD in a Transformer forward pass, is found in practice.

2. Linear self-attention *can* emulate gradient descent on a linear regression task

We start by reviewing a standard multi-head self-attention (SA) layer with parameters θ. A SA layer updates each element e_j of a set of tokens $\{e_1, \ldots, e_N\}$ according to

$$\begin{aligned} e_j &\leftarrow e_j + \mathrm{SA}_\theta(j, \{e_1, \ldots, e_N\}) \\ &= e_j + \sum_h P_h V_h \mathrm{softmax}(K_h^T q_{h,j}) \end{aligned} \tag{1}$$

with P_h, V_h, K_h the projection, value and key matrices, respectively, and $q_{h,i}$ the query, all for the h-th head. To simplify the presentation, we omit bias terms here and throughout. The columns of the value $V_h = [v_{h,1}, \ldots, v_{h,N}]$ and key $K_h = [k_{h,1}, \ldots, k_{h,N}]$ matrices consist of vectors $v_{h,i} = W_{h,V} e_i$ and $k_{h,i} = W_{h,K} e_i$; likewise, the query is produced by linearly projecting the tokens, $q_{h,j} = W_{h,Q} e_j$. The parameters $\theta = \{P_h, W_{h,V}, W_{h,K}, W_{h,Q}\}_h$ of a SA layer consist of all the projection matrices, of all heads.

The self-attention layer described above corresponds to the one used in the standard Transformer model. Following Schlag et al. (2021), we now introduce our first (and only) departure from the standard model, and omit the softmax operation in equation 1, leading to the *linear* self-attention (LSA) layer $e_j \leftarrow e_j + \mathrm{LSA}_\theta(j, \{e_1, \ldots, e_N\}) = e_j + \sum_h P_h V_h K_h^T q_{h,j}$ We next show that with some simple manipulations we can relate the update performed by an LSA layer to one step of gradient descent on a linear regression loss.

Data transformations induced by gradient descent

We now introduce a reference linear model $y(x) = Wx$ parameterized by the weight matrix $W \in \mathbb{R}^{N_y \times N_x}$, and a training dataset $D = \{(x_i, y_i)\}_{i=1}^N$ comprising of input samples $x_i \in \mathbb{R}^{N_x}$ and respective labels $y_i \in \mathbb{R}^{N_y}$. The goal of learning is to minimize the squared-error loss:

$$L(W) = \frac{1}{2N} \sum_{i=1}^N \|W x_i - y_i\|^2. \tag{2}$$

One step of gradient descent on L with learning rate η yields the weight change

$$\Delta W = -\eta \nabla_W L(W) = -\frac{\eta}{N} \sum_{i=1}^N (W x_i - y_i) x_i^T. \tag{3}$$

Considering the loss after changing the weights, we obtain

$$\begin{aligned} L(W + \Delta W) &= \frac{1}{2N} \sum_{i=1}^N \|(W + \Delta W) x_i - y_i\|^2 \\ &= \frac{1}{2N} \sum_{i=1}^N \|W x_i - (y_i - \Delta y_i)\|^2 \end{aligned} \tag{4}$$

where we introduced the transformed targets $y_i - \Delta y_i$ with $\Delta y_i = \Delta W x_i$. Thus, we can view the outcome of a gradient descent step as an update to our regression loss (equation 2), where data, and not weights, are updated. Note that this formulation is closely linked to predicting based on nonparametric kernel smoothing, see Appendix A.8 for a discussion.

Returning to self-attention mechanisms and Transformers, we consider an in-context learning problem where we are given N context tokens together with an extra query token, indexed by $N+1$. In terms of our linear regression problem, the N context tokens $e_j = (x_j, y_j) \in \mathbb{R}^{N_x + N_y}$ correspond to the N training points in D, and the N+1-th token $e_{N+1} = (x_{N+1}, y_{N+1}) = (x_{\text{test}}, \hat{y}_{\text{test}}) = e_{\text{test}}$ to the test input x_{test} and the corresponding prediction $\hat{y}_{\text{test}}$. We use the terms training and in-context data interchangeably, as well as query and test token/data, as we establish their equivalence now.

Transformations induced by gradient descent and a linear self-attention layer can be equivalent

We have re-cast the task of learning a linear model as directly modifying the data, instead of explicitly computing and returning the weights of the model (equation 4). We proceed to establish a connection between self-attention and gradient descent. We provide a construction where learning takes place simultaneously by directly updating all tokens, including the test token, through a linear self-attention layer. In other words, the token produced in response to a query (test) token is transformed from its initial value $W_0 x_{\text{test}}$, where W_0 is the initial value of W, to the post-learning prediction $\hat{y} = (W_0 + \Delta W) x_{\text{test}}$ obtained after one gradient descent step.

Proposition 1. *Given a 1-head linear attention layer and the tokens $e_j = (x_j, y_j)$, for $j = 1, \ldots, N$, one can construct key, query and value matrices W_K, W_Q, W_V as well as the projection matrix P such that a Transformer step on every token e_j is identical to the gradient-induced dynamics $e_j \leftarrow (x_j, y_j) + (0, -\Delta W x_j) = (x_j, y_j) + P V K^T q_j$ such that $e_j = (x_j, y_j - \Delta y_j)$. For the test data token (x_{N+1}, y_{N+1}) the dynamics are identical.*

The simple construction can be found in Appendix A.1 and we denote the corresponding self-attention weights by θ_{GD}.

Below, we provide some additional insights on what is needed to implement the provided LSA-layer weight construction, and further details on what it can achieve:

- **Full self-attention.** Our dynamics model training is based on in-context tokens only, i.e., only $e_1, \ldots, e_N$ are used for computing key and value matrices; the query token e_{N+1} (containing test data) is excluded. This leads to a linear function in x_{test} as well as to the correct ΔW, induced by gradient descent on a loss consisting only of the training data. This is a minor deviation from full self-attention. In practice, this modification can be dropped, which corresponds to assuming that the underlying initial weight matrix is zero, $W_0 \approx 0$, which makes ΔW in equation 8 independent of the test token even if incorporating it in the key and value matrices. In our experiments, we see that these assumptions are met when initializing the attention weights θ to small values.

- **Reading out predictions.** When initializing the y-entry of the test-data token with $-W_0 x_{N+1}$, i.e. $e_{\text{test}} = (x_{\text{test}}, -W_0 x_{\text{test}})$, the test-data prediction $\hat{y}$ can be easily read out by simply multiplying again by -1 the updated token, since $-y_{N+1} + \Delta y_{N+1} = -(y_{N+1} - \Delta y_{N+1}) = y_{N+1} + \Delta W x_{N+1}$. This can easily be done by a final projection matrix, which incidentally is usually found in Transformer architectures. Importantly, we see that a single head of self-attention is sufficient to transform our training targets as well as the test prediction simultaneously.

- **Uniqueness.** We note that the construction is not unique; in particular, it is only required that the products PW_V as well as $W_K W_Q$ match the construction. Furthermore, since no nonlinearity is present, any rescaling s of the matrix products, i.e., $PW_V s$ and $W_K W_Q / s$, leads to an equivalent result. If we correct for these equivalent formulations, we can experimentally verify that weights of our learned Transformers indeed match the presented construction.

- **Meta-learned task-shared learning rates.** When training self-attention parameters θ across a family of in-context learning tasks τ, where the data $(x_{\tau,i}, y_{\tau,i})$ follows a certain distribution, the learning rate can be implicitly (meta-)learned such that an optimal loss reduction (averaged over tasks) is achieved given a fixed number of update steps. In our experiments, we find this to be the case. This kind of meta-learning to improve upon plain gradient descent has been leveraged in numerous previous approaches for deep neural networks (Li et al., 2017; Lee & Choi, 2018; Park & Oliva, 2019; Zhao et al., 2020; Flennerhag et al., 2020).

- **Task-specific data transformations.** A self-attention layer is in principle further capable of exploiting statistics in the current training data samples, beyond modeling task-shared curvature information in θ. More concretely, a LSA layer updates an input sample according to a data transformation $x_j \leftarrow x_j + \Delta x_j = (I + P(X)V(X)K(X)^T W_Q) x_j = H_\theta(X) x_j$, with X the $N_x \times N$ input training data matrix, when neglecting influences by target data y_i. Through $H_\theta(X)$, a LSA layer can encode in θ an algorithm for carrying out data transformations which depend on the actual input training samples in X. In our experiments, we see that trained self-attention learners employ a simple form of $H(X)$ and that this leads to substantial speed ups in for GD and TF learning.

3. Trained Transformers *do* mimic gradient descent on linear regression tasks

We now experimentally investigate whether trained attention-based models implement gradient-based in-context learning in their forward passes. We gradually build up from single linear self-attention layers to multi-layer nonlinear models, approaching full Transformers. In this section, we follow the assumption of Proposition 1 tightly and construct our tokens by concatenating input and target data, $e_j = (x_j, y_j)$ for $1 \leq j \leq N$, and our query token by concatenating the test input and a zero vector, $e_{N+1} = (x_{\text{test}}, 0)$. We show how to lift this assumption in the last section of the

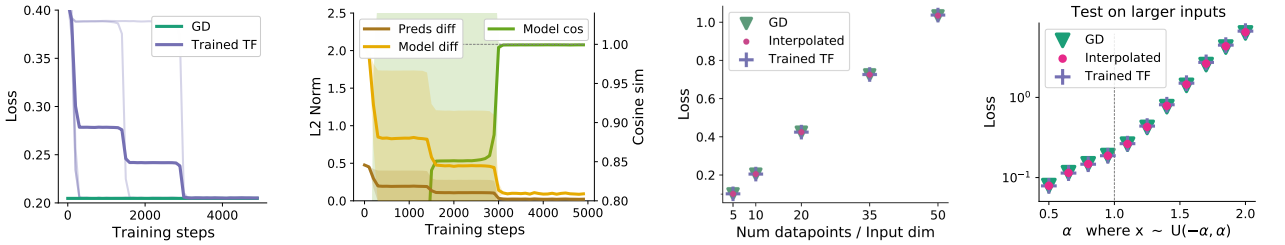


Figure 2. **Comparing one step of GD with a trained single linear self-attention layer.** *Outer left:* Trained single LSA layer performance is identical to the one of gradient descent. *Center left*: Almost perfect alignment of GD and the model generated by the SA layer after training, measured by cosine similarity and the L2 distance between models as well as their predictions. *Center right:* Identical loss of GD, the LSA layer model as well as the model obtained by interpolating between the construction and the optimized LSA layer weights for different $N = N_x$. *Outer right:* The trained LSA layer, gradient descent and their interpolation show identically loss (in log-scale) when provided input data different than during training i.e. with scale of 1. We display the mean/std. or the single runs of 5 seeds.

paper. The prediction $\hat{y}_\theta(\{e_{\tau,1}, \ldots, e_{\tau,N}\}, e_{\tau,N+1})$ of the attention-based model, which depends on all tokens and on the parameters θ, is read-out from the y-entry of the updated $N+1$-th token as explained in the previous section.

The objective of training, visualized in Figure 1, is to minimize the expected squared prediction error, averaged over tasks $\min_\theta \mathbb{E}_\tau[||\hat{y}_\theta(\{e_{\tau,1}, \ldots, e_{\tau,N}\}, e_{\tau,N+1}) - y_{\tau,\text{test}}||^2]$. We achieve this by minibatch online minimization (by Adam (Kingma & Ba, 2014)): At every optimization step, we construct a batch of novel training tasks and take a step of stochastic gradient descent on the loss function:

$$\mathcal{L}(\theta) = \frac{1}{B}\sum_{\tau=1}^{B} ||\hat{y}_\theta(\{e_{\tau,i}\}_{i=1}^{N}, e_{\tau,N+1}) - y_{\tau,\text{test}}||^2 \quad (5)$$

where each task (context) τ consists of in-context training data $D_\tau = \{(x_{\tau,i}, y_{\tau,i})\}_{i=1}^{N}$ and test point $(x_{\tau,N+1}, y_{\tau,N+1})$, which we use to construct our tokens $\{e_{\tau,i}\}_{i=1}^{N+1}$ as described above. We denote the optimal parameters found by this optimization process by θ^*. In our setup, finding θ^* may be thought of as meta-learning, while learning a particular task τ corresponds to simply evaluating the model $\hat{y}_\theta(\{e_{\tau,1}, \ldots, e_{\tau,N}\}, e_{\tau,N+1})$. Note that we therefore never see the exact same training task twice during training. See Appendix A.12, especially Figure 16 for an analyses when using a fixed dataset size which we cycle over during training.

