HiLD 2024: 2nd Workshop on High-dimensional Learning Dynamics

Looking at Deep Learning Phenomena Through a Telescoping Lens

Alan Jeffares* AJ659@CAM.AC.UK
University of Cambridge

Alicia Curth* AMC253@CAM.AC.UK
University of Cambridge

Mihaela van der Schaar MV472@CAM.AC.UK
University of Cambridge

Abstract

Deep learning sometimes appears to work in unexpected ways. In pursuit of deeper understanding of its surprising behaviors, we investigate the utility of a tractable and accurate model of a neural network consisting of a sequence of first-order approximations *telescoping* out into a single empirically operational tool for practical analysis. We illustrate how it can be applied to derive new empirical insights on a diverse range of prominent phenomena in the literature – including double descent, grokking and the challenges of applying deep learning on tabular data.

1. Introduction

Deep learning *works*, but it sometimes works in mysterious ways. The pursuit of a deeper understanding of both has since motivated many subfields, and progress on fundamental questions has been distributed across many distinct yet complementary perspectives that range from purely theoretical to predominantly empirical research. Here, we take a hybrid approach and investigate how we can *apply* ideas primarily used in theoretical research to investigate the behavior of a tractable model of a neural network *empirically*. Building upon previous work that studies linear approximations to learning in neural networks [1, 2], we consider a model that uses first-order approximations for the functional updates made during training. However, unlike most previous work, we define this model incrementally by simply *telescoping out* approximations to individual updates made during training so that it more closely approximates the true behavior of a fully trained network. This provides us with a mechanism with which we can conduct empirical investigations into several prominent deep learning phenomena that highlighted how neural networks sometimes generalize in apparently unpredictable ways. Across three case studies, we then show that this model allows us to construct and extract metrics that help predict and understand the a priori unexpected performance of the networks. In particular, we investigate (i) surprising generalization curves (i.e. double descent [3] & grokking [4], Sec. 4.1), (ii) performance differences between gradient boosting and neural networks (Sec. 4.2) and (iii) the success of weight averaging (i.e. linear mode connectivity [5], Appendix B).

2. Background

Notation and preliminaries. Let $f_{\boldsymbol{\theta}} : \mathcal{X} \subseteq \mathbb{R}^d \to \mathcal{Y} \subseteq \mathbb{R}^k$ denote a neural network parameterized by (stacked) model weights $\boldsymbol{\theta} \in \mathbb{R}^p$. Assume we observe a training sample of n input-output pairs $\{\mathbf{x}_i, y_i\}_{i=1}^n$, i.i.d. realizations of the tuple (X, Y) sampled from some distribution P, and wish to learn good model parameters $\boldsymbol{\theta}$ for predicting outputs from this data by minimizing an empirical prediction loss $\frac{1}{n}\sum_{i=1}^n \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i)$, where $\ell : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$ denotes some differentiable loss function. Throughout, we let $k = 1$ for ease of exposition, but unless otherwise indicated our discussion generally extends to $k > 1$. We focus on the case where $\boldsymbol{\theta}$ is optimized by initializing the model with some $\boldsymbol{\theta}_0$ and then iteratively updating the parameters through stochastic gradient descent (SGD) with learning rates γ_t for T steps, where at each $t \in [T] = \{1, \dots, T\}$ we subsample batches $B_t \subseteq [n] = \{1, \dots, n\}$ of the training indices, leading to parameter updates $\Delta\boldsymbol{\theta}_t := \boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}$ as:

*Equal contribution.

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \Delta\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \frac{\gamma_t}{|B_t|}\sum_{i\in B_t} \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x}_i) g_{it}^{\ell} = \boldsymbol{\theta}_{t-1} - \gamma_t \mathbf{T}_t \mathbf{g}_t^{\ell} \tag{1}$$

where $\mathbf{T}_t = [\frac{\mathbf{1}\{1\in B_t\}}{|B_t|}\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x}_1), \ldots, \frac{\mathbf{1}\{n\in B_t\}}{|B_t|}\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x}_n)]$ is a $p \times n$ matrix that has as columns $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x}_i)$, the gradients of the model prediction w.r.t. its parameters for examples in batch B_t (and $\mathbf{0}$ otherwise), and $\mathbf{g}_t^{\ell} = [g_{1t}^{\ell}, \ldots, g_{nt}^{\ell}]^\top$ with $g_{it}^{\ell} = \frac{\partial \ell(f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x}_i), y_i)}{\partial f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x}_i)}$.

Linearized neural networks & tangent kernels. A growing body of work explores the use of *linearized* neural networks as a tool for theoretical [1, 2, 6] and empirical [7–9] study. In this paper, we similarly make extensive use of the following observation (as in e.g. [7]): we can linearize the difference $\Delta f_t(\mathbf{x}) \coloneqq f_{\boldsymbol{\theta}_t}(\mathbf{x}) - f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})$ between two consecutive parameter updates as

$$\Delta f_t(\mathbf{x}) = \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})^\top \Delta\boldsymbol{\theta}_t + \mathcal{O}(||\Delta\boldsymbol{\theta}_t||^2) \approx \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})^\top \Delta\boldsymbol{\theta}_t \coloneqq \Delta\tilde{f}_t(\mathbf{x}) \tag{2}$$

where the quality of the approximation $\Delta\tilde{f}_t(\mathbf{x})$ is good whenever the parameter updates $\Delta\boldsymbol{\theta}_t$ from a single batch are sufficiently small. If Eq. (2) holds exactly (e.g. for infinitesimal γ_t), then running SGD in the network's parameter space to obtain $\Delta\boldsymbol{\theta}_t$ corresponds to executing steepest descent on the function output $f_{\boldsymbol{\theta}}(\mathbf{x})$ directly using the *neural tangent kernel* $K_t^{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i)$ at time-step t [1], i.e.

$$\Delta f_t(\mathbf{x}) = -\gamma_t \sum_{i\in[n]} K_t^{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i) g_{it}^{\ell} \text{ where } K_t^{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i) \coloneqq \frac{\mathbf{1}\{i\in B_t\}}{|B_t|}\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})^\top \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x}_i). \tag{3}$$

Lazy learning [2] occurs as the model gradients remain close to constant during training. For learned parameters $\boldsymbol{\theta}_T$, this implies that the first-order (linear) approximation around $\boldsymbol{\theta}_0$ holds, i.e. $f_{\boldsymbol{\theta}_T}^{lazy}(\mathbf{x}) = f_{\boldsymbol{\theta}_0}(\mathbf{x}) + \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_0}(\mathbf{x})^\top(\boldsymbol{\theta}_T - \boldsymbol{\theta}_0)$. In sufficiently wide neural networks $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_t}(\mathbf{x})$ can theoretically be shown to be constant in some settings [1, 6] – which has been exploited to study convergence and generalization of neural networks [1, 6, 10–13] – but in practice they generally vary during training [7, 8]. This present work relates most closely to recent *empirical* studies using linearizations, such as [9, 14] highlighting differences between lazy learning and real networks, and [7] who empirically explore the relationship between loss landscapes and the evolution of $K_t^{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i)$.

3. Looking at deep learning through a telescoping lens

In this work, we explore whether we can exploit the approximation in Eq. (2) beyond the laziness assumption to gain new insight into neural network learning. Instead of applying the approximation across the entire training trajectory at once as in $f_{\boldsymbol{\theta}_T}^{lazy}(\mathbf{x})$, we consider using it *incrementally* at each batch update to approximate what has been learned at this step. Specifically, we explore whether – instead of studying the final model $f_{\boldsymbol{\theta}_T}(\mathbf{x})$ as a whole – we can gain insight by *telescoping out* the functional updates made throughout training, i.e. exploiting that we can always write:

$$f_{\boldsymbol{\theta}_T}(\mathbf{x}) = f_{\boldsymbol{\theta}_0}(\mathbf{x}) + \sum_{t=1}^{T}[f_{\boldsymbol{\theta}_t}(\mathbf{x}) - f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})] = f_{\boldsymbol{\theta}_0}(\mathbf{x}) + \sum_{t=1}^{T}\Delta f_t(\mathbf{x}) \tag{4}$$

This representation of a learned neural network in terms of its learning trajectory rather than its final parameters is interesting because we are able to better reason about the impact of training on the intermediate updates $\Delta f_t(\mathbf{x})$ than the final function $f_{\boldsymbol{\theta}_T}(\mathbf{x})$ itself. Then, we investigate whether empirically monitoring behaviors of the sum in Eq. (4) while making use of the approximation in Eq. (2) will enable us to gain practical insights into learning in neural networks. That is, we explore the use of the following *telescoping model* $\tilde{f}_{\boldsymbol{\theta}_T}(\mathbf{x})$ as an approximation of a trained neural network:

$$\tilde{f}_{\boldsymbol{\theta}_T}(\mathbf{x}) \coloneqq f_{\boldsymbol{\theta}_0}(\mathbf{x}) + \sum_{t\in[T]} \underbrace{\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})^\top \Delta\boldsymbol{\theta}_t}_{\text{The } \textit{weight-averaging} \text{ representation}} = f_{\boldsymbol{\theta}_0}(\mathbf{x}) - \sum_{t\in[T]} \underbrace{\sum_{i\in[n]} K_t^T(\mathbf{x}, \mathbf{x}_i) g_{it}^{\ell}}_{\text{The } \textit{kernel} \text{ representation}} \tag{5}$$

where $K_t^T(\mathbf{x}, \mathbf{x}_i)$ is determined by the neural tangent kernel as $\gamma_t K_t^{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i)$ in the case of plain SGD, but can take other forms for different optimizers (see Appendix A).

In Fig. 1, we examine the quality of $\tilde{f}_{\boldsymbol{\theta}_t}(\mathbf{x})$ for a 3-layer ReLU network of width 200, trained to discriminate 3-vs-5 from 1000 MNIST examples using the squared loss. In red, we plot its mean average approximation error ($\frac{1}{1000}\sum_{\mathbf{x}\in\mathcal{X}_{test}}|f_{\boldsymbol{\theta}_t}(\mathbf{x}) - \tilde{f}_{\boldsymbol{\theta}_t}(\mathbf{x})|$) as well as the same quantity for $f^{lazy}_{\boldsymbol{\theta}_t}(\mathbf{x})$ (i.e. the first-order expansion around $\boldsymbol{\theta}_0$) in gray and find that iteratively telescoping out the updates instead improves the approximation *by orders of magnitude* – which is also reflected in their prediction performance (see Appendix F.1). Unsurprisingly, γ controls absolute error as it determines $||\Delta\boldsymbol{\theta}_t||$.

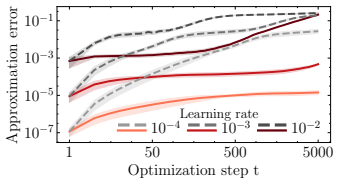

Figure 1: **Approximation error** of telescoping ($\tilde{f}_{\boldsymbol{\theta}_t}(\mathbf{x})$, red) and lazy model ($f^{lazy}_{\boldsymbol{\theta}_t}(\mathbf{x})$, gray).