We focus on solvable tasks and similarly to Garg et al. (2022) generate data for each task using a teacher model with parameters $W_\tau \sim \mathcal{N}(0, I)$. We then sample $x_{\tau,i} \sim U(-1, 1)^{n_I}$ and construct targets using the task-specific teacher model, $y_{\tau,i} = W_\tau x_{\tau,i}$. In the majority of our experiments we set the dimensions to $N = n_I = 10$ and $n_O = 1$. Since we use a noiseless teacher for simplicity, we can expect our regression tasks to be well-posed and analytically solvable as we only compute a loss on the Transformers last token, which stands in contrast to usual autoregressive training and the training setup of Garg et al. (2022). Full details and results for training with a fixed training set size may be found in Appendix A.12.

One-step of gradient descent vs. a single trained self-attention layer

Our first goal is to investigate whether a trained single, linear self-attention layer can be explained by the provided weight construction that implements GD. To that end, we compare the predictions made by a LSA layer with trained weights θ^* (which minimize equation 5) and with constructed weights θ_{GD} (which satisfy Proposition 1).

Recall that a LSA layer yields the prediction $\hat{y}_\theta(x_{\text{test}}) = e_{N+1} + \text{LSA}_\theta(\{e_1, \ldots, e_N\}, e_{N+1}) = \Delta W_{\theta,D} x_{\text{test}}$, which is linear in x_{test}. We denote by $\Delta W_{\theta,D}$ the matrix generated by the LSA layer following the construction provided in Proposition 1, with query token e_{N+1} set such that the initial prediction is set to zero, $\hat{y}_{\text{test}} = 0$. We compare $\hat{y}_\theta(x_{\text{test}})$ to the prediction of the control LSA $\hat{y}_{\theta_{\text{GD}}}(x_{\text{test}})$, which under our token construction corresponds to a linear model trained by one step of gradient descent starting from $W_0 = 0$. For this control model, we determine the optimal learning rate η by minimizing $\mathcal{L}(\eta)$ over a training set of 10^4 tasks through line search, with $\mathcal{L}(\eta)$ defined analogously to equation 5.

More concretely, to compare trained and constructed LSA layers, we sample $T_{\text{val}} = 10^4$ validation tasks and record the following quantities, averaged over validation tasks: (1) the difference in predictions measured with the L2 norm, $\|\hat{y}_\theta(x_{\tau,\text{test}}) - \hat{y}_{\theta_{\text{GD}}}(x_{\tau,\text{test}})\|$, (2) the cosine similarity between the sensitivities $\frac{\partial \hat{y}_{\theta_{\text{GD}}}(x_{\tau,\text{test}})}{\partial x_{\text{test}}}$ and $\frac{\partial \hat{y}_\theta(x_{\tau,\text{test}})}{\partial x_{\text{test}}}$ as well as (3) their difference $\|\frac{\partial \hat{y}_{\theta_{\text{GD}}}(x_{\tau,\text{test}})}{\partial x_{\text{test}}} - \frac{\partial \hat{y}_\theta(x_{\tau,\text{test}})}{\partial x_{\text{test}}}\|$ again according to the L2 norm, which in both cases yields the explicit models computed by the algorithm. We show the results of these comparisons in Figure 2. We find an excellent agreement between the two models over a wide range of hyperparameters. We note that as we do not have direct access to the initialization of W in the attention-based learners (it is hidden in θ), we cannot expect the models to agree exactly.

Although the above metrics are important to show similarities between the resulting learned models (in-context

vs. gradient-based), the underlying algorithms could still be different. We therefore carry out an extended set of analyses:

1. **Interpolation.** We take inspiration on recent work (Benzing et al., 2022; Entezari et al., 2021) that showed approximate equivalence of models found by SGD after permuting weights within the trained neural networks. Since our models are deep linear networks with respect to x_{test} we only correct for scaling mismatches between the two models – in this case the construction that implements GD and the trained weights. As shown in Figure 2, we observe (and can actually inspect by eye, see Appendix Figure 9) that a simple scaling correction on the trained weights is enough to recover the weight construction implementing GD. This leads to an identical loss of GD, the trained Transformer and the linearly interpolated weights $\theta_{\text{I}} = (\theta + \theta_{\text{GD}})/2$. See details in Appendix A.3 on how our weight correction and interpolation is obtained.

2. **Out-of-distribution validation tasks.** To test if our in-context learner has found a generalizable update rule, we investigate how GD, the trained LSA layer and its interpolation behave when providing in-context data in regimes different to the ones used during training. We therefore visualize the loss increase when (1) sampling the input data from $U(-\alpha, \alpha)^{N_x}$ or (2) scaling the teacher weights by α as αW when sampling validation tasks. For both cases, we set $\alpha = 1$ during training. We again observe that when training a single linear self-attention Transformer, for both interventions, the Transformer performs equally to gradient descent outside of this training setups, see Figure 2 as well Appendix Figure 6. Note that the loss obtained through gradient descent also starts degrading quickly outside the training regime. Since we tune the learning rate for the input range $[-1, 1]$ and one gradient step, tasks with larger input range will have higher curvature and the optimal learning rate for smaller ranges will lead to divergence and a drastic increase in loss also for GD.

3. **Repeating the LSA update.** Since we claim that a single trained LSA layer implements a GD-like learning rule, we further test its behavior when applying it repeatedly, not only once as in training. After we correct the learning rate of both algorithms, i.e. for GD and the trained Transformer with a dampening parameter $\lambda = 0.75$ (details in Appendix A.6), we see an identical loss decrease of both GD and the Transformer, see Figure 1.

To conclude, we present evidence that optimizing a single LSA layer to solve linear regression tasks finds weights that (approximately) coincide with the LSA-layer weight construction of Proposition 1, hence implementing a step of gradient descent, leading to the same learning capabilities on in- and out-of-distribution tasks. We comment on the random seed dependent phase transition of the loss during training in Appendix A.11.

Multiple steps of gradient descent vs. multiple layers of self-attention

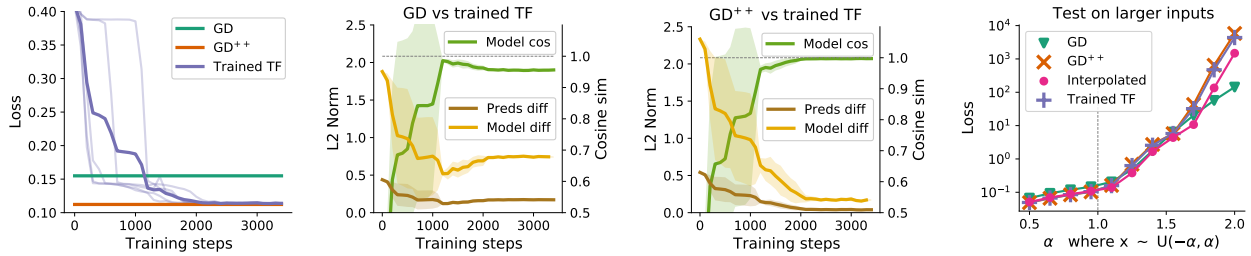
We now turn to deep linear self-attention-only Transformers. The construction we put forth in Proposition 1, can be immediately stacked up over K layers; in this case, the final prediction can be read out from the last layer as before by negating the y-entry of the last test token: $-y_{N+1} + \sum_{k=1}^{K} \Delta y_{k,N+1} = -(y_{N+1} - \sum_{k=1}^{K} \Delta y_{k,N+1}) = y_{N+1} + \sum_{k=1}^{K} \Delta W_k x_{N+1}$, where $y_{k,N+1}$ are the test token values at layer k, and $\Delta y_{k,N+1}$ the change in the y-entry of the test token after applying the k-th step of self-attention, and ΔW_k the k-th implicit change in the underlying linear model parameters W. When optimizing such Transformers with K layers, we observe that these models generally outperform K steps of plain gradient descent, see Figure 3. Their behavior is however well described by a variant of gradient descent, for which we tune a single parameter γ defined through the transformation function $H(X)$ which transforms the input data according to $x_j \leftarrow H(X)x_j$, with $H(X) = (I - \gamma X X^T)$. We term this gradient descent variant GD^{++} which we explain and analyze in Appendix A.10.

To analyze the effect of adding more layers to the architecture, we first turn to the arguably simplest extension of a single SA layer and analyze a *recurrent* or *looped* 2-layer LSA model. Here, we simply repeatably apply the same layer (with the same weights) multiple times i.e. drawing the analogy to learning an iterative algorithm that applies the same logic multiple times.

Somewhat surprisingly, we find that the trained model surpasses plain gradient descent, which also results in decreasing alignment between the two models (see center left column), and the recurrent Transformer realigns perfectly with GD^{++} while matching its performance on in- and out-of distribution tasks. Again, we can interpolate between the Transformer weights found by optimization and the LSA-weight construction with learned η, γ, see Figure 3 & 6.

We next consider deeper, non-recurrent 5-layer LSA-only Transformers, with different parameters per layer (i.e. no weight tying). We see that a different GD learning rate as well as γ per step (layer) need to be tuned to match the Transformer performance. This slight modification leads again to almost perfect alignment between the trained TF and GD^{++} with in this case 10 additional parameters and

(a) Comparing two steps of gradient descent with trained *recurrent* two-layer Transformers.



(b) Comparing five steps of gradient descent with trained five-layer Transformers.

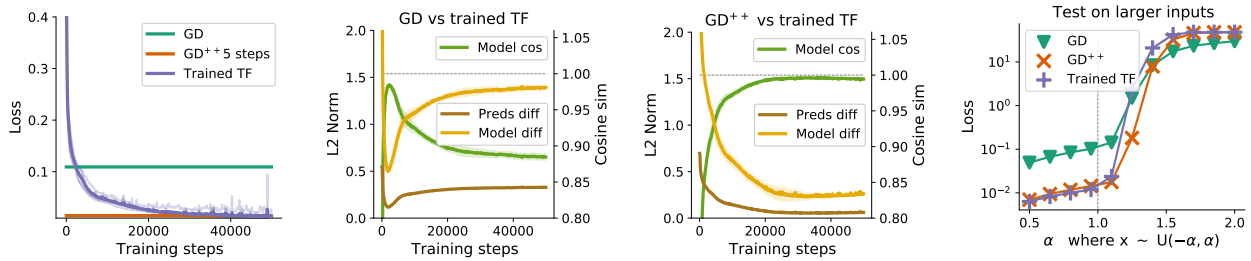


Figure 3. Far left column: The trained TF performance surpasses standard GD but matches GD^{++}, our GD variant with simple iterative data transformation. On both cases, we tuned the gradient descent learning rates as well as the scalar γ which governs the data transformation $H(X)$. *Center left & center right columns*: We measure the alignment between the GD as well as the GD^{++} models and the trained TF. In both cases the TF aligns well with GD in the beginning of training but aligns much better with GD^{++} after training. *Far right column:* TF performance (in log-scale) mimics the one of GD^{++} well when testing on OOD tasks ($\alpha \neq 1$).

loss close to 0, see Figure 3. Nevertheless, we see that the naive correction necessary for model interpolation used in the aforementioned experiments is not enough to interpolate without a loss increase. We leave a search for better weight corrections to future work. We further study Transformers with different depths for recurrent as well as non-recurrent architectures with multiple heads and equipped with MLPs, and find qualitatively equivalent results, see Appendix Figure 7 and Figure 8. Additionally, in Appendix A.9, we provide results obtained when using softmax SA layers as well as LayerNorm, thus essentially retrieving the standard Transformer architecture. We again observe and are able to explain (after slight architectural modifications) good learning performance and as well as alignment with the construction of Proposition 1, though worse than when using linear self-attention. These findings suggest that the in-context learning abilities of the standard Transformer with these common architecture choices can be explained by the gradient-based learning hypothesis explored here. Our findings also question the ubiquitous use of softmax attention, and suggest further investigation is warranted into the performance of linear vs. softmax SA layers in real-world learning tasks, as initiated by Schlag et al. (2021).

Transformers solve nonlinear regression tasks by gradient descent on deep data representations

It is unreasonable to assume that the astonishing in-context learning flexibility observed in large Transformers is explained by gradient descent on linear models. We now show that this limitation can be resolved by incorporating one additional element of fully-fledged Transformers: preceding self-attention layers by MLPs enables learning linear models by gradient descent on deep representations which motivates our illustration in Figure 1. Empirically, we demonstrate this by solving non-linear sine-wave regression tasks, see Figure 4. Experimental details can be found in Appendix A.7. We state

Proposition 2. *Given a Transformer block i.e. a MLP $m(e)$ which transforms the tokens $e_j = (x_j, y_j)$ followed by an attention layer, we can construct weights that lead to gradient descent dynamics descending $\frac{1}{2N}\sum_{i=1}^{N} ||Wm(x_i) - y_i||^2$. Iteratively applying Transformer blocks therefore can solve kernelized least-squares regression problems with kernel function $k(x, y) = m(x)^\top m(y)$ induced by the MLP $m(\cdot)$.*

A detailed discussion on this form of kernel regression as well as kernel smoothing w/wo softmax nonlinearity through gradient descent on the data can be found in Appendix A.8. The way MLPs transform data in Transformers diverges from the standard meta-learning approach, where a task-shared *input* embedding network is optimized by backpropagation-through-training to improve the learning performance of a task-specific readout (e.g., Raghu et al., 2020; Lee et al., 2019; Bertinetto et al., 2019). On the other hand, given our token construction in Proposition 1, MLPs in Transformers intriguingly process both inputs *and* targets. The output of this transformation is then processed by a sin-

gle linear self-attention layer, which, according to our theory, is capable of implementing gradient descent learning. We compare the performance of this Transformer model, where all weights are learned, to a control Transformer where the final LSA weights are set to the construction θ_{GD} which is therefore identical to training an MLP by backpropagation through a GD updated output layer.

Intriguingly, both obtained functions show again surprising similarity on (1) the initial (meta-learned) prediction, read out after the MLP, and (2) the final prediction, after altering the output of the MLP through GD or the self-attention layer. This is again reflected in our alignment measures that now, since the obtained models are nonlinear w.r.t. x_{test}, only represent the two first parts of the Taylor approximation of the obtained functions. Our results serve as a first demonstration of how MLPs and self-attention layers can interplay to support nonlinear in-context learning, allowing to fine-tune deep data representations by gradient descent. Investigating the interplay between MLPs and SA-layer in deep TFs is left for future work.

4. Do self-attention layers build regression tasks?

The construction provided in Proposition 1 and the previous experimental section relied on a token structure where both input and output data are concatenated into a single token. This design is different from the way tokens are typically built in most of the related work dealing with simple few-shot learning problems as well as in e.g. language modeling. We therefore ask: Can we overcome the assumption required in Proposition 1 and allow a Transformer to build the required token construction on its own? This motivates

Proposition 3. *Given a 1-head linear or softmax attention layer and the token construction* $e_{2j} = (x_j), e_{2j+1} = (0, y_j)$ *with a zero vector* 0 *of dim* $N_x - N_y$ *and concatenated positional encodings, one can construct key, query and value matrix* W_K, W_Q, W_V *as well as the projection matrix* P *such that all tokens* e_j *are transformed into tokens equivalent to the ones required in Proposition 1.*

The construction and its discussion can be found in Appendix A.5. To provide evidence that copying is performed in trained Transformers, we optimize a two-layer self-attention circuit on in-context data where alternating tokens include input or output data i.e. $e_{2j} = (x_j)$ and $e_{2j+1} = (0, y_j)$. We again measure the loss as well as the mean of the norm of the partial derivative of the first layer's output w.r.t. the input tokens during training, see Figure 5. First, the training speeds are highly variant given different training seeds, also reported in Garg et al. (2022). Nevertheless, the Transformer is able to match the performance of a *single* (not two) step gradient descent. Interestingly, before the Transformer performance jumps to the one of GD, token e_j transformed by the first self-attention layer becomes notably dependant on the neighboring token e_{j+1} while staying independent on the others which we denote as e_{other} in Figure 5.

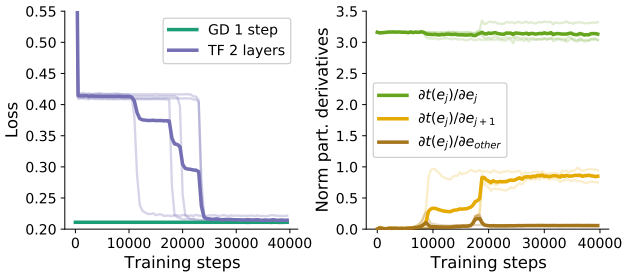


Figure 5. **Training a two layer SA-only Transformer using the standard token construction.** *Left:* The loss of trained TFs matches one step of GD, not two, and takes an order of magnitude longer to train. *Right*: Norm of the partial derivatives of the output of the first self-attention layer w.r.t. input tokens. Before the Transformer performance jumps to the one of GD, the first layer becomes highly sensitive to the next token.

We interpret this as evidence for a copying mechanism of the Transformer's first layer to merge input and output data into single tokens as required by Proposition 1. Then, in the second layer the Transformer performs a single step of GD. Notably, we were not able to train the Transformer with linear self-attention layers, but had to incorporate the softmax operation in the first layer. These preliminary findings support the study of Olsson et al. (2022) showing that softmax self-attention layers easily learn to copy; we confirm this claim, and further show that such copying allows the Transformer to proceed by emulating gradient-based learning in the second or deeper attention layers.

We conclude that copying through (softmax) attention layers is the second crucial mechanism for in-context learning in Transformers. This operation enables Transformers to merge data from different tokens and then to compute dot products of input and target data downstream, allowing for in-context learning by gradient descent to emerge.

5. Discussion

Transformers show remarkable in-context learning behavior. Mechanisms based on attention, associative memory and copying by induction heads are currently the leading explanations for this remarkable feature of learning within the Transformer forward pass. In this paper, we put forward the hypothesis, similar to Garg et al. (2022) and Akyürek et al. (2023), that Transformer's in-context learning is driven by gradient descent, in short – *Transformers learn to learn by gradient descent based on their context*. Viewed through the lens of meta-learning, learning Transformer weights corresponds to the outer-loop which then enables the forward

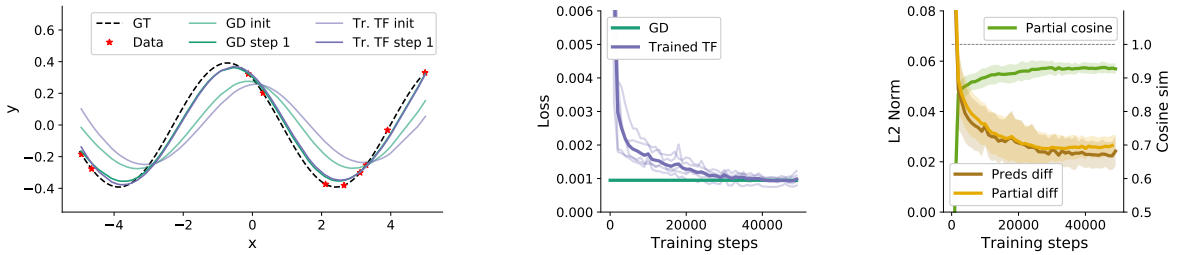


Figure 4. **Sine wave regression: comparing trained Transformers with meta-learned MLPs for which we adjust the output layer with one step of gradient descent.** *Left:* Plots of the learned initial functions as well as the adjusted functions through either a layer of self-attention or a step of GD. We observe similar initial functions as well as solutions for the trained TF compared fine-tuning a meta-learned MLP. *Center*: The performance of the trained Transformer is matched by meta-learned MLPs. *Left*: We observe strong alignment when comparing the prediction as well as the partial derivatives of the the meta-learned MLP and the trained Transformer.

pass to transform tokens by gradient-based optimization.

To provide evidence for this hypothesis, we build on Schlag et al. (2021) that already provide a linear self-attention layer variant with (fast-)inner loop learning by the error-correcting delta rule (Widrow & Hoff, 1960). We diverge from their setting and focus on (in-context) learning where we specifically construct a dataset by considering neighboring elements in the input sequence as input- and target training pairs, see assumptions of Proposition 1. This construction could be realized, for example, due to the model learning to implement a copying layer, see section 4 and proposition 3, and allows us to provide a simple and different construction to Schlag et al. (2021) that solely is built on the standard linear, and approximately softmax, self-attention layer but still implements gradient descent based learning dynamics. We, therefore, are able to explain gradient descent based learning in these standard architectures. Furthermore, we extend this construction based on a single self-attention layer and provide an explanation of how deeper K-layer Transformer models implement principled K-step gradient descent learning, which deviates again from Schlag et al. and allows us to identify that deep Transformers implement GD++, an accelerated version of gradient descent.

We highlight that our construction of gradient descent and GD++ is not suggestive but when training multi-layer self-attention-only Transformers on simple regression tasks, we provide strong evidence that the construction is actually found. This allows us, at least in our restricted problems settings, to explain mechanistically in-context learning in trained Transformers and its close resemblance to GD observed by related work. Further work is needed to incorporate regression problems with noisy data and weight regularization into our hypothesis. We speculate aspects of learning in these settings are meta-learned – e.g., the weight magnitudes to be encoded in the self-attention weights. Additionally, we did not analyze logistic regression for which one possible weight construction is already presented in Zhmoginov et al. (2022).

Our refined understanding of in-context learning based on gradient descent motives us to investigate how to improve it. We are excited about several avenues of future research. First, to exceed upon a single step of gradient descent in every self-attention layer it could be advantageous to incorporate so called *declarative* nodes (Amos & Kolter, 2017; Bai et al., 2019; Gould et al., 2021; Zucchet & Sacramento, 2022) into Transformer architectures. This way, we would treat a single self-attention layer as the solution of a fully optimized regression loss leading to possibly more efficient architectures. Second, our findings are restricted to small Transformers and simple regression problems. We are excited to delve deeper into research trying to understand how further mechanistic understanding of Transformers and in-context learning in larger models is possible and to what extend. Third, we are excited about targeted modifications to Transformer architectures, or their training protocols, leading to improved gradient descent based learning algorithms or allow for alternative in-context learners to be implemented within Transformer weights, augmenting their functionality, as e.g. in Dai et al. (2023). Finally, it would be interesting to analyze in-context learning in HyperTransformers (Zhmoginov et al., 2022) that produce weights for target networks and already offer a different perspective on merging Transformers and meta-learning. There, Transformers transform weights instead of data and could potentially allow for gradient computations of weights deep inside the target network lifting the limitation of GD on linear models analyzed here.

Acknowledgments

João Sacramento and Johannes von Oswald deeply thank Angelika Steger for her support and guidance. The authors also thank Seijin Kobayashi, Marc Kaufmann, Nicolas Zucchet, Yassir Akram, Guillaume Obozinski and Mark Sandler for many valuable insights throughout the project and Dale Schuurmans and Timothy Nguyen for their valuable comments on the manuscript. João Sacramento was supported by an Ambizione grant (PZ00P3_186027) from the Swiss National Science Foundation and an ETH Research Grant (ETH-23 21-1).

References

Akyürek, E., Schuurmans, D., Andreas, J., Ma, T., and Zhou, D. What learning algorithm is in-context learning? investigations with linear models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=0g0X4H8yN4I.

Amos, B. and Kolter, J. Z. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, 2017.

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. In *Advances in Neural Information Processing Systems*, 2016.

Ba, J., Hinton, G. E., Mnih, V., Leibo, J. Z., and Ionescu, C. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems 29*, 2016.

Bai, S., Kolter, J. Z., and Koltun, V. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 2019.

Bengio, Y., Bengio, S., and Cloutier, J. Learning a synaptic learning rule. Technical report, Université de Montréal, Département d'Informatique et de Recherche opérationnelle, 1990.

Benzing, F., Schug, S., Meier, R., von Oswald, J., Akram, Y., Zucchet, N., Aitchison, L., and Steger, A. Random initialisations performing above chance and how to find them. *OPT2022: 14th Annual Workshop on Optimization for Machine Learning*, 2022.

Bertinetto, L., Henriques, J. F., Torr, P. H. S., and Vedaldi, A. Meta-learning with differentiable closed-form solvers. In *International Conference on Learning Representations*, 2019.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. End-to-end object detection with transformers. In *Computer Vision – ECCV 2020*. Springer International Publishing, 2020.

Chalmers, D. J. The evolution of learning: an experiment in genetic connectionism. In Touretzky, D. S., Elman, J. L., Sejnowski, T. J., and Hinton, G. E. (eds.), *Connectionist Models*, pp. 81–90. Morgan Kaufmann, 1991.

Chan, S. C. Y., Dasgupta, I., Kim, J., Kumaran, D., Lampinen, A. K., and Hill, F. Transformers generalize differently from information stored in context vs in weights. *arXiv preprint arXiv:2210.05675*, 2022a.

Chan, S. C. Y., Santoro, A., Lampinen, A. K., Wang, J. X., Singh, A., Richemond, P. H., McClelland, J., and Hill, F. Data distributional properties drive emergent in-context learning in transformers. *Advances in Neural Information Processing Systems*, 2022b.

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J., and Weller, A. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Ua6zuk0WRH.

Dai, D., Sun, Y., Dong, L., Hao, Y., Ma, S., Sui, Z., and Wei, F. Why can GPT learn in-context? language models implicitly perform gradient descent as meta-optimizers. In *ICLR 2023 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2023. URL https://openreview.net/forum?id=fzbHRjAd8U.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Entezari, R., Sedghi, H., Saukh, O., and Neyshabur, B. The role of permutation invariance in linear mode connectivity of neural networks. *arXiv preprint arXiv:2110.06296*, 2021.