4. A closer look at modern deep learning phenomena

We now turn to *applying* the telescoping model, presenting case studies revisiting existing experiments that provided evidence for a range of unexpected behaviors of neural networks. These have in common that they highlight cases in which neural networks appear to generalize somewhat *unpredictably*, which is also why each phenomenon has received considerable attention in recent years. For each, we then show that the telescoping model allows us to construct and extract metrics that help predict and understand the unexpected performance of the networks. In particular, we investigate (i) surprising generalization curves (Sec. 4.1) and (ii) performance differences between gradient boosting and neural networks on some tabular tasks (Sec. 4.2) here, and include a third study on linear mode connectivity & the success of weight averaging in Appendix B. An extended literature review can be found in Appendix C and a detailed discussion of all experimental setups in Appendix E.

4.1. Case study 1: Exploring surprising generalization curves and benign overfitting

Classical statistical wisdom provides clear intuitions about overfitting: models that *can* fit the training data too well are expected to generalize poorly [15]. Modern phenomena like double descent [3], however, highlighted that pure capacity measures (capturing what *could* be learned instead of what *is* actually learned) would not be sufficient to understand the complexity-generalization relationship in deep learning [16]. Raw parameter counts, for example, cannot be enough to understand the complexity of what has been learned by a neural network during training because, even when using *the same architecture*, what is learned could be wildly different across various implementation choices within the optimization process – and even at different points during the training process of the same model, as prominently exemplified by the grokking phenomenon [4]. Here, we explore whether the telescoping model allows us to gain insight into the relative complexity of what is learned.

A candidate complexity measure that avoids the shortcomings listed above because it only considers the behavior of the final fitted model was recently used by [17] in their study of non-deep double descent. Because their measure $p^0_{\hat{\mathbf{s}}}$ builds on ideas from the literature on smoothers [18], it requires to express learned predictions as a function of training labels, i.e. as $f(\mathbf{x}) = \hat{\mathbf{s}}(\mathbf{x})\mathbf{y} = \sum_{i\in[n]}\hat{s}^i(\mathbf{x})y_i$. Then, $p^0_{\hat{\mathbf{s}}} \equiv p(\mathcal{I}_0, \hat{\mathbf{s}}(\cdot)) = \frac{n}{|\mathcal{I}_0|}\sum_{j\in\mathcal{I}_0}||\hat{\mathbf{s}}(\mathbf{x}^0_j)||^2$ denote the *effective parameters* used by the model when issuing predictions for inputs $\{\mathbf{x}^0_j\}_{j\in\mathcal{I}_0}$ (with indices collected in $\mathcal{I}_0$); intuitively the larger $p^0_{\hat{\mathbf{s}}}$, the less smoothing is performed. Does the telescoping model allow us to make use of $p^0_{\hat{\mathbf{s}}}$ as a proxy for complexity? Yes! For the special case of a single output ($k = 1$) and training with squared loss $\ell(f(\mathbf{x}), y) = \frac{1}{2}(y - f(\mathbf{x}))^2$, if we let $\mathbf{y} = [y_1, \ldots, y_n]^\top$ and $\mathbf{f}_{\boldsymbol{\theta}_t} = [f_{\boldsymbol{\theta}_t}(\mathbf{x}_1), \ldots, f_{\boldsymbol{\theta}_t}(\mathbf{x}_n)]^\top$, the SGD weight update simplifies to $\Delta\boldsymbol{\theta}_t = \gamma_t\mathbf{T}_t(\mathbf{y} - \mathbf{f}_{\boldsymbol{\theta}_{t-1}})$. Assuming the telescoping approximation holds exactly, this implies functional updates $\Delta\tilde{f}_t(\mathbf{x}) = \gamma_t\nabla_{\boldsymbol{\theta}}f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})^\top\mathbf{T}_t(\mathbf{y} - \tilde{\mathbf{f}}_{\boldsymbol{\theta}_{t-1}})$. Recursively substituting this into Eq. (5) then allows us to write $\tilde{f}_{\boldsymbol{\theta}_t}(\mathbf{x}) = \mathbf{s}_{\boldsymbol{\theta}_t}(\mathbf{x})\mathbf{y} + c^0_{\boldsymbol{\theta}_t}(\mathbf{x})$, where the $1\times n$ vector $\mathbf{s}_{\boldsymbol{\theta}_t}(\mathbf{x})$ is a function of the kernels $\{K^t_{t'}(\cdot,\cdot)\}_{t'\leq t}$

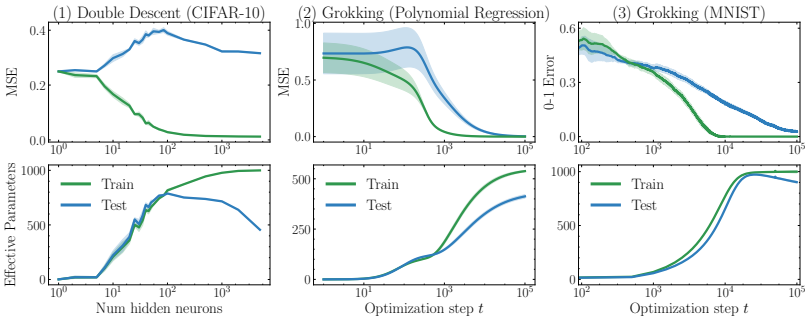

Figure 2: **Double Descent (1) and Grokking (2, 3)**. Error (top) vs effective parameters $p^0_{\hat{\mathbf{s}}}$ (bottom).

alone, and the scalar $c^0_{\boldsymbol{\theta}_t}(\mathbf{x})$ is a function of the $\{K^t_{t'}(\cdot,\cdot)\}_{t' \leq t}$ and the initialization $f_{\boldsymbol{\theta}_0}(\cdot)$ – enabling us to use $p^0_{\hat{\mathbf{s}}}$ with $\mathbf{s}_{\boldsymbol{\theta}_t}(\mathbf{x})$! We derive $\mathbf{s}_{\boldsymbol{\theta}_t}(\mathbf{x})$ and $c^0_{\boldsymbol{\theta}_t}(\mathbf{x})$ for different optimizers in Appendix D.1.

Double descent: Model complexity vs model size. [3] made the surprising observation that, while training error always monotonically decreases as model size (measured by parameter count) increases, test error initially worsens when the model is increasingly able to overfit to the training data (as is expected) but can then improve again as model size is increased further past the so-called *interpolation threshold* where perfect training performance is achieved. This *double descent* shape appears to contradict the classical U-shaped relationship between model complexity and test error [15]! Here, we investigate whether tracking $p^0_{\hat{\mathbf{s}}}$ on train- and test data separately will allow us to gain new insight into the phenomenon in neural networks. In Fig. 2 (1), we replicate the binary classification example of double descent in neural networks of [3], training single-hidden-layer ReLU networks of increasing width to distinguish cats and dogs on CIFAR-10. First, we indeed replicate the characteristic behavior of error curves of [3]. Measuring learned complexity using $p^0_{\hat{\mathbf{s}}}$, we then find that while $p^{train}_{\hat{\mathbf{s}}}$ monotonically increases as model size is increasing, the effective parameters used on test data $p^{test}_{\hat{\mathbf{s}}}$ implied by the trained neural network *decrease* as model size is increased past the interpolation threshold. Thus, paralleling the findings made in [17] for linear regression and tree-based methods, we find that distinguishing between train- and test-time complexity of a neural network using $p^0_{\hat{\mathbf{s}}}$ provides new quantitative evidence that bigger networks are *not* necessarily learning more complex prediction functions for unseen test examples, which resolves the ostensible tension between deep double descent and the classical U-curve. Importantly, note that $p^{test}_{\hat{\mathbf{s}}}$ can be computed without access to test-time labels, which means that the observed difference between $p^{train}_{\hat{\mathbf{s}}}$ and $p^{test}_{\hat{\mathbf{s}}}$ allows to *quantify* whether there is *benign overfitting* [19, 20] in a neural network.

Grokking: Model complexity throughout training. The grokking phenomenon [4] then showcased that improvements in test performance *during a single training run* can occur long after perfect training performance has been achieved (contradicting early stopping practice!). While [21] attribute this to weight decay causing $||\boldsymbol{\theta}_t||$ to shrink late in training – which they demonstrate on an MNIST example using unusually large $\boldsymbol{\theta}_0$ –, [22] highlight that grokking can also occur as the weight norm $||\boldsymbol{\theta}_t||$ *grows* later in training – which they demonstrate on a polynomial regression task. In Fig. 2 we replicate a version of both experiments while tracking $p^0_{\hat{\mathbf{s}}}$ to investigate whether this provides new insight into this apparent disagreement. Then, we observe that the continued improvement in test error, past the point of perfect training performance, is associated with divergence of $p^{train}_{\hat{\mathbf{s}}}$ and $p^{test}_{\hat{\mathbf{s}}}$ in *both* experiments (analogous to the double descent experiment), suggesting that grokking may reflect *transition into* a measurably benign overfitting regime during training.

4.2. Case study 2: Understanding differences between gradient boosting and neural networks

Despite their overwhelming successes on image and language data, neural networks are – perhaps surprisingly – still widely considered to be outperformed by *gradient boosted trees* (GBTs) on *tabular data*, an important modality in many data science applications. Exploring this apparent Achilles

heel of neural networks has therefore been the goal of multiple extensive benchmarking studies [23, 24]. Here, we concentrate on a specific empirical finding of [24]: their results suggest that GBTs may particularly outperform deep learning on heterogeneous data with greater irregularity in input features, a characteristic often present in tabular data. Below, we first show that the telescoping model offers a useful lens to compare and contrast the two methods, and then use this insight to provide and test a new explanation of why GBTs can perform better in the presence of dataset irregularities.

Identifying (dis)similarities between GBTs and neural networks. We begin by introducing GBTs [25] closely following [15, Ch. 10.10]. GBTs, with learning rate γ and initialized at $h_0(\mathbf{x})$, consists of a sequence $\hat{f}_T^{GB}(\mathbf{x}) = h_0(\mathbf{x}) + \gamma \sum_{t=1}^{T} \hat{h}_t(\mathbf{x})$ where each $\hat{h}_t(\mathbf{x})$ iteratively improves upon the existing predictions $\hat{f}_{t-1}^{GB}(\mathbf{x})$. Specifically, GBTs execute steepest descent in function space *directly*, where each update $\hat{h}_t(\cdot)$ simply outputs an estimate of the negative gradient of the loss function w.r.t. the previous model, i.e. $\hat{h}_t(\mathbf{x}_i) = -\hat{g}_t^{\ell}(\mathbf{x})$ where $g_t^{\ell}(\mathbf{x}) = \partial \ell(\hat{f}_{t-1}^{GB}(\mathbf{x}), y(\mathbf{x}))/\partial \hat{f}_{t-1}^{GB}(\mathbf{x})$. This is achieved by fitting *trees* $\hat{h}_t(\cdot)$ to the training input-gradient pairs $\{\mathbf{x}_i, -g_{it}^{\ell}\}_{i\in[n]}$, which can then also be evaluated at new inputs. Focusing on trees that issue predictions by averaging, which are sometimes interpreted as kernel smoothers [26–28], lets us express the learned predictor as:

$$\hat{f}^{GB}(\mathbf{x}) = h_0(\mathbf{x}) - \gamma \sum_{t\in[T]} \sum_{i\in[n]} \frac{\mathbf{1}\{l_{h_t}(\mathbf{x}) = l_{h_t}(\mathbf{x}_i)\}}{n_{l(\mathbf{x})}} g_{it}^{\ell} = h_0(\mathbf{x}) - \gamma \sum_{t\in[T]} \sum_{i\in[n]} K_{\hat{h}_t}(\mathbf{x}, \mathbf{x}_i) g_{it}^{\ell} \quad (6)$$

where $l_{\hat{h}_t}(\mathbf{x})$ is the leaf $\mathbf{x}$ falls into, $n_{l(\mathbf{x})} = \sum_{i\in[n]} \mathbf{1}\{l_{h_t}(\mathbf{x}) = l_{h_t}(\mathbf{x}_i)\}$ counts the training examples in it and $K_{\hat{h}_t}(\mathbf{x}, \mathbf{x}_i) = {}^1/_{n_{l(\mathbf{x})}} \mathbf{1}\{l_{\hat{h}_t}(\mathbf{x}) = l_{\hat{h}_t}(\mathbf{x}_i)\}$ is the kernel of the t^{th} tree. Comparing Eq. (6) to the kernel representation of the telescoping model (Eq. (5)), we make a perhaps surprising observation: the telescoping model of neural networks and GBTs have *identical* structure, differing only in kernel!

Why can GBTs outperform deep learning in the presence of dataset irregularities? Contrasting Eq. (5) and Eq. (6) thus suggests that at least some of the performance differences between neural networks and GBTs are likely to be rooted in the differences between $K_t^{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i)$ and $K_{\hat{h}_t}(\mathbf{x}, \mathbf{x}_i)$. One difference obvious: it is possible that either kernel encodes a better inductive bias to fit the outcome-generating process of a given dataset. Another difference is more subtle and relates to the behavior of the learned model on new inputs $\mathbf{x}$ – the tree kernels are likely to behave *much* more predictable at test-time than the tangent kernels. To see this, note that for the tree kernels we have that $\forall \mathbf{x} \in \mathcal{X}$ and $\forall i \in [n]$, $0 \leq K_{\hat{h}_t}(\mathbf{x}, \mathbf{x}_i) \leq 1$ and $\sum_{i\in[n]} K_{\hat{h}_t}(\mathbf{x}, \mathbf{x}_i) = 1$. For the tangent kernels on the other hand, $K_t^{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i)$ is in general unbounded and could behave very erratically for $\mathbf{x}$ not observed during training. This leads us to hypothesize that this difference may be able to explain [24]'s observation that GBTs perform better whenever features are heavy-tailed: if a test point $\mathbf{x}$ is very different from training points, $\mathbf{k}_t^{\boldsymbol{\theta}}(\mathbf{x}) := [K_t^{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_1), \ldots, K_t^{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_n)]^\top$ may behave very differently than at train-time while $\mathbf{k}_{\hat{h}_t}(\mathbf{x}) = [K_{\hat{h}_t}(\mathbf{x}, \mathbf{x}_1), \ldots, K_{\hat{h}_t}(\mathbf{x}, \mathbf{x}_n)]^\top$ will be less affected.

We empirically test this hypothesis on standard tabular benchmark datasets proposed in [23], comparing the models as inputs become increasingly irregular. As a simple notion for input irregularity, we apply principal component analysis to the inputs to obtain a lower dimensional representation of the data and sort the observations according to their distance from the centroid. For a fixed trained model, we then evaluate on test sets consisting of increasing proportions p of the most irregular inputs. We compare GBTs to neural networks by examining (i) the most extreme values their kernel weights take at test-time relative to the training data (measured as $\frac{\frac{1}{T}\sum_{t=1}^{T} \max_{j\in\mathcal{I}_{test}^p} ||\mathbf{k}_t(\mathbf{x}_j)||_2}{\frac{1}{T}\sum_{t=1}^{T} \max_{i\in\mathcal{I}_{train}} ||\mathbf{k}_t(\mathbf{x}_i)||_2}$) and (ii) how their relative mean squared error (measured as $\frac{MSE_{NN}^p - MSE_{GBT}^p}{MSE_{NN}^0 - MSE_{GBT}^0}$)

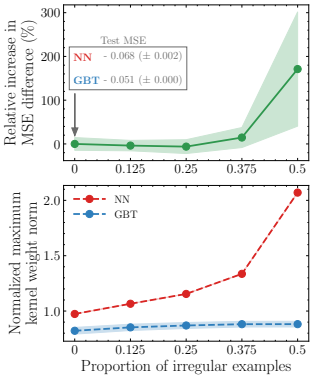

Figure 3: **Neural Nets vs GBTs:** Performance (top) & behavior of kernels (bottom).

changes as the proportion p of irregular examples increases. In Fig. 3 on `houses` and in Appendix F.3 using additional datasets, we first observe that GBTs outperform neural networks already in the absence of irregular examples; this highlights that there may indeed be differences in the kernels' fit to the outcome-generating processes. Consistent with our expectations, we then find that as the test data becomes more irregular, the performance of the neural network decays faster than the GBTs'. Importantly, this is well tracked by their kernels, where the unbounded nature of the network's tangent kernel indeed results in changed behavior on irregular examples.

5. Conclusion

We investigated the utility of a telescoping model for neural network learning, consisting of a sequence of linear approximations, as a tool for understanding deep learning phenomena. We believe there are many interesting opportunities to study this in more generality, empirically *and* theoretically.

Acknowledgements

We would like to thank James Bayliss, who first suggested to us to look into explicitly unravelling SGD updates to write trained neural networks as approximate smoothers to study deep double descent after a seminar on our paper [17] on non-deep double descent. This suggestion ultimately inspired many investigations far beyond the original double descent context. AC and AJ gratefully acknowledge funding from AstraZeneca and the Cystic Fybrosis Trust, respectively. This work was supported by a G-Research grant, and Azure sponsorship credits granted by Microsoft's AI for Good Research Lab.

References

[1] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

[2] Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. *Advances in neural information processing systems*, 32, 2019.

[3] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[4] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

[5] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, pages 3259–3269. PMLR, 2020.

[6] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32, 2019.

[7] Stanislav Fort, Gintare Karolina Dziugaite, Mansheej Paul, Sepideh Kharaghani, Daniel M Roy, and Surya Ganguli. Deep learning versus kernel learning: an empirical study of loss landscape

geometry and the time evolution of the neural tangent kernel. *Advances in Neural Information Processing Systems*, 33:5850–5861, 2020.

[8] Chaoyue Liu, Libin Zhu, and Misha Belkin. On the linearity of large non-linear models: when and why the tangent kernel is constant. *Advances in Neural Information Processing Systems*, 33:15954–15964, 2020.

[9] Guillermo Ortiz-Jiménez, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. What can linearized neural networks actually say about generalization? *Advances in Neural Information Processing Systems*, 34:8998–9010, 2021.

[10] Simon Du, Jason Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *International conference on machine learning*, pages 1675–1685. PMLR, 2019.

[11] Alberto Bietti and Julien Mairal. On the inductive bias of neural tangent kernels. *Advances in Neural Information Processing Systems*, 32, 2019.

[12] Behrooz Ghorbani, Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Limitations of lazy training of two-layers neural network. *Advances in Neural Information Processing Systems*, 32, 2019.

[13] Mario Geiger, Stefano Spigler, Arthur Jacot, and Matthieu Wyart. Disentangling feature and lazy training in deep neural networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2020(11):113301, 2020.

[14] Jaehoon Lee, Samuel Schoenholz, Jeffrey Pennington, Ben Adlam, Lechao Xiao, Roman Novak, and Jascha Sohl-Dickstein. Finite versus infinite neural networks: an empirical study. *Advances in Neural Information Processing Systems*, 33:15156–15172, 2020.

[15] Trevor Hastie, Robert Tibshirani, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[16] Mikhail Belkin. Fit without fear: remarkable mathematical phenomena of deep learning through the prism of interpolation. *Acta Numerica*, 30:203–248, 2021.

[17] Alicia Curth, Alan Jeffares, and Mihaela van der Schaar. A u-turn on double descent: Rethinking parameter counting in statistical learning. *Advances in Neural Information Processing Systems*, 36, 2023.

[18] Trevor Hastie and Robert Tibshirani. Generalized additive models. *Monographs on statistics and applied probability. Chapman & Hall*, 43:335, 1990.

[19] Peter L Bartlett, Philip M Long, Gábor Lugosi, and Alexander Tsigler. Benign overfitting in linear regression. *Proceedings of the National Academy of Sciences*, 117(48):30063–30070, 2020.

[20] Yaoqing Yang, Liam Hodgkinson, Ryan Theisen, Joe Zou, Joseph E Gonzalez, Kannan Ramchandran, and Michael W Mahoney. Taxonomizing local versus global structure in neural network loss landscapes. *Advances in Neural Information Processing Systems*, 34:18722–18733, 2021.

[21] Ziming Liu, Eric J Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data. In *The Eleventh International Conference on Learning Representations*, 2022.

[22] Tanishq Kumar, Blake Bordelon, Samuel J Gershman, and Cengiz Pehlevan. Grokking as the transition from lazy to rich training dynamics. In *The Twelfth International Conference on Learning Representations*, 2024.

[23] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in neural information processing systems*, 35:507–520, 2022.

[24] Duncan McElfresh, Sujay Khandagale, Jonathan Valverde, Vishak Prasad C, Ganesh Ramakrishnan, Micah Goldblum, and Colin White. When do neural nets outperform boosted trees on tabular data? *Advances in Neural Information Processing Systems*, 36, 2024.

[25] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[26] Yi Lin and Yongho Jeon. Random forests and adaptive nearest neighbors. *Journal of the American Statistical Association*, 101(474):578–590, 2006.

[27] Gérard Biau and Luc Devroye. On the layered nearest neighbour estimate, the bagged nearest neighbour estimate and the random forest method in regression and classification. *Journal of Multivariate Analysis*, 101(10):2499–2518, 2010.

[28] Alicia Curth, Alan Jeffares, and Mihaela van der Schaar. Why do random forests work? understanding tree ensembles as self-regularizing adaptive smoothers. *arXiv preprint arXiv:2402.01502*, 2024.

[29] Simon JD Prince. *Understanding Deep Learning*. MIT press, 2023.

[30] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[31] Samuel K Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries. *arXiv preprint arXiv:2209.04836*, 2022.

[32] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

[33] Sidak Pal Singh and Martin Jaggi. Model fusion via optimal transport. *Advances in Neural Information Processing Systems*, 33:22045–22055, 2020.