Finn, C. and Levine, S. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HyjC5yWCW.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017.

Flennerhag, S., Rusu, A. A., Pascanu, R., Visin, F., Yin, H., and Hadsell, R. Meta-learning with warped gradient descent. In *International Conference on Learning Representations*, 2020.

Garg, S., Tsipras, D., Liang, P., and Valiant, G. What can transformers learn in-context? a case study of simple function classes. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=flNZJ2eOet.

Gordon, J., Bronskill, J., Bauer, M., Nowozin, S., and Turner, R. Meta-learning probabilistic inference for prediction. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HkxStoC5F7.

Gould, S., Hartley, R., and Campbell, D. J. Deep declarative networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Gulati, A., Qin, J., Chiu, C.-C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., Wu, Y., and Pang, R. Conformer: Convolution-augmented transformer for speech recognition. *arXiv preprint arXiv:2005.08100*, 2020.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Hinton, G. E. and Plaut, D. C. Using fast weights to deblur old memories. 1987.

Hochreiter, S., Younger, A. S., and Conwell, P. R. Learning to learn using gradient descent. In Dorffner, G., Bischof, H., and Hornik, K. (eds.), *Artificial Neural Networks — ICANN 2001*, pp. 87–94, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-44668-2.

Hubinger, E., van Merwijk, C., Mikulik, V., Skalse, J., and Garrabrant, S. Risks from learned optimization in advanced machine learning systems. *arXiv [cs.AI]*, Jun 2019. URL http://arxiv.org/abs/1906.01820.

Irie, K., Schlag, I., Csordás, R., and Schmidhuber, J. Going beyond linear transformers with recurrent fast weight programmers. *CoRR*, abs/2106.06295, 2021. URL https://arxiv.org/abs/2106.06295.

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

Kirsch, L. and Schmidhuber, J. Meta learning backpropagation and improving it. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=hhU9TEvB6AF.

Kirsch, L., Harrison, J., Sohl-Dickstein, J., and Metz, L. General-purpose in-context learning by meta-learning transformers. In *Sixth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=t6tA-KB4dO.

Lee, K., Maji, S., Ravichandran, A., and Soatto, S. Meta-learning with differentiable convex optimization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Lee, Y. and Choi, S. Gradient-based meta-learning with learned layerwise metric and subspace. In *International Conference on Machine Learning*, 2018.

Li, Z., Zhou, F., Chen, F., and Li, H. Meta-SGD: Learning to learn quickly for few shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., and Neubig, G. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586*, 2021.

Nadaraya, E. A. On estimating regression. *Theory of Probability & its Applications*, 9(1):141–142, 1964.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Park, E. and Oliva, J. B. Meta-curvature. In *Advances in Neural Information Processing Systems*, 2019.

Power, A., Burda, Y., Edwards, H., Babuschkin, I., and Misra, V. Grokking: Generalization beyond overfitting on small algorithmic datasets. abs/2201.02177, 2022.

Raghu, A., Raghu, M., Bengio, S., and Vinyals, O. Rapid learning or feature reuse? Towards understanding the effectiveness of MAML. In *International Conference on Learning Representations*, 2020.

Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Adler, T., Gruber, L., Holzleitner, M., Pavlović, M., Sandve, G. K., Greiff, V., Kreil, D., Kopp, M., Klambauer, G., Brandstetter, J., and Hochreiter, S. Hopfield networks is all you need. *arXiv preprint arXiv:2008.02217*, 2020.

Rusu, A. A., Rao, D., Sygnowski, J., Vinyals, O., Pascanu, R., Osindero, S., and Hadsell, R. Meta-learning with latent embedding optimization. In *International Conference on Learning Representations*, 2019.

Schlag, I., Irie, K., and Schmidhuber, J. Linear transformers are secretly fast weight programmers. In *ICML*, 2021.

Schmidhuber, J. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. Diploma thesis, Institut für Informatik, Technische Universität München, 1987.

Schmidhuber, J. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992. doi: 10.1162/neco.1992.4.1.131.

Thrun, S. and Pratt, L. *Learning to learn*. Springer US, 1998.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2017.

von Oswald, J., Zhao, D., Kobayashi, S., Schug, S., Caccia, M., Zucchet, N., and Sacramento, J. Learning where to learn: Gradient sparsity in meta and continual learning. In *Advances in Neural Information Processing Systems*, 2021.

Watson, G. S. Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, pp. 359–372, 1964.

Widrow, B. and Hoff, M. E. Adaptive switching circuits. In *1960 IRE WESCON Convention Record, Part 4*, pp. 96–104, New York, 1960. IRE.

Yun, S., Jeong, M., Kim, R., Kang, J., and Kim, H. J. Graph transformer networks. In Wallach, H., Larochelle, H., Beygelzimer, A., dÁlché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, 2019.

Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

Zhao, D., Kobayashi, S., Sacramento, J., and von Oswald, J. Meta-learning via hypernetworks. In *NeurIPS Workshop on Meta-Learning*, 2020.

Zhmoginov, A., Sandler, M., and Vladymyrov, M. HyperTransformer: Model generation for supervised and semi-supervised few-shot learning. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 27075–27098. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/zhmoginov22a.html.

Zucchet, N. and Sacramento, J. Beyond backpropagation: bilevel optimization through implicit differentiation and equilibrium propagation. *Neural Computation*, 34(12), December 2022.

A. Appendix

A.1. Proposition 1

First, we highlight the dependency on the tokens e_i of the linear self-attention operation

$$e_j \leftarrow e_j + \text{LSA}_\theta(\{e_1, \dots, e_N\}) = e_j + \sum_h P_h V_h K_h^T q_{h,j} = e_j + \sum_h P_h \sum_i v_{h,i} \otimes k_{h,i} q_{h,j}$$
$$= e_j + \sum_h P_h W_{h,V} \sum_i e_{h,i} \otimes e_{h,i} W_{h,K}^T W_{h,Q} e_j \tag{6}$$

with $\otimes$ the outer product between two vectors. With this we can now easily draw connections to one step of gradient descent on $L(W) = \frac{1}{2N} \sum_{i=1}^N \|W x_i - y_i\|^2$ with learning rate η which yields weight change

$$\Delta W = -\eta \nabla_W L(W) = -\frac{\eta}{N} \sum_{i=1}^N (W x_i - y_i) x_i^T. \tag{7}$$

We first restate

Proposition 1. *Given a 1-head linear attention layer and the tokens $e_j = (x_j, y_j)$, for $j = 1, \dots, N$, one can construct key, query and value matrices W_K, W_Q, W_V as well as the projection matrix P such that a Transformer step on every token e_j is identical to the gradient-induced dynamics $e_j \leftarrow (x_j, y_j) + (0, -\Delta W x_j) = (x_i, y_i) + PVK^T q_j$ such that $e_j = (x_j, y_j - \Delta y_j)$. For the test data token (x_{N+1}, y_{N+1}) the dynamics are identical.*

We provide the weight matrices in block form: $W_K = W_Q = \begin{pmatrix} I_x & 0 \\ 0 & 0 \end{pmatrix}$ with I_x and I_y the identity matrices of size N_x and N_y respectively. Furthermore, we set $W_V = \begin{pmatrix} 0 & 0 \\ W_0 & -I_y \end{pmatrix}$ with the weight matrix $W_0 \in \mathbb{R}^{N_y \times N_x}$ of the linear model we wish to train and $P = \frac{\eta}{N} I$ with identity matrix of size $N_x + N_y$. With this simple construction we obtain the following dynamics

$$\begin{pmatrix} x_j \\ y_j \end{pmatrix} \leftarrow \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \frac{\eta}{N} I \sum_{i=1}^N \left(\begin{pmatrix} 0 & 0 \\ W_0 & -I_y \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \right) \otimes \left(\begin{pmatrix} I_x & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \right) \begin{pmatrix} I_x & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_j \\ y_j \end{pmatrix}$$
$$= \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \frac{\eta}{N} I \sum_{i=1}^N \begin{pmatrix} 0 \\ W_0 x_i - y_i \end{pmatrix} \otimes \begin{pmatrix} x_i \\ 0 \end{pmatrix} \begin{pmatrix} x_j \\ 0 \end{pmatrix} = \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \begin{pmatrix} 0 \\ -\Delta W x_j \end{pmatrix}. \tag{8}$$

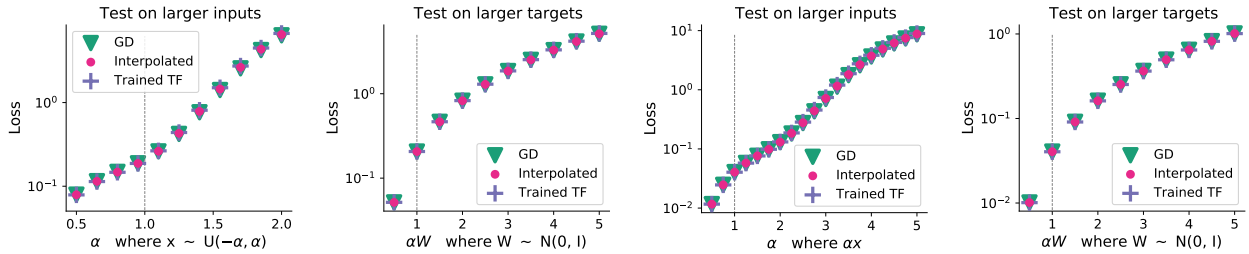
for every token $e_j = (x_j, y_j)$ including the query token $e_{N+1} = e_{\text{test}} = (x_{\text{test}}, -W_0 x_{\text{test}})$ which will give us the desired result.

A.2. Comparing the out-of-distribution behavior of trained Transformers and GD

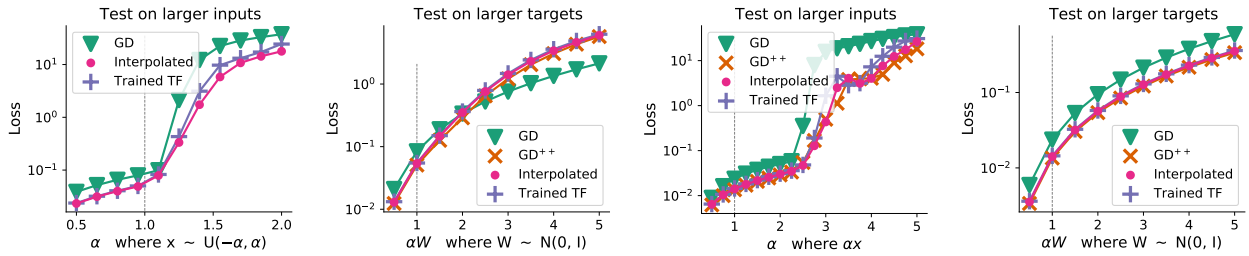
We provide more experimental results when comparing GD with tuned learning rate η and data transformation scalar γ and the trained Transformer on other data distributions than provided during training, see Figure 6. We do so by changing the in-context data distribution and measure the loss of both methods averaged over 10.000 tasks when either changing α that 1) affects the input data range $x \sim U(-\alpha, \alpha)^{N_x}$ or 2) the teacher by αW with $W \sim \mathcal{N}(0, I)$. This setups leads to results shown in the main text, in the first two columns of Figure 6 and in the corresponding plots of Figure 7. Although the match for deeper architectures starts to become worse, overall the trained Transformers behaves remarkably similar to GD and GD^{++} for layer depth greater than 1.

Furthermore, we try GD and the trained Transformer on input distributions that it never has seen during training. Here, we chose by chance of $1/3$ either a normal, exponential or Laplace distribution (with JAX default parameters) and depict the average loss value over 10.000 tasks where the α value now simply scales the input values that are sampled from one of the distributions αx. The teacher scaling is identical to the one described above. See for results the two right columns of Figure 6, where we see almost identical behavior for recurrent architectures with less good match for deeper non-recurrent

(a) Comparing one step of gradient descent with trained one layer Transformers on OOD data.



(b) Comparing two steps of gradient descent with trained *recurrent* two layer Transformers on OOD data.



(c) Comparing five steps of gradient descent with trained five layer Transformers on OOD data.