[34] Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks. *arXiv preprint arXiv:2110.06296*, 2021.

[35] Taiga Abe, E Kelly Buchanan, Geoff Pleiss, and John Patrick Cunningham. Pathologies of predictive diversity in deep ensembles. *Transactions on Machine Learning Research*, 2023.

[36] Alan Jeffares, Tennison Liu, Jonathan Crabbé, and Mihaela van der Schaar. Joint training of deep ensembles fails due to learner collusion. *Advances in Neural Information Processing Systems*, 36, 2024.

[37] Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? *Advances in neural information processing systems*, 33:512–523, 2020.

[38] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, pages 23965–23998. PMLR, 2022.

[39] Leshem Choshen, Elad Venezian, Noam Slonim, and Yoav Katz. Fusing finetuned models for better pretraining. *arXiv preprint arXiv:2204.03044*, 2022.

[40] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1995.

[41] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[42] Brady Neal, Sarthak Mittal, Aristide Baratin, Vinayak Tantia, Matthew Scicluna, Simon Lacoste-Julien, and Ioannis Mitliagkas. A modern take on the bias-variance tradeoff in neural networks. *arXiv preprint arXiv:1810.08591*, 2018.

[43] Brady Neal. On the bias-variance tradeoff: Textbooks need an update. *arXiv preprint arXiv:1912.08286*, 2019.

[44] F Vallet, J-G Cailton, and Ph Refregier. Linear and nonlinear extension of the pseudo-inverse solution for learning boolean functions. *Europhysics Letters*, 9(4):315, 1989.

[45] Siegfried Bös and Manfred Opper. Dynamics of training. *Advances in Neural Information Processing Systems*, 9, 1996.

[46] Madhu S Advani, Andrew M Saxe, and Haim Sompolinsky. High-dimensional dynamics of generalization error in neural networks. *Neural Networks*, 132:428–446, 2020.

[47] Stefano Spigler, Mario Geiger, Stéphane d'Ascoli, Levent Sagun, Giulio Biroli, and Matthieu Wyart. A jamming transition from under-to over-parametrization affects loss landscape and generalization. *arXiv preprint arXiv:1810.09665*, 2018.

[48] Marco Loog, Tom Viering, Alexander Mey, Jesse H Krijthe, and David MJ Tax. A brief prehistory of double descent. *Proceedings of the National Academy of Sciences*, 117(20):10625–10626, 2020.

[49] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021.

[50] Zheng He, Zeke Xie, Quanzhi Zhu, and Zengchang Qin. Sparse double descent: Where network pruning aggravates overfitting. In *International Conference on Machine Learning*, pages 8635–8659. PMLR, 2022.

[51] Preetum Nakkiran, Prayaag Venkat, Sham Kakade, and Tengyu Ma. Optimal regularization can mitigate double descent. *arXiv preprint arXiv:2003.01897*, 2020.

[52] Mikhail Belkin, Daniel Hsu, and Ji Xu. Two models of double descent for weak features. *SIAM Journal on Mathematics of Data Science*, 2(4):1167–1180, 2020.

[53] Michal Derezinski, Feynman T Liang, and Michael W Mahoney. Exact expressions for double descent and implicit regularization via surrogate random design. *Advances in neural information processing systems*, 33:5152–5164, 2020.

[54] Trevor Hastie, Andrea Montanari, Saharon Rosset, and Ryan J Tibshirani. Surprises in high-dimensional ridgeless least squares interpolation. *The Annals of Statistics*, 50(2):949–986, 2022.

[55] Rylan Schaeffer, Mikail Khona, Zachary Robertson, Akhilan Boopathy, Kateryna Pistunova, Jason W Rocks, Ila Rani Fiete, and Oluwasanmi Koyejo. Double descent demystified: Identifying, interpreting & ablating the sources of a deep learning puzzle. *arXiv preprint arXiv:2303.14151*, 2023.

[56] Lin Chen, Yifei Min, Mikhail Belkin, and Amin Karbasi. Multiple descent: Design your own generalization curve. *Advances in Neural Information Processing Systems*, 34:8898–8912, 2021.

[57] Ben Adlam and Jeffrey Pennington. Understanding double descent requires a fine-grained bias-variance decomposition. *Advances in neural information processing systems*, 33:11022–11032, 2020.

[58] Stéphane d'Ascoli, Maria Refinetti, Giulio Biroli, and Florent Krzakala. Double trouble in double descent: Bias and variance (s) in the lazy regime. In *International Conference on Machine Learning*, pages 2280–2290. PMLR, 2020.

[59] Licong Lin and Edgar Dobriban. What causes the test error? going beyond bias-variance via anova. *The Journal of Machine Learning Research*, 22(1):6925–7006, 2021.

[60] Mikhail Belkin, Siyuan Ma, and Soumik Mandal. To understand deep learning we need to understand kernel learning. In *International Conference on Machine Learning*, pages 541–549. PMLR, 2018.

[61] Siyuan Ma, Raef Bassily, and Mikhail Belkin. The power of interpolation: Understanding the effectiveness of sgd in modern over-parametrized learning. In *International Conference on Machine Learning*, pages 3325–3334. PMLR, 2018.

[62] Niladri S Chatterji and Philip M Long. Finite-sample analysis of interpolating linear classifiers in the overparameterized regime. *The Journal of Machine Learning Research*, 22(1):5721–5750, 2021.

[63] Neil Mallinar, James Simon, Amirhesam Abedsoltan, Parthe Pandit, Misha Belkin, and Preetum Nakkiran. Benign, tempered, or catastrophic: Toward a refined taxonomy of overfitting. *Advances in Neural Information Processing Systems*, 35:1182–1195, 2022.

[64] Abraham J Wyner, Matthew Olson, Justin Bleich, and David Mease. Explaining the success of adaboost and random forests as interpolating classifiers. *Journal of Machine Learning Research*, 18(48):1–33, 2017.

[65] Moritz Haas, David Holzmüller, Ulrike Luxburg, and Ingo Steinwart. Mind the spikes: Benign overfitting of kernels and neural networks in fixed dimension. *Advances in Neural Information Processing Systems*, 36, 2024.

[66] Olivier Bousquet, Stéphane Boucheron, and Gábor Lugosi. Introduction to statistical learning theory. In *Summer school on machine learning*, pages 169–207. Springer, 2003.

[67] John Moody. The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. *Advances in neural information processing systems*, 4, 1991.

[68] David MacKay. Bayesian model comparison and backprop nets. *Advances in neural information processing systems*, 4, 1991.

[69] Wesley J Maddox, Gregory Benton, and Andrew Gordon Wilson. Rethinking parameter counting in deep models: Effective dimensionality revisited. *arXiv preprint arXiv:2003.02139*, 2020.

[70] Ziming Liu, Ouail Kitouni, Niklas S Nolte, Eric Michaud, Max Tegmark, and Mike Williams. Towards understanding grokking: An effective theory of representation learning. *Advances in Neural Information Processing Systems*, 35:34651–34663, 2022.

[71] Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. *arXiv preprint arXiv:2301.05217*, 2023.

[72] Vikrant Varma, Rohin Shah, Zachary Kenton, János Kramár, and Ramana Kumar. Explaining grokking through circuit efficiency. *arXiv preprint arXiv:2309.02390*, 2023.

[73] Vimal Thilak, Etai Littwin, Shuangfei Zhai, Omid Saremi, Roni Paiss, and Joshua Susskind. The slingshot mechanism: An empirical study of adaptive optimizers and the grokking phenomenon. *arXiv preprint arXiv:2206.04817*, 2022.

[74] Kaifeng Lyu, Jikai Jin, Zhiyuan Li, Simon Shaolei Du, Jason D Lee, and Wei Hu. Dichotomy of early and late phase implicit biases can provably induce grokking. In *The Twelfth International Conference on Learning Representations*, 2024.

[75] Thomas G Dietterich. Ensemble learning. *The handbook of brain theory and neural networks*, 2(1):110–125, 2002.

[76] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[77] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective. *arXiv preprint arXiv:1912.02757*, 2019.

[78] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[79] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. *Advances in neural information processing systems*, 31, 2018.

[80] C Daniel Freeman and Joan Bruna. Topology and geometry of half-rectified network optimization. *arXiv preprint arXiv:1611.01540*, 2016.

[81] Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In *International conference on machine learning*, pages 1309–1318. PMLR, 2018.

[82] Gül Sena Altıntaş, Gregor Bachmann, Lorenzo Noci, and Thomas Hofmann. Disentangling linear mode connectivity. In *UniReps: the First Workshop on Unifying Representations in Neural Models*, 2023.

[83] Alexandre Rame, Matthieu Kirchmeyer, Thibaud Rahier, Alain Rakotomamonjy, Patrick Gallinari, and Matthieu Cord. Diverse weight averaging for out-of-distribution generalization. *Advances in Neural Information Processing Systems*, 35:10821–10836, 2022.

[84] Gabriel Ilharco, Mitchell Wortsman, Samir Yitzhak Gadre, Shuran Song, Hannaneh Hajishirzi, Simon Kornblith, Ali Farhadi, and Ludwig Schmidt. Patching open-vocabulary models by interpolating weights. *Advances in Neural Information Processing Systems*, 35:29262–29277, 2022.

[85] Frederik Benzing, Simon Schug, Robert Meier, Johannes Von Oswald, Yassir Akram, Nicolas Zucchet, Laurence Aitchison, and Angelika Steger. Random initialisations performing above chance and how to find them. *arXiv preprint arXiv:2209.07509*, 2022.

[86] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[87] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[88] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[89] Devin Kwok, Nikhil Anand, Jonathan Frankle, Gintare Karolina Dziugaite, and David Rolnick. Dataset difficulty and the role of inductive bias. *arXiv preprint arXiv:2401.01867*, 2024.

[90] Nabeel Seedat, Fergus Imrie, and Mihaela van der Schaar. Dissecting sample hardness: Fine-grained analysis of hardness characterization methods. In *The Twelfth International Conference on Learning Representations*, 2023.

[91] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[92] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[93] Yerlan Idelbayev. Proper ResNet implementation for CIFAR10/CIFAR100 in PyTorch. https://github.com/akamaster/pytorch_resnet_cifar10. Accessed: 2024-05-15.

[94] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 7. Granada, Spain, 2011.

[95] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

Appendix

This appendix is structured as follows: In Appendix A, we derive the linearized functional updates implied by modern optimizers. In Appendix B, we present an additional case study investigating linear mode connectivity the success of weight averaging. Further, Appendix C presents an extended literature review related to the phenomena we consider in , Appendix D presents additional theoretical derivations, Appendix E presents an extended discussion of experimental setups and Appendix F presents additional results.

Appendix A. Understanding the effect of modern design choices on linearized functional updates

The literature on the neural tangent kernel primarily considers plain SGD, while modern deep learning practice typically relies on a range of important modifications to the training process (see e.g. [29, Ch. 6]) – this includes many of the experiments demonstrating surprising deep learning phenomena we examine in Sec. 4. To enable us to use modern optimizers, we derive their implied linearized functional updates through the weight-averaging representation $\Delta\tilde{f}_t(\mathbf{x}) = \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})^\top \Delta\boldsymbol{\theta}_t$ below. As a by-product, we find that this provides us with an interesting and pedagogical formalism to reason about the relative effect of different design choices in neural network training.

Momentum with scalar hyperparameter β_1 smoothes weight updates by employing an exponentially weighted average over the previous parameter gradients as $\Delta\boldsymbol{\theta}_t = -\gamma_t \frac{1-\beta_1}{1-\beta_1^t} \sum_{k=1}^t \beta_1^{t-k} \mathbf{T}_k \mathbf{g}_k^\ell$ instead of using the current gradients alone. This implies linearized functional updates

$$\Delta\tilde{f}_t(\mathbf{x}) = -\gamma_t \tfrac{1-\beta_1}{1-\beta_1^t} \textstyle\sum_{i\in[n]} (K_t^{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i) g_{it}^\ell + \sum_{k=1}^{t-1} \beta_1^{t-k} K_{t,k}^{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i) g_{ik}^\ell) \tag{7}$$

where $K_{t,k}^{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i) \coloneqq \frac{\mathbf{1}\{i\in B_k\}}{|B_k|} \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})^\top \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{k-1}}(\mathbf{x}_i)$ denotes the cross-temporal tangent kernel. Thus, the functional updates also utilize *previous* loss gradients, where their weight is determined using an inner product of the model gradient features from different time steps. If $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_t}(\mathbf{x})$ is constant throughout training and we use full-batch GD, then the contribution of each training example i to $\Delta\tilde{f}_t(\mathbf{x})$ reduces to $-\gamma_t K_0^{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i) \frac{1-\beta_1}{1-\beta_1^t} [\sum_{k=1}^t \beta_1^{t-k} g_{ik}^\ell]$, an exponentially weighted moving average over its past loss gradients – making the effect of momentum on functional updates analogous to its effect on updates in parameter space. However, if $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_t}(\mathbf{x})$ changes over time, it is e.g. possible that $K_{k,t}^{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i)$ has opposite sign from $K_t^{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i)$ in which case momentum reduces instead of amplifies the effect of a previous g_{it}^ℓ. This is more obvious when re-writing Eq. (7) to collect all terms containing a specific g_{it}^ℓ, leading to $K_t^T(\mathbf{x}, \mathbf{x}_i) = \sum_{k=t}^T \gamma_k \frac{1-\beta_1}{1-\beta_1^k} \beta_1^{k-t} K_{k,t}(\mathbf{x}, \mathbf{x}_i)$ for Eq. (5).

Weight decay with scalar hyperparameter λ uses $\Delta\boldsymbol{\theta}_t = -\gamma_t(\mathbf{T}_t \mathbf{g}_t^\ell + \lambda\boldsymbol{\theta}_{t-1})$. For constant learning rate γ this gives $\boldsymbol{\theta}_t = \boldsymbol{\theta}_0 - \sum_{k=1}^t \gamma(\mathbf{T}_k \mathbf{g}_k^\ell + \lambda\boldsymbol{\theta}_{k-1}) = (1-\lambda\gamma)^t \boldsymbol{\theta}_0 - \gamma \sum_{k=1}^t (1-\lambda\gamma)^{t-k} \mathbf{T}_k \mathbf{g}_k^\ell$. This then implies linearized functional updates

$$\begin{aligned} \Delta\tilde{f}_t(\mathbf{x}) = -\gamma \textstyle\sum_{i\in[n]} (K_t(\mathbf{x}, \mathbf{x}_i) g_{it}^\ell - \lambda\gamma \sum_{k=1}^{t-1} (1-\lambda\gamma)^{t-1-k} K_{t,k}(\mathbf{x}, \mathbf{x}_i) g_{ik}^\ell) \\ -\gamma\lambda(1-\lambda\gamma)^{t-1} \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})^\top \boldsymbol{\theta}_0 \end{aligned} \tag{8}$$

For full-batch GD and constant tangent kernels, $-\gamma K_0^{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i)[g_{it} - \lambda\gamma \sum_{k=1}^{t-1} (1-\lambda\gamma)^{t-1-k} g_{ik}]$ is the contribution of each training example to the functional updates, which effectively decays the previous contributions of this example. Further, comparing the signs in Eq. (8) to Eq. (7) highlights that momentum with $\beta_1 > (1-\lambda\gamma)$ approximately *offsets* the effect of weight decay on the learned updates in function space (in which case weight decay mainly acts through the term decaying $\boldsymbol{\theta}_0$).

Adaptive & parameter-dependent learning rates are another important modification in practice which enable the use of different step-sizes across parameters by dividing $\Delta\boldsymbol{\theta}_t$ *elementwise* by a $p \times 1$ scaling vector $\boldsymbol{\phi}_t$. Most prominently, this is used to adaptively normalize the magnitude of updates (e.g. Adam [30] uses $\boldsymbol{\phi}_t = \sqrt{\frac{1-\beta_2}{1-\beta_2^t}\sum_{k=1}^{t}\beta_2^{t-k}[\mathbf{T}_k\mathbf{g}_k^\ell]^2 + \epsilon}$). When combined with plain SGD, this results in kernel $K_t^{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{x}_i) = \frac{\mathbf{1}\{i\in B_t\}}{|B_t|}\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})^\top \text{diag}(\frac{1}{\boldsymbol{\phi}_t})\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x}_i)$. This expression highlights that $\boldsymbol{\phi}_t$ admits an elegant interpretation as re-scaling the relative influence of features on the tangent kernel, similar to structured kernels in non-parametric regression [15, Ch. 6.4.1].

Architecture design choices also impact the form of the kernel. One important practical example is whether $f_{\boldsymbol{\theta}}(\mathbf{x})$ applies a non-linear activation function to the output $g_{\boldsymbol{\theta}}(\mathbf{x}) \in \mathbb{R}$ of its final layer. Consider the choice of using the sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ for a binary classification problem and recall $\frac{\partial}{\partial x}\sigma(x) = \sigma(x)(1-\sigma(x)) \in (0, 1/4]$, which is largest where $\sigma(x) = 1/2$ and smallest when $\sigma(x) \to 0 \vee 1$. If $K_t^{\boldsymbol{\theta},g}(\mathbf{x}, \mathbf{x}_i) := \frac{\mathbf{1}\{i\in B_t\}}{|B_t|}\nabla_{\boldsymbol{\theta}}\, g_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})^\top \nabla_{\boldsymbol{\theta}}\, g_{\boldsymbol{\theta}_{t-1}}(\mathbf{x}_i)$ denotes the tangent kernel of the model without activation, it is easy to see that the tangent kernel of the model $\sigma(g_{\boldsymbol{\theta}_t}(\mathbf{x}))$ is

$$K_t^{\boldsymbol{\theta},\sigma}(\mathbf{x}, \mathbf{x}_i) = \sigma(g_{\boldsymbol{\theta}_t}(\mathbf{x}))(1-\sigma(g_{\boldsymbol{\theta}_t}(\mathbf{x})))\sigma(g_{\boldsymbol{\theta}_t}(\mathbf{x}_i))(1-\sigma(g_{\boldsymbol{\theta}_t}(\mathbf{x}_i)))K_t^{\boldsymbol{\theta},g}(\mathbf{x}, \mathbf{x}_i) \tag{9}$$

This indicates that $K_t^{\boldsymbol{\theta},\sigma}(\mathbf{x}, \mathbf{x}_i)$ will give relatively higher weight in functional updates to training examples i for which the model is uncertain ($\sigma(g(\mathbf{x}_i)) \approx 1/2$) and lower weight to examples where the model is certain ($\sigma(g_{\boldsymbol{\theta}_t}(\mathbf{x}_i)) \approx 0 \vee 1$) – *regardless* of whether $\sigma(g_{\boldsymbol{\theta}_t}(\mathbf{x}_i))$ is the correct label. Conversely, Eq. (9) also implies that when comparing the functional updates of $g_{\boldsymbol{\theta}}(\mathbf{x})$ to those of $\sigma(g_{\boldsymbol{\theta}}(\mathbf{x}))$ across inputs $\mathbf{x} \in \mathcal{X}$, updates will be relatively larger for $\mathbf{x}$ where the model is uncertain ($\sigma(g_{\boldsymbol{\theta}_t}(\mathbf{x})) \approx 1/2$)). Finally, Eq. (9) highlights that the (post-activation) tangent kernel of a model with sigmoid activation *cannot* be constant over time unless the model predictions $\sigma(g_{\boldsymbol{\theta}_t}(\mathbf{x}))$ do not change.

Appendix B. Case study 3: Towards understanding the success of weight averaging

The final interesting phenomenon we investigate is that it is sometimes possible to simply average the weights $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ obtained from two stochastic training runs of the same model, resulting in a weight-averaged model that performs no worse than the individual models [5, 31] – which has important applications in areas such as federated learning. This phenomenon is known as linear mode connectivity (LMC) and is surprising as, a priori, it is not obvious that simply *averaging the weights* of independent neural networks (instead of their predictions, as in a deep ensemble [32]), which are highly nonlinear functions of their parameters, would not greatly worsen performance. While recent work has demonstrated empirically that it is sometimes possible to weight-average an even broader class of models after permuting weights [31, 33, 34], we focus here on understanding when LMC can be achieved for two models trained from the same initialization $\boldsymbol{\theta}_0$. In particular, we are interested in [5]'s observation that LMC can emerge during training: the weights of two models $\boldsymbol{\theta}_{jT}^{t'}, j \in \{1, 2\}$, which are initialized identically and follow identical optimization routine up until checkpoint t' but receive different batch orderings and data augmentations after t', can be averaged to give an equally performant model as long as t' exceeds a so-called *stability point* t^*, which was empirically discovered to occur early in training in [5]. Interestingly, [7, Sec. 5] implicitly hint at an explanation for this phenomenon in their empirical study of tangent kernels and loss landscapes, where they found an association between the disappearance of loss barriers between solutions during training and the rate of change in $K_t^{\boldsymbol{\theta}}(\cdot, \cdot)$.