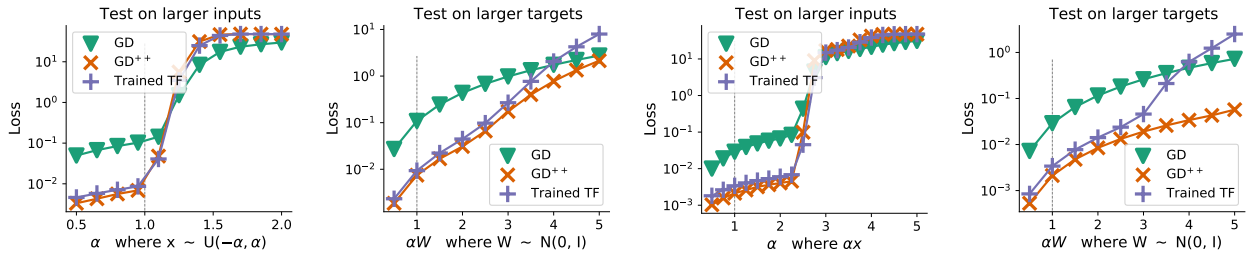


Figure 6. Left & center left column: Comparing Transformers, GD and their weight interpolation on rescaled training distributions. In all setups, the trained Transformer behaves remarkably similar to GD or GD^{++}. *Right & center right*: Comparing Transformers, GD and their weight interpolation on data distributions never seen during training. Again, in all setups, the trained Transformer behaves remarkably similar to GD or GD^{++} with less good match for deep non-recurrent Transformers far away from training regimes.

architectures far away from the training range of $\alpha = 1$. Note that for deeper Transformers ($K > 2$) the corresponding GD and GD^{++} version, see for more experimental details Appendix section A.12, we include a harsh clipping of the token values after every step of transformation between $[-10, 10]$ (for the trained TF and GD) to improve training stability. Therefore, the loss increase is restricted to a certain value and plateaus.

A.3. Linear mode connectivity between the weight construction of Prop 1 and trained Transformers

In order to interpolate between the construction θ_{GD} and the trained weights of the Transformer θ, we need to correct for some scaling ambiguity. For clarification, we restate here the linear self-attention operation for a single head

$$e_j \leftarrow e_j + PW_V \sum_i e_i \otimes e_i W_K^T W_Q e_j \tag{9}$$

$$= e_j + W_{PV} \sum_i e_i \otimes e_i W_{KQ} e_j \tag{10}$$

Now, to match the weight construction of Prop. 1 we have the aim for the matrix product W_{KQ} to match an identify matrix (except for the last diagonal entry) after re-scaling. Therefore we compute the mean of the diagonal of the matrix product of the trained Transformer weights W_{KQ} which we denote by β. After resealing both operations i.e. $W_{KQ} \leftarrow W_{KQ}/\beta$ and $W_{PV} \leftarrow W_{PV}\beta$ we interpolate linearly between the matrix products of GD as well as these rescaled trained matrix products i.e. $W_{I,KQ} = (W_{GD,KQ} + W_{TF,KQ})/2$ as well as $W_{I,PV} = (W_{GD,PV} + W_{TF,PV})/2$. We use these parameters to obtain results throughout the paper denote with *Interpolated*. We do so for GD as well as GD^{++} when comparing to

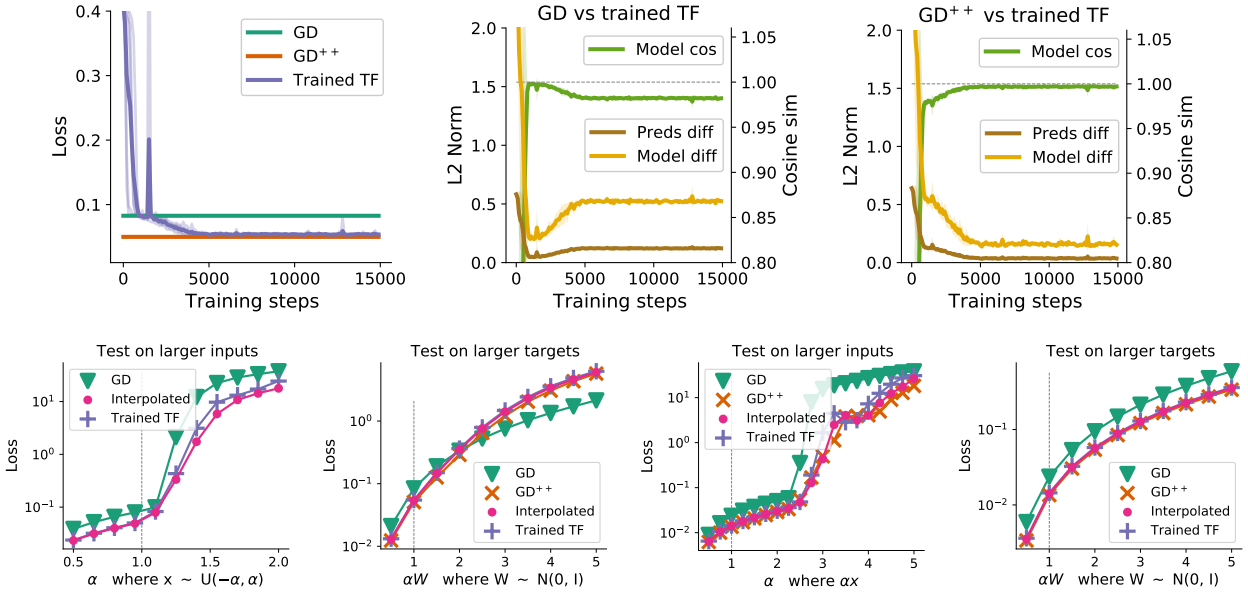


Figure 7. **Comparing ten steps of gradient descent with trained *recurrent* ten-layer Transformers.** Results comparable to recurrent Transformer with two layers, see Figure 3, but now with 10 repeated layers. We again observe for deeper recurrent linear self-attention only Transformers that overall GD^{++} and the trained Transformer align very well with one another and are again interpolatable leading to very similar behavior insight as well as outside training situations. Note the inferior performance to the non-recurrent five-layer Transformer which highlights the importance on specific learning rate as well γ parameter per layer/step.

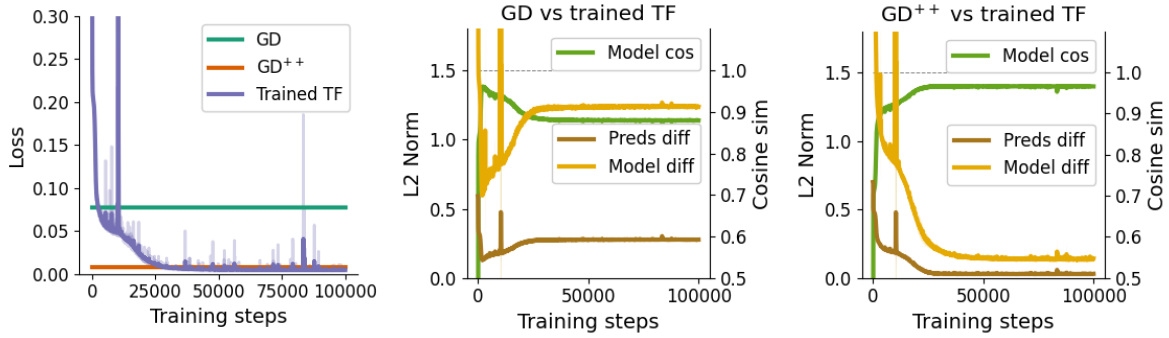


Figure 8. **Comparing twelve steps of GD^{++} with a trained twelve-layer Transformers with MLPs and 4 headed linear self-attention layer.** Results comparable to the deep recurrent Transformer, see Figure 7, but now with 12 independent Transformer blocks including MLPs and 4-head linear self-attention. We omit LayerNorm. We again observe a close resemblance of the trained Transformers and GD^{++}. We hypotheses that even when equipped with multiple heads and MLPs, Transformers approximate GD^{++}.

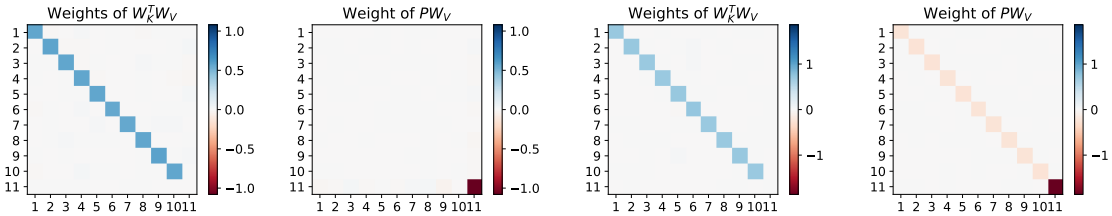


Figure 9. **Visualizing the weight matrices of trained Transformers**. *Left & outer left:* Weight matrix products of a trained single linear self-attention layer. We see (after scalar correction) a perfect resemblance of our construction. *Right & outer right:* Weight matrix products of a trained 3-layer recurrent linear self-attention Transformer. Again, we see (after scalar correction) a perfect resemblance of our construction and an additional curvature correction i.e. diagonal values in PW_V of the same magnitude except the last entry that functions as the learning rate.

recurrent Transformers. Note that for non-recurrent Transformers, we face more ambiguity that we have to correct for since e.g. scalings influence each other across layer. We also see this in practice and are not able (only for some seeds) to interpolate between weights with our simple correction from above. We leave the search for more elaborate corrections for future work.

A.4. Visualizing the trained Transformer weights

The simplicity of our construction enables us to visually compare trained Transformers and the construction put forward in Proposition A.1 in weight space. As discussed in the previous section A.3 there is redundancy in the way the trained Transformer can construct the matrix products leading to the weights corresponding to gradient descent. We therefore visualize $W_{KQ} = W_K^T W_Q$ as well as $W_{PV} = P_K W_V$ in Figure 9.

A.5. Proof and discussion of Proposition 3

We state here again Proposition 3, provide the necessary construction and a short discussion.

Proposition 3. *Given a 1-head linear- or softmax attention layer and the token construction $e_{2j} = (x_j), e_{2j+1} = (0, y_j)$ with a zero vector 0 of dim $N_x - N_y$ and concatenated positional encodings, one can construct key, query and value matrix W_K, W_Q, W_V as well as the projection matrix P such that all tokens e_j are transformed into tokens equivalent to the ones required in proposition 1.*

To get a simple and clean construction, we choose wlog $x_j \in \mathbb{R}^{2N+1}$ and $(0, y_j) \in \mathbb{R}^{2N+1}$ as well as model the positional encodings as unit vectors $p_j \in \mathbb{R}^{2N+1}$ and concatenate them to the tokens i.e. $e_j = (x_{j/2}, p_j)$. We wish for a construction that realizes

$$e_j \leftarrow \begin{pmatrix} x_{j/2} \\ p_j \end{pmatrix} + PVK^T W_Q \begin{pmatrix} x_{j/2} \\ p_j \end{pmatrix} \tag{11}$$

$$= \begin{pmatrix} x_{j/2} \\ p_j \end{pmatrix} + \begin{pmatrix} 0 \\ y_{j/2+1} - p_j \end{pmatrix}. \tag{12}$$

This means that a token replaces its own positional encoding by coping the target data of the next token to itself leading to $e_j = (x_{j/2}, 0, y_{j/2+1})$, with slight abusive of notation. This can simply be realized by (for example) setting $P = I$, $W_V = \begin{pmatrix} 0 & 0 \\ I_x & -I_{x,off} \end{pmatrix}, W_K = \begin{pmatrix} 0 & 0 \\ 0 & I_x \end{pmatrix}$ and $W_Q = \begin{pmatrix} 0 & 0 \\ 0 & I_{x,off}^T \end{pmatrix}$ with $I_{x,off}$ the lower diagonal identity matrix fo size N_x. Note that then simply $K^T W_Q e_j = p_{j+1}$ i.e. it chooses the $j+1$ element of V which stays p_{j+1} if we apply the softmax operation on $K^T q_j$. Since the $j+1$ entry of V is $(0, y_{j/2+1} - p_j)$ we obtain the desired result.

For the (toy-)regression problems considered in this manuscript, the provided result would give $N/2$ tokens for which we also copy (parts) of x_j underneath y_j. This is desired for modalities such as language where every two tokens could be considered an in-and output pair for the implicit autoregressive inner-loop loss. These tokens do not have be necessarily next to each other, see for this behavior experimental findings presented in (Olsson et al., 2022). For the experiments conducted here, one solution is to zero out these tokens which could be constructed by a two-head self-attention layer that given uneven j simply subtracts itself resulting in a zero token. For all even tokens, we use the construction from above which effectively coincides with the token construction required in Proposition 1.

Rolling out experiment with different dampening strength

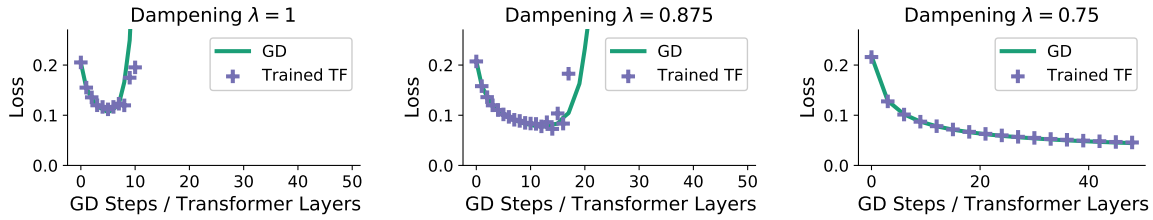


Figure 10. **Roll-out experiments: applying a trained single linear self-attention layer multiple times.** We observe that different dampening strengths affect the generalization of both methods with slightly better robustness for GD which matching performance for 50 steps when $\lambda = 0.75$.