LMC and train-time transition into the lazy regime. Using the weight-averaging representation of the telescoping model, we can extend on [7]'s empirical results to show that, not only is the stabilization of the tangent kernel *associated* with lower linear loss barriers, but the transition into

a lazy regime during training – i.e. reaching a point t^* after which the model gradients no longer change – can be *sufficient* to imply LMC during training as observed in [5] under a mild assumption on the performance of the two networks' *ensemble*. To see this, let $L(f) \coloneqq \mathbb{E}_{X,Y\sim P}[\ell(f(X), Y)]$ denote the expected loss of f and recall that if $sup_{\alpha\in[0,1]} L(f_{\alpha\boldsymbol{\theta}^{t'}_{1T}+(1-\alpha)\boldsymbol{\theta}^{t'}_{2T}}) - [\alpha L(f_{\boldsymbol{\theta}^{t'}_{1T}}) + (1-\alpha)L(f_{\boldsymbol{\theta}^{t'}_{2T}})] \leq 0$ LMC is said to hold. If we assume that the ensemble $\bar{f}^{\alpha}(\mathbf{x}) \coloneqq \alpha f_{\boldsymbol{\theta}^{t'}_{1T}}(\mathbf{x}) + (1-\alpha) f_{\boldsymbol{\theta}^{t'}_{2T}}(\mathbf{x})$ performs no worse than the individual models (i.e. $L(\bar{f}^{\alpha}) \leq \alpha L(f_{\boldsymbol{\theta}^{t'}_{1T}}) + (1-\alpha)L(f_{\boldsymbol{\theta}^{t'}_{2T}})$ $\forall \alpha \in [0,1]$, as is usually the case in practice [35]), then one case in which LMC is guaranteed is if the predictions of weight-averaged model and ensemble are identical. In Appendix D.2, we show that if there exists some $t^* \in [0, t']$ after which the model gradients $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}^{t'}_{jt}}(\cdot)$ no longer change (i.e. for all $t \geq t^*$ the learned updates $\boldsymbol{\theta}^{t'}_{jt}$ lie in a convex set Θ^{stable}_{j} in which $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\cdot) \approx \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t^*}}(\cdot)$), then indeed

$$\bar{f}^{\alpha}(\mathbf{x}) \approx f_{\alpha\boldsymbol{\theta}^{t'}_{1T}+(1-\alpha)\boldsymbol{\theta}^{t'}_{2T}}(\mathbf{x}) \approx f_{\boldsymbol{\theta}_{t'}}(\mathbf{x}) + \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t^*}}(\mathbf{x})^{\top} \textstyle\sum_{t=t'+1}^{T} (\alpha \Delta\boldsymbol{\theta}^{t'}_{1t} + (1-\alpha)\Delta\boldsymbol{\theta}^{t'}_{2t}). \quad (10)$$

That is, transitioning into a lazy regime during training can imply LMC because the ensemble and weight-averaged model become near-identical! This also has as an immediate corollary that models with the same $\boldsymbol{\theta}_0$ which train fully within the lazy regime will have $t^* = 0$. Note that, when using nonlinear (final) output activation $\sigma(\cdot)$ the post-activation model gradients will generally *not* become constant during training (as we discussed in Appendix A for the sigmoid and as was shown theoretically in [8] for general nonlinearities). If, however, the *pre-activation model gradients* become constant during training and the *pre-activation ensemble* – which averages the two model's pre-activation outputs *before* applying $\sigma(\cdot)$ – performs no worse than the individual models (as is also usually the case in practice [36]), then the above also immediately implies LMC for such models.

This suggests a candidate explanation for why LMC emerged at specific points t^* in [5]. To test this, we replicate their CIFAR-10 experiment using a ResNet-20 in Fig. 4 (blue). In addition to plotting the maximal decrease in accuracy when comparing $f_{\alpha\boldsymbol{\theta}^{t'}_{1T}+(1-\alpha)\boldsymbol{\theta}^{t'}_{2T}}(\mathbf{x})$ to the weighted average of the accuracies of the original models as [5], we also plot the relative change in pre-activation gradients $\frac{||\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t'+390}}(\mathbf{x}) - \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t'}(\mathbf{x})}||_2}{||\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t'}}(\mathbf{x})||_2}$ over the next epoch (390 batches) after checkpoint t'. We find that, as hypothesized, the disappearance of the loss barrier indeed coincides with the time in training when the gradients become close to stable.

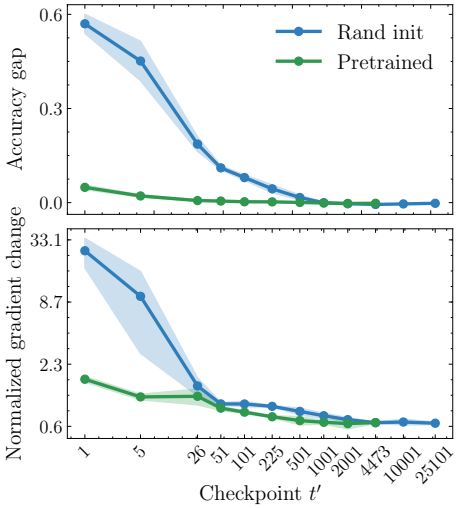

Figure 4: **LMC.** Decrease in accuracy when using averaged weights $\alpha\boldsymbol{\theta}^{t'}_{1T} + (1-\alpha)\boldsymbol{\theta}^{t'}_{2T}$ (top) and relative change in gradients within one epoch of t', by t' for randomly initialized (blue) and pre-trained ResNet-20 (green).

Pre-training and weight averaging. Because weight averaging methods have become increasingly popular when using *pre-trained* instead of randomly initialized models [37–39], we are interested in testing whether pre-training may improve mode connectibility through stabilizing the model gradients. To test this, we replicate the above experiment with the same architecture pre-trained on the SVHN dataset (in green in Fig. 4). Mimicking findings of [37], we find the loss barrier to be substantially smaller after pre-training – and can now link this to a new explanation for this phenomenon, which is that pre-training can lead to gradients stabilizing much earlier in training.

Takeaway Case Study 3. Reasoning through the learning process by telescoping out functional updates suggests that weight-averaging model parameters trained from the same checkpoint can be effective if their models' gradients are relatively stable (i.e. they train *lazily*).

Appendix C. Additional literature review

In this section, we present an extended literature review related to the phenomena we consider in Sec. 4.1 and Appendix B.

C.1. The model complexity-performance relationship (for Sec. 4.1)

Classical statistical textbooks convey a well-understood relationship between model complexity – historically captured by a model's parameter count – and prediction error: increasing model complexity is expected to modulate a transition between under- and overfitting regimes, usually represented by a U-shaped error-curve with model complexity on the x-axis in which test error first improves before it worsens as the training data can be fit too well [15, 18, 40]. While this relationship was originally believed to hold for neural networks as well [41], later work provided evidence that – when using parameter counts to measure complexity – this U-shaped relationship no longer holds [42, 43].

Double descent. Instead, the double descent [3] shape has claimed its place, which postulates that the well-known U-shape holds only in the underparameterized regime where the number of model parameters p is smaller than the number of training examples n; once we reach the interpolation threshold $p = n$ at which models have sufficient capacity to fit the training data perfectly, increasing p further into the overparametrized (or: interpolation) regime leads to test error improving again. While the double descent shape itself had been previously observed in linear regression and neural networks in [42, 44–47] (see also the historical note in [48]), the seminal paper by [3] both popularized it as a phenomenon and highlighted that the double descent shape can also occur tree-based methods. In addition to double descent as a function of the number of model parameters, the phenomenon has since been shown to emerge also in e.g. the number of training epochs[49] and sparsity [50]. Optimal regularization has been shown to mitigate double descent [51].

Due to its surprising and counterintuitive nature, the emergence of the double descent phenomenon sparked a rich theoretical literature attempting to understand it. One strand of this literature has focused on modeling double descent in the number of features in linear regression and has produced precise theoretical analyses for particular data-generating models [19, 46, 52–56]. Another strand of work has focused on deriving exact expressions of bias and variance terms as the total number of model parameters is increased in a neural network by taking into account all sources of randomness in model training [42, 57–59]. A different perspective was presented in [17], who highlighted that in the non-deep double descent experiments of [3], a subtle change in the parameter-increasing mechanism is introduced exactly at the interpolation threshold, which is what causes the second descent. [17] also demonstrated that when using a measure of the test-time *effective* parameters used by the model to measure complexity on the x-axes, the double descent shapes observed for linear regression, trees, and boosting fold back into more traditional U-shaped curves. In Sec. 4.1, we show that the telescoping model enables us to discover the same effect also in deep learning.

Benign overfitting. Closely related to the double descent phenomenon is *benign overfitting* (e.g. [19, 60–65]), i.e. the observation that, incompatible with conventional statistical wisdom about overfitting [15], models with perfect training performance can nonetheless generalize well to unseen test examples. In this literature, it is often argued in theoretical studies that overparameterized neural networks generalize well because they are much more well-behaved around unseen test examples than examples seen during training [63, 65]. In Sec. 4.1 we provide new *empirical* evidence for this by highlighting that there is a difference between $p_{\mathbf{s}}^{train}$ and $p_{\mathbf{s}}^{test}$.

Understanding modern model complexity. Many measures for model complexity capture some form of *capacity* of a hypothesis class, which gives insight into the most complex function

that *could be* learned – e.g. raw parameter counts and VC dimensions [66]. The double descent and benign overfitting phenomena prominently highlighted that complexity measures that consider only what *could* be learned and not what *is actually* learned for test examples, would be unlikely to help understand generalization in deep learning [16]. Further, [17] highlighted that many other measures for model complexity – so-called measures of effective parameters (or: degrees of freedom) including measures from the literature of smoothers [18, Ch. 3.5] as well as measures relying on the model's Hessian [67, 68] (which have been considered for use in deep learning in [69]) – were derived in the context of in-sample prediction (where train- and test inputs would be the same) and do thus not allow to distinguish differences in the behavior of learned functions on training examples from new examples. For this reason, [17] proposed an adapted effective parameter measure for smoothers that can distinguish the two, and highlighted that differentiating between the amount of smoothing performed on train- vs test examples is crucial to understanding double descent in linear regression, trees and gradient boosting. In Sec. 4.1, we show that the telescoping model makes it possible to use [17]'s effective parameter measure for neural networks, allowing interesting insight into implied differences in train- and test-time complexity of neural networks.

Grokking. Similar to double descent in the number of training epochs as observed in [49] (where the test error first improves then gets worse and then improves again during training), the *grokking* phenomenon [4] demonstrated the emergence of another type of unexpected behavior during the training run of a single model. Originally demonstrated on arithmetic tasks, the phenomenon highlights that improvements in test performance can sometimes occur long after perfect training performance has already been achieved. [21] later demonstrated that this can also occur on more standard tasks such as image classification. This phenomenon has attracted much recent attention both because it appears to challenge the common practice of early stopping during training and because it showcases further gaps in our current understanding of learning dynamics. A number of explanations for this phenomenon have been put forward recently: [70] attribute grokking to delayed learning of representations, [71] use mechanistic explanations to examine case studies of grokking, [72] attribute grokking to more efficient circuits being learned later in training, [21] attribute grokking to the effects of weight decay setting in later in training and [73] attribute grokking to the use of adaptive optimizers. [22] highlight that the latter two explanations cannot be the sole reason for grokking by constructing an experiment where grokking occurs as the weight norm grows without the use of adaptive optimizers. Instead, [22, 74] conjecture that grokking occurs as a model transitions from the lazy regime to a feature learning regime later in training. Our perspective presented in Sec. 4.1 is complementary to these lines of work: we highlight that grokking coincides with the widening of a gap in effective parameters used for training and testing examples and that there is thus a quantifiable benign overfitting effect at play.

C.2. Weight averaging in deep learning (for Appendix B)

Ensembling [75], i.e. averaging the *predictions* of multiple independent models, has long established itself as a popular strategy to improve prediction performance over using single individual models. While ensembles have historically been predominantly implemented using weak base learners like trees to form random forests [76], *deep* ensembles [32] – i.e. ensembles of neural networks – have more recently emerged as a popular strategy for improving upon the performance of a single network [32, 77]. Interestingly, deep ensembles have been shown to perform well both when averaging the predictions of the underlying models and when averaging the pre-activations of the final network layers [36].

A much more surprising empirical observation made in recent years is that, instead of averaging model predictions as in an ensemble, it is sometimes also possible to average the learned *weights*

$\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ of two trained neural networks and obtain a model that performs well [5, 78]. This is unexpected because neural networks are *highly nonlinear* functions of their weights, so it is unclear a priori when and why averaging two sets of weights would lead to a sensible model at all. When weight averaging works, it is a much more attractive solution relative to ensembling: an ensemble consisting of k models requires $k \times p$ model parameters, while a weight-averaged model requires only p parameters – making weight-averaged models both more efficient in terms of storage and at inference time. Additionally, weight averaging has interesting applications in federated learning because it could enable the merging of models trained on disjoint datasets. [78] were the first to demonstrate that weight averaging can work in the context of neural networks by showing that model weights obtained by simple averaging of multiple points along the trajectory of SGD during training – a weight-space version of the method of fast geometric ensembling [79] – could improve upon using the final solution directly.

Mode connectivity. The literature on mode connectivity first empirically demonstrated that there are simple (but nonlinear) paths of nonincreasing loss connecting different final network weights obtained from different random initializations [79–81]. As discussed in the main text, [5] then demonstrated empirically that two learned sets of weights can sometimes be *linearly* connected by simply interpolating between the learned weights, as long as two models were trained together until some stability point t^*. [82] perform an empirical study investigating which networks and optimization protocols lead to mode connectivity from initialization (i.e. $t^* = 0$) and which modifications ensure $t^* > 0$. As highlighted in Appendix B, our theoretical reasoning indicates that one sufficient condition for linear mode connectivity from initialization is that models stay in the lazy regime during training.

Methods that average weights. Beyond [78]'s stochastic weight averaging method, which averages weights from checkpoints within a single training run, weight averaging has also recently gained increased popularity in the context of averaging multiple models finetuned from the same pre-trained model [37–39]: while [37] showed that multiple models finetuned from the same pretrained model lie in the same loss basin and are linearly mode connectible, the model soups method of [38] highlighted that simply averaging the weights of multiple models fine-tuned from the same pre-trained parameters with different hyperparameters leads to performance improvements over choosing the best individual fine-tuned model. A number of methods have since been proposed that use weight-averaging of models fine-tuned from the same pretrained model for diverse purposes (e.g. [83, 84]). Our results in Appendix B complement the findings of [37] by highlighting that fine-tuning from a pre-trained model leads to better mode connectivity because the gradients of a pre-trained model remain much more stable than those trained from a random initialization.

Weight averaging after permutation matching. Most recently, a growing number of papers have investigated whether attempts to merge models through weight-averaging can be improved by first performing some kind of permutation matching that corrects for potential permutation symmetries in neural networks. [34] conjecture that all solutions learned by SGD are linearly mode connectible once permutation symmetries are corrected for. [31, 33, 85] use different methods for permutation matching and find that this improves the quality of weight-averaged models.

Appendix D. Additional theoretical results

D.1. Derivation of smoother expressions in the telescoping model (for Sec. 4.1)

Vanilla SGD. Recall that letting $\mathbf{y} = [y_1, \ldots, y_n]^\top$ and $\mathbf{f}_{\boldsymbol{\theta}_t} = [f_{\boldsymbol{\theta}_t}(\mathbf{x}_1), \ldots, f_{\boldsymbol{\theta}_t}(\mathbf{x}_n)]$, the SGD weight update with squared loss $\ell(f(\mathbf{x}), y) = \frac{1}{2}(y - f(\mathbf{x}))^2$, in the special case of single outputs $k = 1$, simplifies to $\Delta\boldsymbol{\theta}_t = \gamma_t \mathbf{T}_t(\mathbf{y} - \mathbf{f}_{\boldsymbol{\theta}_{t-1}})$. If we assume that the telescoping model holds

exactly, this implies functional updates $\Delta\tilde{f}_t(\mathbf{x}) = \gamma_t \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})^\top \mathbf{T}_t(\mathbf{y} - \tilde{\mathbf{f}}_{\boldsymbol{\theta}_{t-1}})$. If we could write $\tilde{\mathbf{f}}_{\boldsymbol{\theta}_{t-1}} = \mathbf{S}_{\boldsymbol{\theta}_{t-1}}\mathbf{y} + \mathbf{c}_{\boldsymbol{\theta}_{t-1}}$, then we would have

$$\Delta\tilde{\mathbf{f}}_t(\mathbf{x}) = \gamma_t \mathbf{T}_t^\top \mathbf{T}_t(\mathbf{y} - (\mathbf{S}_{\boldsymbol{\theta}_{t-1}}\mathbf{y} + \mathbf{c}_{\boldsymbol{\theta}_{t-1}})) = \gamma_t \mathbf{T}_t^\top \mathbf{T}_t(\mathbf{I}_n - \mathbf{S}_{\boldsymbol{\theta}_{t-1}})\mathbf{y} - \gamma_t \mathbf{T}_t^\top \mathbf{T}_t \mathbf{c}_{\boldsymbol{\theta}_{t-1}} \tag{11}$$

Noting that we must have $\mathbf{c}_{\boldsymbol{\theta}_0} = \mathbf{f}_{\boldsymbol{\theta}_0}$ and $\mathbf{S}_{\boldsymbol{\theta}_0} = \mathbf{0}^{n\times n}$, and recursively substituting Eq. (11) into Eq. (5) then allows to write the vector of training predictions as

$$\begin{aligned}\tilde{\mathbf{f}}_{\boldsymbol{\theta}_T} = &\underbrace{\left(\sum_{t=1}^{T}\left(\prod_{k=1}^{T-t}(\mathbf{I}_n - \gamma_{t+k}\mathbf{T}_{t+k}^\top \mathbf{T}_{t+k})\right)\gamma_t \mathbf{T}_t^\top \mathbf{T}_t\right)}_{\mathbf{S}_{\boldsymbol{\theta}_T}\mathbf{y}}\mathbf{y} \\ &+ \underbrace{\left(\prod_{k=0}^{T-1}(\mathbf{I}_n - \gamma_{t-k}\mathbf{T}_{t-k}^\top \mathbf{T}_{t-k})\right)\mathbf{f}_{\boldsymbol{\theta}_0}}_{\mathbf{c}_{\boldsymbol{\theta}_T}}\end{aligned} \tag{12}$$

which is a function of the training labels $\mathbf{y}$, the predictions at initialization $\mathbf{f}_{\boldsymbol{\theta}_0}$ and the model gradients $\{\mathbf{T}_t\}_{t=1}^T$ traversed during training (captured in the $n \times n$ matrix $\mathbf{S}_{\boldsymbol{\theta}_T}$ and the $n \times 1$ vector $\mathbf{c}_{\boldsymbol{\theta}_T}$) *alone*. Similarly, we can then also write the weight updates (and, by extension, the weights $\boldsymbol{\theta}_T$) using the same quantities, i.e. $\Delta\boldsymbol{\theta}_t = \gamma_t \mathbf{T}_t(\mathbf{I}_n - \mathbf{S}_{\boldsymbol{\theta}_{t-1}})\mathbf{y} - \gamma_t \mathbf{T}_t \mathbf{c}_{\boldsymbol{\theta}_{t-1}}$. By Eq. (5), this also implies that we can write predictions at *arbitrary* test input points as a function of the same quantities:

$$\tilde{f}_{\boldsymbol{\theta}_T}(\mathbf{x}) = \underbrace{\left(\sum_{t=1}^{T}\gamma_t \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})^\top \mathbf{T}_t(\mathbf{I}_n - \mathbf{S}_{\boldsymbol{\theta}_{t-1}})\right)}_{\mathbf{s}_{\boldsymbol{\theta}_T}(\mathbf{x})\mathbf{y}}\mathbf{y} + \underbrace{\left(f_{\boldsymbol{\theta}_0}(\mathbf{x}) - \sum_{t=1}^{T}\gamma_t \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})^\top \mathbf{T}_t \mathbf{c}_{\boldsymbol{\theta}_{t-1}}\right)}_{c_{\boldsymbol{\theta}_T}(\mathbf{x})}$$

where the matrix $\mathbf{S}_{\boldsymbol{\theta}_{t-1}}$ is as defined in Eq. (12), which indeed has $\mathbf{s}_{\boldsymbol{\theta}_{t-1}}(\mathbf{x}_i)$ as its i-th row (and analogously for $\mathbf{c}_{\boldsymbol{\theta}_{t-1}}$).

General optimization strategies. Adapting the previous expressions to enable the use of adaptive learning rates is straightforward and requires only inserting $\text{diag}(\frac{1}{\boldsymbol{\phi}_t})\mathbf{T}_t$ into the expression for $\Delta\tilde{f}_t(\mathbf{x})$ instead of $\mathbf{T}_t$ alone; then defining the matrices similarly proceeds by recursively unraveling updates using $\Delta\tilde{f}_t(\mathbf{x}) = \gamma_t \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})^\top \text{diag}(\frac{1}{\boldsymbol{\phi}_t})\mathbf{T}_t(\mathbf{y} - \tilde{\mathbf{f}}_{\boldsymbol{\theta}_{t-1}})$. Both momentum and weight decay lead to somewhat more tedious updates and necessitate the introduction of additional notation. Let $\Delta\mathbf{s}_t(\mathbf{x}) = \mathbf{s}_{\boldsymbol{\theta}_t}(\mathbf{x}) - \mathbf{s}_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})$, with $\mathbf{s}_{\boldsymbol{\theta}_0}(\mathbf{x}) = \mathbf{0}^{1\times n}$ and $\Delta c_t(\mathbf{x}) = c_{\boldsymbol{\theta}_t}(\mathbf{x}) - c_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})$, with $c_{\boldsymbol{\theta}_0}(\mathbf{x}) = f_{\boldsymbol{\theta}_0}(\mathbf{x})$, so that $\mathbf{s}_{\boldsymbol{\theta}_T}(\mathbf{x}) = \sum_{t=1}^T \Delta\mathbf{s}_t(\mathbf{x})$ and $\mathbf{c}_{\boldsymbol{\theta}_T}(\mathbf{x}) = f_{\boldsymbol{\theta}_0}(\mathbf{x}) + \sum_{t=1}^T \Delta\mathbf{c}_t(\mathbf{x})$. Further, we can write