A.6. Dampening the self-attention layer

As an additional out-of-distribution experiment, we test the behavior when repeating a single LSA-layer trained to lower our objective, see equation 5, with the aim to repeat the learned learning/update rule. Note that GD as well as the self-attention layer were optimized to be optimal for one step. For GD we line search the otpimal learning rate η on 10.000 task. Interestingly, for both methods we observe quick divergence when applied multiple times, see left plot of Figure 10. Nevertheless, both of our update functions are described by a linear self-attention layer for which we can control the norm, post training, by a simple scale which we denote as λ. This results in the new update $y_{\text{test}} + \lambda \Delta W x_{\text{test}}$ for GD and $y_{\text{test}} + \lambda P V K^T W_Q x_{\text{test}}$ for the trained self-attention layer which effectively re-tunes the learning rate for GD and the trained self-attention layer. Intriguingly, both methods do generalize similarly well (or poorly) on this out-of-distribution experiment when changing λ, see again Figure 10. We show in Figure 1 the behavior for $\lambda = 0.75$ for which we see both methods steadily decreasing the loss within 50 steps.

A.7. Sine wave regression

For the sine wave regression tasks, we follow (Finn et al., 2017) and other meta-learning literature and sample for each task an amplitude $a \sim U(0.1, 5)$ and a phase $\rho \sim U(0, \pi)$. Each tasks consist of $N = 10$ data points where inputs are sampled $x \sim U(-5, 5)$ and targets computed by $y = a \sin(\rho + x)$. We choose here for the first time, for GD as well as for the Transformer, an input embedding emb that maps tokens $e_i = (x_i, y_i)$ into a 40 dimensional space $\text{emb}(e_i) = W_{\text{emb}} e_i$ through an affine projection without bias. We skip the first self-attention layer but, as usually done in Transformers, then transform the embedded tokens through an MLP m with a single hidden layer, widening factor of 4 (160 hidden neuros) and GELU nonlinearity (Hendrycks & Gimpel, 2016) i.e. $e_j \leftarrow m(\text{emb}(e_j)) + \text{emb}(e_j)$.

We interpret the last entry of the transformed tokens as the (transformed) targets and the rest as a higher-dimensional input data representation on which we train a model with a single gradient descent step. We compare the obtained meta-learned GD solution with training a Transformer on the same token embeddings but instead learn a self-attention layer. Note that the embeddings of the tokens, including the transformation through the MLP, are not dependent on an interplay between the tokens. Furthermore, the initial transformation is dependent on $e_i = (x_i, y_i)$, i.e., input as well as on the target data except for the query token for which $y_{\text{test}} = 0$. This means that this construction is, except for the additional dependency on targets, close to a large corpus of meta-learning literature that aims to find a deep representation optimized for (fast) fine tuning and few-shot learning. In order to compare the meta-training of the MLP and the Transformer, we choose the same seed to initialize the network weights for the MLPs and the input embedding trained by meta-learning i.e. backprop through training or the Transformer. This leads to the plots and almost identical learned initial function and updated functions shown in Figure 4.

A.8. Proposition 2 and connections between gradient descent, kernelized regression and kernel smoothing

Let's consider the data transformation induced by an MLP $\tilde{m}(x)$ and a residual connection commonly used in Transformer blocks i.e. $e_j \leftarrow e_j + \tilde{m}(e_j) = (x_j, y_j) + (\tilde{m}(x_j), 0) = (m(x_j), y_j)$ with $m(x_j) = x_j + \tilde{m}(x_j)$ and $\tilde{m}$ not changing the targets y. When simply applying Proposition 1, it is easy to see that given this new token construction, a linear self-attention layer can induce the token dynamics $e_j \leftarrow (m(x_j), y_j) + (0, -\Delta W m(x_j))$ with $\Delta W = -\eta \nabla L(W)$ given the loss function $L(W) = \frac{1}{2N} \sum_{i=1}^{N} ||W m(x_i) - y_i||^2$.

Interestingly, for the test token $e_{\text{test}} = (x_{\text{test}}, 0)$ this induces, after a multiplication with -1, an initial prediction after a single Transformer block given by

$$\hat{y} = \Delta W m(x_{\text{test}}) = -\eta \nabla_W L(0) m(x_{\text{test}}) = \sum_{i=1}^{N} y_i m(x_i)^T m(x_{\text{test}}) = \sum_{i=1}^{N} y_i k(x_i, x_{\text{test}}) \tag{13}$$

with $m(x_i)^T m(x_{\text{test}}) = k(x_i, x_{\text{test}}) \in \mathbb{R}$ interpreted as a kernel function. Concluding, we see that the combination of MLPs and a *single* self-attention layer can lead to dynamics induced when descending a kernelized regression (squared error) loss with a *single* step of gradient-descent.

Interestingly, when choosing $W_0 = 0$, we furthermore see that a single self-attention layer or Transformer block can be regarded as doing nonparametric kernel smoothing $\hat{y} = \sum_{i=1}^{N} y_i k(x_i, x_{\text{test}})$ based on the data given in-context (Nadaraya, 1964; Watson, 1964). Note that we made a particular choice of kernel function here and that this view still holds when $m(x_j) = \mathbb{1}$ i.e. consider Transformers without MLPs or leverage the well-known view of softmax self-attention layer as a kernel function used to measure similarity between tokens (e.g. Choromanski et al., 2021; Zhang et al., 2021). Thus, implementing one step of gradient descent through a self-attention layer (w/wo softmax nonlinearity) is equivalent to performing kernel smoothing estimation. We however argue that this nonparametric kernel smoothing view of in-context learning is limited, and arises from looking only at a *single* self-attention layer. When considering deeper Transformer architectures, we see that multiple Transformer blocks can iteratively transform the targets based on multiple steps of gradient descent leading to minimization of a kernelized squared error loss $L(W)$. One way to obtain a suitable construction is by neglecting MLPs everywhere except in the first Transformer block. We leave the study of the exact mechanics, especially how the Transformer makes use of possibility transforming the targets through the MLPs, and the possibility of iteratively changing the kernel function throughout depth for future study.

A.9. Linear vs. softmax self-attention as well LayerNorm Transformers

Although linear Transformers and their variants have been shown to be competitive with their softmax counterpart (Irie et al., 2021), the removal of this nonlinearity is still a major departure from classic Transformers and more importantly from the Transformers used in related studies analyzing in-context learning. In this section we investigate whether and when gradient-based learning emerges in trained softmax self-attention layers, and we provide an analytical argument to back our findings.

First, we show, see Figure 12, that a single layer of softmax self-attention is not able to match GD performance. We tuned the learning rate as well as the weight initialization but found no significant difference over the hyperparameters we used througout this study. In general, we hypothesize that GD is an optimal update given the limited capacity of a single layer of (single-head) self-attention. We therefore argue that the softmax induces (at best) a linear offset of the matrix product of training data and query vector

$$\text{softmax}(K^T q_j) = (e^{k_1^T q_j}, \dots, e^{k_N^T q_j})^T / (\sum_i e^{k_i^T q_j}) \tag{14}$$

$$= (e^{x_1^T W_{KQ} x_j}, \dots, e^{x_N^T W_{KQ} x_j})^T / (\sum_i e^{x_i^T W_{KQ} x_j}) \tag{15}$$

$$\approx (1 + x_1^T W_{KQ} x_j, \dots, 1 + x_N^T W_{KQ} x_j)^T / (\sum_i 1 + x_i^T W_{KQ} x_j) \tag{16}$$

$$\propto K^T q_j + \epsilon \tag{17}$$

proportional to a factor dependent on all $\{x_{\tau,i}\}_{i=1}^{N+1}$. We speculate that the dependency on the specific task τ, for large N_x vanishes or that the x-dependent value matrix could introduce a correcting effect. In this case the softmax operation introduces an additive error w.r.t. to the optimal GD update. To overcome this disadvantageous offset, the Transformer can (approximately) introduce a correction with a second self-attention head by a simple subtraction i.e.

$$P_1 V_1 \text{softmax}(K_1^T W_Q x_j) + P_2 V_2 \text{softmax}(K_2^T W_Q x_j) \tag{18}$$

$$\approx PV((1 + x_1^T W_{1,KQ} x_j, \dots, 1 + x_N^T W_{1,KQ} x_j) - (1 + x_1^T W_{2,KQ} x_j, \dots, 1 + x_N^T W_{2,KQ} x_j)) \tag{19}$$

$$= PV(x_1^T (W_{1,KQ} - W_{2,KQ}) x_j, \dots, x_N^T (W_{1,KQ} - W_{2,KQ}) x_j) \tag{20}$$

$$\propto PVK^T q_j. \tag{21}$$

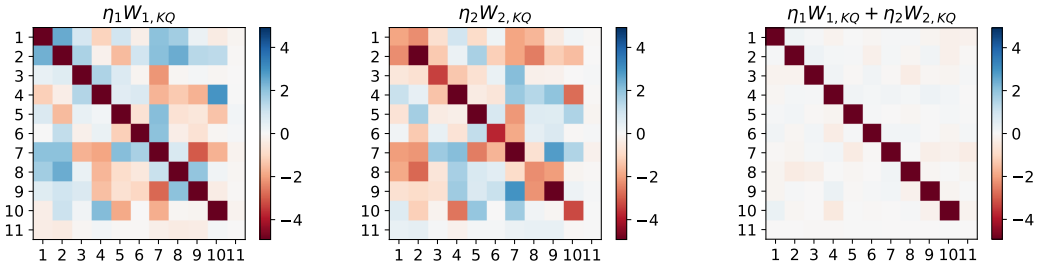


Figure 11. **Visualizing the correction to the softmax operation when training Transformers on regression tasks.** The left and center plot show the matrix product $W_{KQ} = W_K^T W_Q$ including its scaling by η induced through PW_V of the two heads of the trained softmax self-attention layer. We observe that both of the matrices are approximate diagonal almost perfect sign reversed values on the off-diagonal terms. After adding the matrices (right plot), we observe a diagonal matrix and therefore to much improved approximation of our construction and therefore gradient descent dynamics.

Here we assume that PV 1) subsumes the dividing factor of the softmax and that 2) is the same (up to scaling) for each head. Note that if $(W_{1,KQ} - W_{2,KQ})$ is diagonal, and P and V chosen as in the Proposition of Appendix A.1, we recover our gradient descent construction.

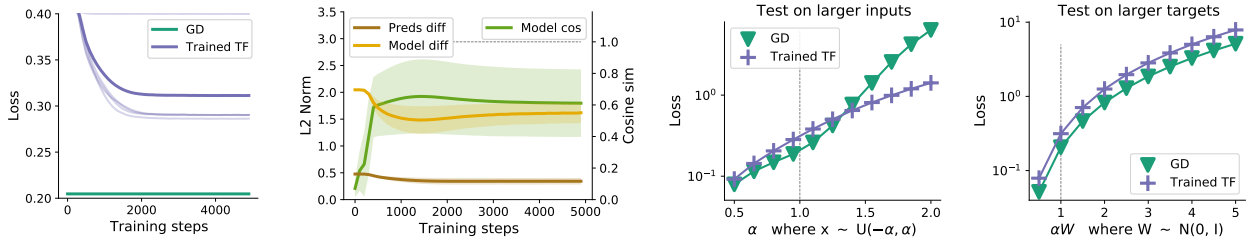
We base this derivation on empirical findings, see Figure 12, that, first of all, show the softmax self-attention performance increases drastically when using two heads instead of one. Nevertheless, the self-attention layer has difficulties to match the loss values of a model trained with GD. Furthermore, this architecture change leads to a very much improved alignment of the trained model and GD. Second, we can observe that when training a two-headed softmax self-attention layer on regression tasks the correction proposed above is actually observed in weight space, see Figure 11. Here, we visualize the matrix product within the softmax operation $W_{h,KQ}$ per head which we scale with the last diagonal entry of $P_h W_{h,V}$ which we denote by $\eta_h = P_h W_{h,V}(-1,-1)$. Intriguingly, this results in an almost perfect cancellation (right plot) of the off-diagonal terms and therefore in sum to an improved approximation of our construction, see the derivation above.

We would like to reiterate that the stronger inductive bias for copying data of the softmax layer remains, and is not invalidated by the analysis above. Therefore, even for our shallow and simple constructions they indeed fulfill an important role in support for our hypotheses: The ability to merge or copy input and target data into single tokens allowing for their dot product computation necessary for the construction in Proposition 1, see Section 4 in the main text.

We end this section by analysing Transformers equipped with LayerNorm which we apply as usually done before the self-attention layer: Overall, we observe qualitatively similar results to Transformers with softmax self-attention layer i.e. a decrease in performance compared to GD accompanied with a decrease in alignment between models generated by the Transformer and models trained with GD, see Figure 14. Here, we test again a single linear self-attention layer succeeding LayerNorm as well as two layers where we skip the first LayerNorm and only include a LayerNorm between the two. Including more heads does not help substantially. We again assume the optimality of GD and argue that information of targets and inputs present in the tokens is lost by averaging when applying LayerNorm. This naturally leads to decreasing performance compared to GD, see first row of Figure 14. Although the alignment to GD and GD^{++}, especially for two layers, is high, we overall see inferior performance to one or two steps of GD or two steps of GD^{++}. Nevertheless, we speculate that LayerNorm might not only stabilize Transformer training but could also act as some form of data normalization procedure that implicitly enables better generalization for larger inputs as well as targets provided in-context, see OOD experiments in Figure 14.

Overall we conclude that common architecture choices like softmax and LayerNorm seem supoptimal for the constructed in-context learning settings when comparing to GD or linear self-attention. Nevertheless, we speculate that the potentially small performance drops of in-context learning are negligible when turning to deep and wide Transformers for which these architecture choices have empirically proven to be superior.

(a) Comparing one step of GD with a trained *softmax one*-headed self-attention layer.



(b) Comparing one step of GD with a trained *softmax two*-headed self-attention layer.

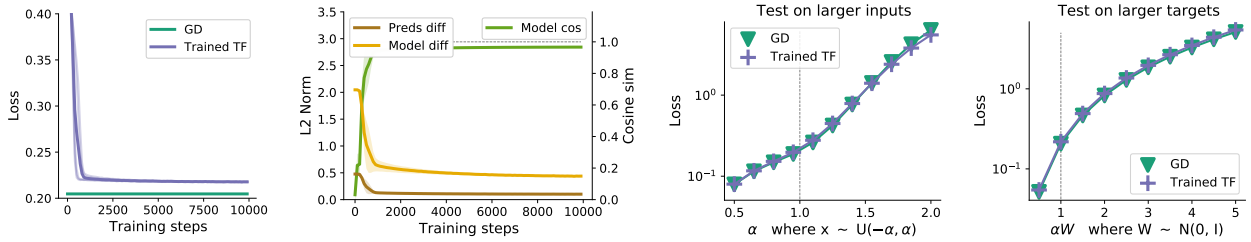


Figure 12. **Comparing trained two-headed and one-headed single-layer *softmax* self-attention with 1 step of gradient descent on linear regression tasks.** *Left column:* Softmax self-attention is not able to match gradient descent performance with hand-tuned learning rate, but adding a second attention head significantly reduces the gap, as expected by our analytical argument. *Center left*: The alignment suffers significantly for single-head softmax SA. We observe good but not as precise alignment when compared to linear Transformers for the two-headed softmax SA layer. *Center right & right:* The two-headed self-attention compared to the single-head layer shows similar robust out-of-distribution behavior compared to gradient descent.

A.10. Details of curvature correction

We give here a precise construction showing how to implement in a single head, a step of GD and the discussed data transformation, resulting in GD^{++}. Recall again the linear self-attention operation with a single head

$$e_j \leftarrow e_j + PW_V \sum_i e_i \otimes e_i W_K^T. \tag{22}$$

We provide again the weight matrices in block form of the construction of Prop. 1 but now enabling additionally our described data transformation: $W_K = W_Q = \begin{pmatrix} I_x & 0 \\ 0 & 0 \end{pmatrix}$ with I_x the identity matrix of size N_x, I_y od size N_y resp. Furthermore, we set $W_V = \begin{pmatrix} I_x & 0 \\ W & -I_y \end{pmatrix}$ with the weight matrix $W \in \mathbb{R}^{N_y \times N_x}$ of the linear model we wish to train and $P = \begin{pmatrix} -\gamma I_x & 0 \\ 0 & \frac{\eta}{N} \end{pmatrix}$. This leads to the following update

$$\begin{aligned} \begin{pmatrix} x_j \\ y_j \end{pmatrix} &\leftarrow \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \begin{pmatrix} -\gamma I_x & 0 \\ 0 & \frac{\eta}{N} \end{pmatrix} \sum_{i=1}^N \left(\begin{pmatrix} I_x & 0 \\ W & -I_y \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \right) \otimes \left(\begin{pmatrix} I_x & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \right) \begin{pmatrix} I_x & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_j \\ y_j \end{pmatrix} \\ &= \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \begin{pmatrix} -\gamma I_x & 0 \\ 0 & \frac{\eta}{N} \end{pmatrix} \sum_{i=1}^N \begin{pmatrix} x_i \\ W x_i - y_i \end{pmatrix} \otimes \begin{pmatrix} x_i \\ 0 \end{pmatrix} \begin{pmatrix} x_j \\ 0 \end{pmatrix} = \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \begin{pmatrix} -\gamma X X^T x_j \\ -\Delta W x_j \end{pmatrix}. \end{aligned} \tag{23}$$

for every token $e_j = (x_j, y_j)$ including the query token $e_{N+1} = e_{\text{test}} = (x_{\text{test}}, 0)$ which will give us the desired result.

Why does GD^{++} perform better? We give here one possible explanation of the superior performance of GD^{++} compared to GD. Note that there is a close resemblance of the GD transformation and a heavily truncated Neuman series approximation of the inverse XX^T. We provide here a more heuristic explanation for the observed acceleration.

Given $\gamma \in \mathbb{R}$, GD^{++} transforms every input according to $x_i \leftarrow x_i - \gamma X X^T x_i = (I - \gamma X X^T) x_i$. We can therefore look at the change of squared regression loss $L(W) = \frac{1}{2} \sum_{i=0}^N (W x_i - y_i)^2$ induced by this transformation i.e. $L^{++}(W) =$

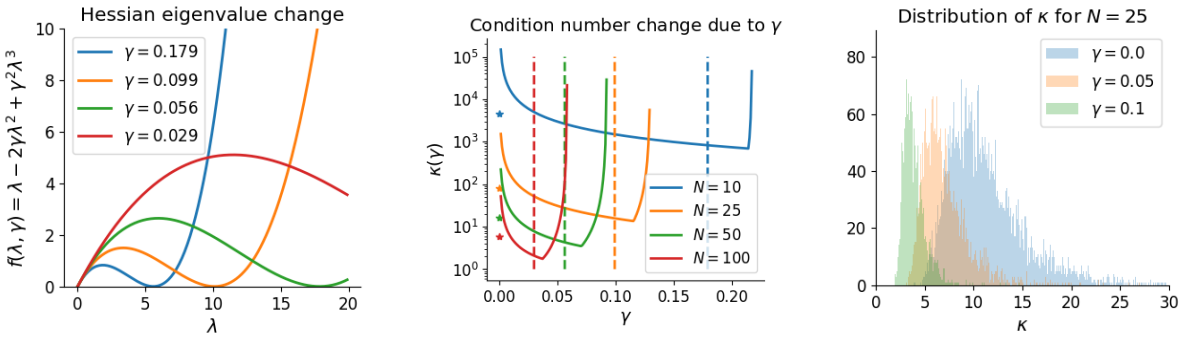


Figure 13. **GD^{++} analyses**. *Left:* We visualize the change of the eigenspectrum induced by the input data transformation of GD^{++} for different γ observed in practice. *Center:* Given we know the maximum and minimum of eigenvalues λ_1, λ_n of the loss Hessian XX^T with $X = (x_0, \ldots, x_N)$ for different N, we compare the original condition number (depicted by *'s at $\gamma = 0$) and the condition number (in log scale) of the GD^{++} altered loss Hessian when varying γ. We plot in dotted lines the γ values that we observe in practice which are close the optimal ones i.e. the local minimum derived through our analysis. *Right:* For$N = 25$, we plot for different γ values the distribution of condition numbers $\kappa = \lambda_1/\lambda_n$ for 10000 tasks and observe favorable κ values close to 1 when approaching the $\gamma = 0.099$ value was found in practice. The κ values quickly explode for $\gamma > 0.1$.

$\frac{1}{2}\sum_{i=0}^{N}(W(I-\gamma XX^T)x_i - y_i)^2 = \frac{1}{2}(W(I-\gamma XX^T)X - Y)^2$ which in turn leads to a change of the loss Hessian from $\nabla^2 L = XX^T$ to $\nabla^2 L^{++} = (I-\gamma XX^T)X((I-\gamma XX^T)X)^T$.

Given the original Hessian $H = XX^T = U\Sigma U^T$ with it's set of sorted eigenvalues $\{\lambda_1, \ldots, \lambda_n\}$ and $\lambda_i \geq 0$ on the diagonal matrix Σ we can express the new Hessian through U, Σ i.e. $H^{++} = (I-\gamma XX^T)X((I-\gamma XX^T)X)^T = (I-\gamma U\Sigma U^T)U\Sigma U^T(I-\gamma U\Sigma U^T)^T$.

We can simplify H^{++} further as

$$H^{++} = (I-\gamma U\Sigma U^T)U\Sigma U^T(I-\gamma U\Sigma U^T)^T = U(\Sigma-\gamma\Sigma^2)U^T U(I-\gamma\Sigma)U^T \tag{24}$$

$$= U(\Sigma - 2\gamma\Sigma^2 + \gamma^2\Sigma^3)U^T \tag{25}$$

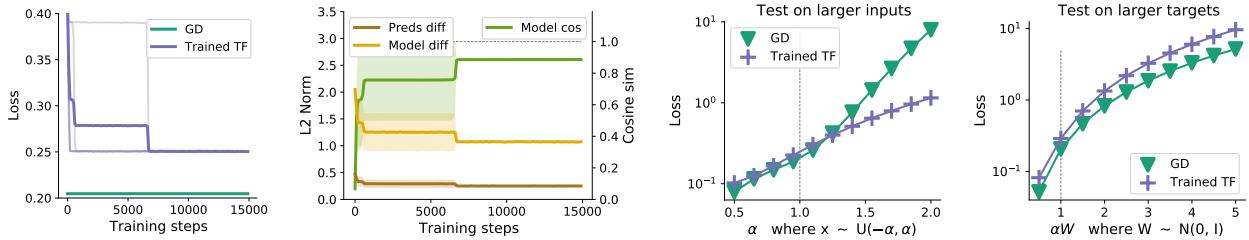
Given the eigenspectrum $\{\lambda_1, \ldots, \lambda_n\}$ of H, we obtain an (unsorted) eigenspecturm for H^{++} with $\{\lambda_1 - 2\gamma\lambda_1^2 + \gamma^2\lambda_1^3, \ldots, \lambda_n - 2\gamma\lambda_n^2 + \gamma^2\lambda_n^3\}$ which we visualize in Figure 13 for different γ observed in practice. We hypotheses that the Transformer chooses γ in a way that on average, across the distribution of tasks, the data transformation (iteratively) decreases the condition number λ_1/λ_n leading to accelerated learning. This could be achieved, for example, by keeping the smallest eigenvalue $\lambda_n \approx \lambda_n^{++}$ fixed and choosing γ such that the largest eigenvalue of the transformed data λ_1^{++} is reduced, while the original λ_1 stays within $[\lambda_1^{++}, \lambda_n^{++}]$.

To support our hypotheses empirically, we computed the minimum and maximum eigenvalues of XX^T across 10000 tasks while changing the number of datapoints $N \in [10, 25, 50, 100]$ i.e. $X = (x_0, \ldots, x_N)$ leading to better conditioned loss Hessians i.e. $[1\text{e}{-10}, 0.097, 0.666, 2.870]$ and $[4.6, 7.712, 10.845, 17.196]$ as the minimum and maximum eigenvalues of XX^T across all tasks where we cut the smallest eigenvalue for $N = 10$ at $1\text{e}{-10}$. Furthermore, we extract the γ values from the weights of optimized recurrent 2-layer Transformers trained on different task distributions and obtain γ values of $[0.179, 0.099, 0.056, 0.029]$, see again Figure 13. Note that the observed eigenvalues stay within $[0, 1/\gamma]$ i.e. the two roots of $f(\lambda, \gamma) = \lambda - 2\gamma\lambda^2 + \gamma^2\lambda^3$.

Given the derived function of eigenvalue change $f(\lambda, \gamma)$, we compute the condition number of H^{++} by dividing the novel maximum eigenvalues $\lambda_1^{++} = f(1/(3\gamma), \gamma)$ where $\lambda = 1/(3\gamma)$ as the local maximum of $f(\lambda, \gamma)$, for fixed γ, and the novel minimum eigenvalue $\lambda_n^{++} = \min(f(\lambda_1, \gamma), f(\lambda_n, \gamma))$. Note that with too small γ, we move the original λ_n closer to the root of $f(\lambda, \gamma)$ i.e. $\lambda = 1/\gamma$ and therefore can change the smallest eigenvalue.

Given the task distribution and its corresponding eigenvalue distribution, we see that choosing γ reduces the new condition number $\kappa^{++} = \lambda_1^{++}/\lambda_n^{++}$ which leads to better conditioned learning, see center plot of Figure 13. Note that the optimal γ based on our derivation above is based on the maximum and minimum eigenvalue across all tasks and does not take the change of the eigenvalue distribution into account. We argue therefore that the simplicity of the arguments above does not capture the task statistics and distribution shifts entirely and therefore obtains a slightly larger γ as an optimal value.

(a) Comparing one step of GD with a single-layer LSA Transformer with LayerNorm.



(b) Comparing two steps of GD with a two-layer LSA Transformer with LayerNorm.

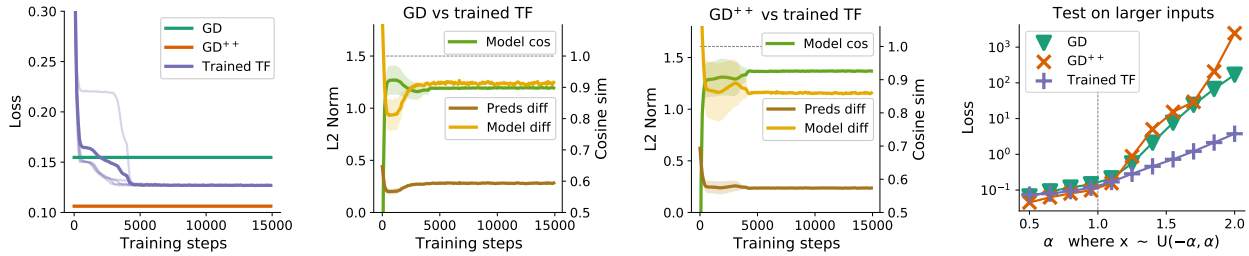


Figure 14. **Comparing trained 1-layer and 2-layer Transformers with *LayerNorm* and 1 step or 2 steps of gradient descent resp.** *Left column:* The Transformers is not able to match the gradient descent performance with hand-tuned learning rate. *Alignment plots*: The alignment suffers significantly when comparing to linear self-attention layers although still reasonable alignment is obtained which decreases slightly when comparing to GD^{++} for the two-layer Transformer.*Center right & right:* The LayerNorm Transformer outperforms when GD when providing training input data that is significantly larger than the data provided during training.

We furthermore visualize the condition number change for $N = 25$ and 10000 tasks in the right plot of Figure 13 and observe the distribution moving to desirable κ values close to 1. For γ values larger than 0.1 the distribution quickly exhibits exploding condition numbers.

A.11. Phase transitions

We comment shorty on the curiously looking phase transitions of the training loss observed in many of our experiments, see Figure 2. Nevertheless, simply switching from a single-headed self-attention layer to a two-headed self-attention layer mitigates the random seed dependent training instabilities in our experiments presented in the main text, see left and center plot of Figure 15.

Furthermore, these transitions look reminiscent of the recently observed "grokking" behaviour (Power et al., 2022). Interestingly, when carefully tuning the learning rate and batchsize we can also make the Transformers trained in these linear regression tasks *grokk*. For this, we train a single Transformer block (self-attention layer and MLP) on a limited amount of data (8192 tasks), see right plot of Figure 15, and observe grokking like train and test loss phase transitions where test set first increases drastically before experiencing a sudden drop in loss almost matching the desired GD loss of 0.2. We leave a thorough investigation of these phenomena for future study.

A.12. Experimental details

We use for most experiments identical hyperparameters that were tuned by hand which we list here

- Optimizer: Adam (Kingma & Ba, 2014) with default parameters and learning rate of 0.001 for Transformer with depth $K < 3$ and 0.0005 otherwise. We use a batchsize of 2048 and applied gradient clipping to obtain gradients with global norm of 10. We used the Optax library.
- Haiku weight initialisation (fan-in) with truncated normal and std $0.002/K$ where K the number of layers.
- We did not use any regularisation and observed for deeper Transformers with $K > 2$ instabilities when reaching GD performance. We speculate that this occurs since the GD performance is, for the given training tasks, already close to divergence as seen when providing tasks with larger input ranges. Therefore, training Transformers also becomes

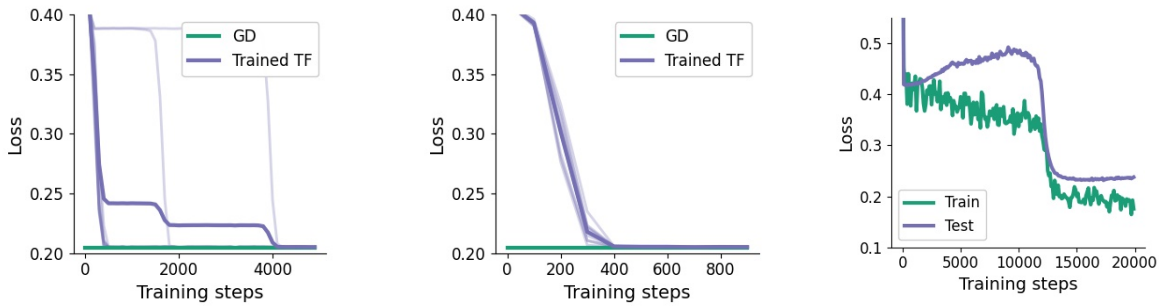


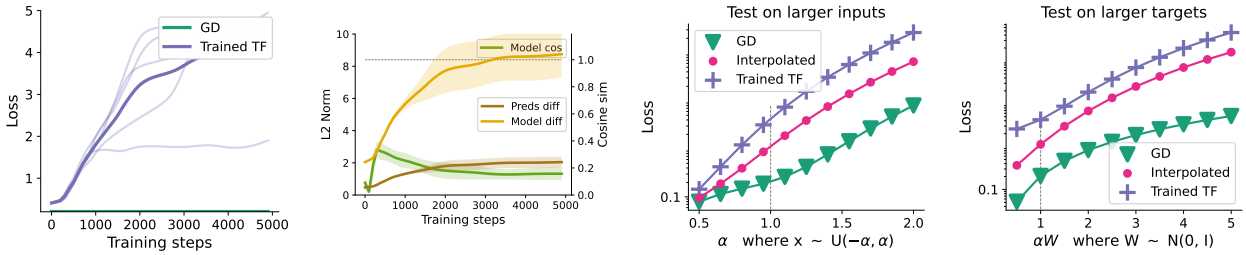
Figure 15. **Phase transitions during training**. *Left:* Loss based on 10 different random seeds when optimizing a single-headed self-attention layer. We observe for some seeds very long initial phases of virtually zero progress after which the loss drops suddenly to the desired GD loss. *Center:* The same experiment but optimizing a *two*-headed self-attention layer. We observe fast and robust convergence to the loss of GD. *Right:* Training a single Transformer block i.e. a self-attention layer with MLP and a reduced training set size of 8192 tasks. We observe grokking like train and test loss phase transitions where test set first increases drastically before experiencing a sudden drop in loss almost matching the desired GD loss of 0.2.

instable when we approach GD with an optimal learning rate. In order to stabilize training, we simply clipped the token values to be in the range of $[-10, 10]$.

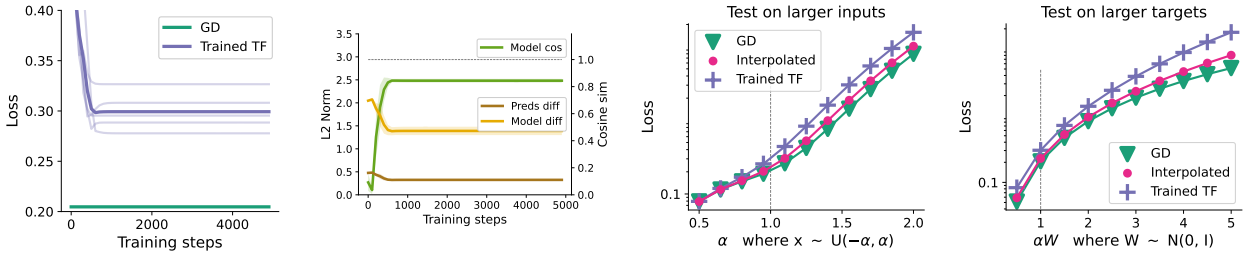
- When applicable we use standard positional encodings of size 20 which we concatenated to all tokens.
- For simplicity, and to follow the provided weight construction closely, we did use square key, value and query parameter matrix in all experiments.
- The training length varied throughout our experimental setups and can be read off our training plots in the article.
- When training meta-parameters for gradient descent i.e. η and γ we used an identical training setup but usually training required much less iterations.
- In all experiments we choose inital $W_0 = 0$ for gradient descent trained models.

Inspired by (Garg et al., 2022), we additionally provide results when training a single linear self-attention layer on a fixed number of training tasks. Therefore, we iterate over a single fixed batch of size B instead of drawing new batch of tasks at every iteration. Results can be found in Figure 16. Intriguingly, we find that (meta-)gradient descent finds Transformer weights that align remarkable well with the provided construction and therefore gradient descent even when provided with an arguably very small number of training tasks. We argue that this again highlights the strong inductive bias of the LSA-layer to match (approximately) gradient descent learning in its forward pass.

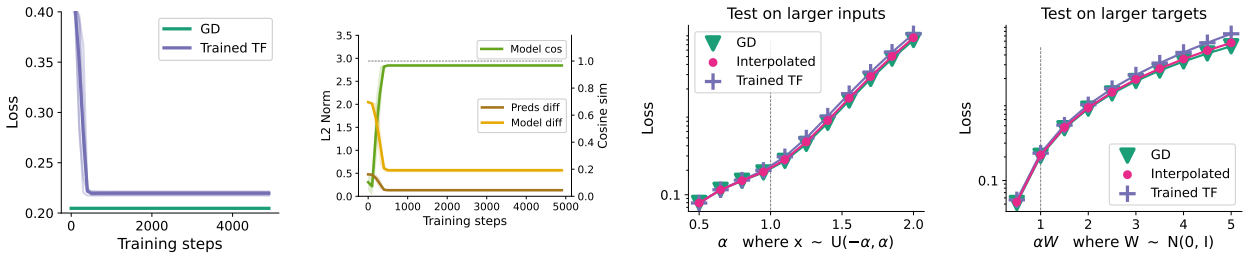
(a) Comparing 1 step of gradient descent with training a LSA-layer on 128 tasks.



(b) Comparing 1 step of gradient descent with training a LSA-layer on 512 tasks.



(c) Comparing 1 step of gradient descent with training a LSA-layer on 2048 tasks.



(d) Comparing 1 step of gradient descent training a LSA-layer on 8192 tasks.

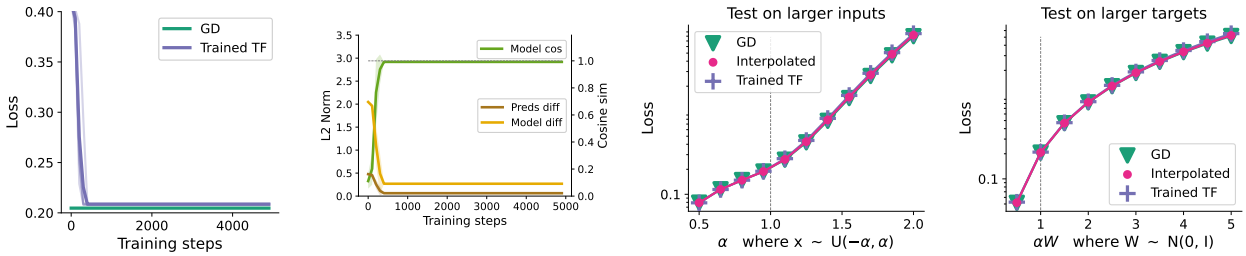


Figure 16. **Comparing trained Transformers with GD and their weight interpolation when training the Transformer on a fixed training set size** B. Across our alignment measures as well as our tests on out-of-training behaviour, trained Transformers fail to align with GD when provided with a very small amount of tasks. Nevertheless, we see already almost perfect alignment in our base setting $N = N_x = 10$ when provided with $B > 2048$ tasks. In all settings, we train the Transformer on (non-stochastic) gradient descent iterating over a single batch of tasks of size B equal to the number provided in the Figure titles (128, 512, 2048, 8192).