$$\Delta\tilde{f}_t(\mathbf{x}) = \Delta\mathbf{s}_t(\mathbf{x})\mathbf{y} + \mathbf{c}_t(\mathbf{x}) = \gamma_t \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})^\top \mathbf{U}_t^S \mathbf{y} + \gamma_t \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})^\top \mathbf{U}_t^C \tag{13}$$

which means that to derive $\mathbf{s}_{\boldsymbol{\theta}_t}(\mathbf{x})$ for each t, we can use the weight update formulas to define the $p \times n$ update matrix $\mathbf{U}_t^S$ and the $p \times 1$ update vector $\mathbf{U}_t^C$ that can then be used to compute $\Delta\mathbf{s}_t(\mathbf{x})$ as $\gamma_t \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})^\top \mathbf{U}_t^S$ and $\Delta\mathbf{c}_t(\mathbf{x})$ as $\gamma_t \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t-1}}(\mathbf{x})^\top \mathbf{U}_t^C$. For vanilla SGD,

$$\mathbf{U}_t^S = \mathbf{T}_t(\mathbf{I}_n - \mathbf{S}_{\boldsymbol{\theta}_{t-1}}) \text{ and } \mathbf{U}_t^C = -\mathbf{T}_t \mathbf{c}_{\boldsymbol{\theta}_{t-1}} \tag{14}$$

while SGD with only adaptive learning rates uses

$$\mathbf{U}_t^S = \text{diag}(\frac{1}{\boldsymbol{\phi}_t})\mathbf{T}_t(\mathbf{I}_n - \mathbf{S}_{\boldsymbol{\theta}_{t-1}}) \text{ and } \mathbf{U}_t^C = -\text{diag}(\frac{1}{\boldsymbol{\phi}_t})\mathbf{T}_t \mathbf{c}_{\boldsymbol{\theta}_{t-1}} \tag{15}$$

Momentum, without other modifications, uses $\mathbf{U}_t^S = \frac{1}{1-\beta_1^t}\tilde{\mathbf{U}}_t^S$ and $\mathbf{U}_t^C = \frac{1}{1-\beta_1^t}\tilde{\mathbf{U}}_t^C$, where

$$\tilde{\mathbf{U}}_t^S = (1-\beta_1)\mathbf{T}_t(\mathbf{I}_n - \mathbf{S}_{\boldsymbol{\theta}_{t-1}}) + \beta_1\tilde{\mathbf{U}}_{t-1}^S \text{ and } \tilde{\mathbf{U}}_t^C = -((1-\beta_1)\mathbf{T}_t\mathbf{c}_{\boldsymbol{\theta}_{t-1}} + \beta_1\tilde{\mathbf{U}}_{t-1}^C) \tag{16}$$

with $\tilde{\mathbf{U}}_0^S = \mathbf{0}^{p\times n}$ and $\tilde{\mathbf{U}}_0^S = \mathbf{0}^{p\times 1}$.

Weight decay, without other modifications, uses

$$\mathbf{U}_t^S = \mathbf{T}_t(\mathbf{I}_n - \mathbf{S}_{\boldsymbol{\theta}_{t-1}} + \lambda\mathbf{D}_t^S) \text{ and } \mathbf{U}_t^C = -\mathbf{T}_t(\mathbf{c}_{\boldsymbol{\theta}_{t-1}} + \lambda\mathbf{D}_t^C) \tag{17}$$

where $\mathbf{D}_t^S = \gamma_{t-1}\mathbf{U}_{t-1}^S + (1-\lambda\gamma_{t-1})\mathbf{D}_{t-1}^S$ and $\mathbf{D}_t^C = \gamma_{t-1}\mathbf{U}_{t-1}^C + (1-\lambda\gamma_{t-1})\mathbf{D}_{t-1}^C$ with $\mathbf{D}_0^S = \mathbf{0}^{p\times n}$ and $\mathbf{D}_0^C = \boldsymbol{\theta}_0$.

Putting all together leads to AdamW [86] (which decouples weight decay and momentum, so that weight decay does not enter the momentum term), which uses

$$\mathbf{U}_t^S = \text{diag}(\frac{1}{\boldsymbol{\phi}_t})\frac{1}{1-\beta_1^t}\tilde{\mathbf{U}}_t^S + \lambda\mathbf{T}_t\mathbf{D}_t^S \text{ and } \mathbf{C}_t^S = \frac{1}{1-\beta_1^t}\text{diag}(\frac{1}{\boldsymbol{\phi}_t})\tilde{\mathbf{U}}_t^C + \lambda\mathbf{T}_t\mathbf{D}_t^C \tag{18}$$

where all terms are as in Eq. (16) and Eq. (17).

Remark: Writing $\tilde{\mathbf{f}}_{\boldsymbol{\theta}_T} = \mathbf{S}_{\boldsymbol{\theta}_T}\mathbf{y} + \mathbf{C}_{\boldsymbol{\theta}_T}\mathbf{f}_{\boldsymbol{\theta}_0}$ is reminiscent of a *smoother* as used in the statistics literature [18]. Prototypical smoothers issue predictions $\hat{\mathbf{y}} = \mathbf{S}\mathbf{y}$ – which include k-Nearest Neighbor regressors, kernel smoother, and (local) linear regression as prominent members –, and are usually linear in that $\mathbf{S}$ does not depend on $\mathbf{y}$. The smoother implied by the telescoping model is *not* necessarily a linear smoother because $\mathbf{S}_{\boldsymbol{\theta}_T}$ can depend on $\mathbf{y}$ through changes in gradients during training, making $\tilde{\mathbf{f}}_{\boldsymbol{\theta}_T}$ an *adaptive* smoother. This adaptivity in the implied smoother is similar to trees as recently studied in [17, 28]. In this context, effective parameters as measured by $p_{\mathbf{s}}^0$ can be interpreted as measuring how non-uniform and extreme the learned smoother weights are when issuing predictions for specific inputs [17].

D.2. Comparing predictions of ensemble and weight-averaged model after train-time transition into lazy regime (for Appendix B)

Here, we compare the predictions of the weight-averaged model $f_{\alpha\boldsymbol{\theta}_{1T}^{t'}+(1-\alpha)\boldsymbol{\theta}_{2T}^{t'}}(\mathbf{x})$ to the ensemble $\bar{f}^{\alpha}(\mathbf{x}) = \alpha f_{\boldsymbol{\theta}_{1T}^{t'}}(\mathbf{x}) + (1-\alpha)f_{\boldsymbol{\theta}_{2T}^{t'}}(\mathbf{x})$ if the models transition into a lazy regime at time $t^* \leq t'$.

We begin by noting that the assumption that the gradients no longer change after t^* (i.e. $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{jt}^{t'}}(\cdot) \approx \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t^*}}(\cdot)$ for all $t \geq t^*$) implies that the rate of change of $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t^*}}(\mathbf{x})$ in the direction of the weight updates must be approximately $\mathbf{0}$. That is, $\nabla_{\boldsymbol{\theta}}^2 f_{\boldsymbol{\theta}_{t^*}}(\mathbf{x})(\boldsymbol{\theta} - \boldsymbol{\theta}_{t^*}) \approx \mathbf{0}$ for all $\boldsymbol{\theta} \in \Theta_j^{stable}$, or equivalently all weight changes in each Θ_j^{stable} are in directions that are in the null-space of the Hessian (or in directions corresponding to diminishingly small eigenvalues). To avoid clutter in notation, we use splitting point $t' = t^*$ below, but note that the same arguments hold for $t' > t^*$.

First, we now consider rewriting the predictions of the ensemble, and note that we can now write the *second-order* Taylor approximation of each model $f_{\boldsymbol{\theta}_{jT}^{t^*}}(\mathbf{x})$ around $\boldsymbol{\theta}_{t^*}$ as

$$\begin{aligned} f_{\boldsymbol{\theta}_{jT}^{t^*}}(\mathbf{x}) = f_{\boldsymbol{\theta}^{t^*}}(\mathbf{x}) + \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t^*}}(\mathbf{x})^\top \sum_{t=t^*+1}^{T} \Delta\boldsymbol{\theta}_{jt}^{t^*} + \underbrace{\frac{1}{2}\left[\sum_{t=t^*+1}^{T} \Delta\boldsymbol{\theta}_{jt}^{t^*}\right]^\top \nabla^2_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t^*}}(\mathbf{x}) \left[\sum_{t=t^*+1}^{T} \Delta\boldsymbol{\theta}_{jt}^{t^*}\right]}_{\approx 0} \\ + R_2(\sum_{t=t^*+1}^{T} \Delta\boldsymbol{\theta}_{jt}^{t^*}) \\ \approx f_{\boldsymbol{\theta}^{t^*}}(\mathbf{x}) + \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t^*}}(\mathbf{x})^\top \sum_{t=t^*+1}^{T} \Delta\boldsymbol{\theta}_{jt}^{t^*} + R_2(\sum_{t=t^*+1}^{T} \Delta\boldsymbol{\theta}_{jt}^{t^*}) \end{aligned}$$

where $R_2(\sum_{t=t^*+1}^{T} \Delta\boldsymbol{\theta}_{jt}^{t^*})$ contains remainders of order 3 and above. Then the prediction of the ensemble can be written as

$$\begin{aligned} \bar{f}^{\alpha}(\mathbf{x}) \approx f_{\boldsymbol{\theta}_{t^*}}(\mathbf{x}) + f_{\boldsymbol{\theta}_{t^*}}(\mathbf{x})^\top \sum_{t=t^*+1}^{T} (\alpha\Delta\boldsymbol{\theta}_{1t}^{t^*} + (1-\alpha)\Delta\boldsymbol{\theta}_{2t}^{t^*}) \\ + \alpha R_2(\sum_{t=t^*+1}^{T} \Delta\boldsymbol{\theta}_{1t}^{t^*}) + (1-\alpha) R_2(\sum_{t=t^*+1}^{T} \Delta\boldsymbol{\theta}_{2t}^{t^*})) \end{aligned} \tag{19}$$

Now consider the weight-averaged model $f_{\alpha\boldsymbol{\theta}_{1T}^{t'} + (1-\alpha)\boldsymbol{\theta}_{2T}^{t'}}(\mathbf{x})$. Note that we can always write $\boldsymbol{\theta}_{jT}^{t^*} = \boldsymbol{\theta}_0 + \sum_{t=1}^{T} \Delta\boldsymbol{\theta}_{jt}^{t^*} = \boldsymbol{\theta}_{t^*} + \sum_{t=t^*+1}^{T} \Delta\boldsymbol{\theta}_{jt}^{t^*}$ and thus $\alpha\boldsymbol{\theta}_{1T}^{t^*} + (1-\alpha)\boldsymbol{\theta}_{2T}^{t^*} = \boldsymbol{\theta}_{t^*} + \sum_{t=t^*+1}^{T} \left(\alpha\Delta\boldsymbol{\theta}_{1t}^{t^*} + (1-\alpha)\Delta\boldsymbol{\theta}_{2t}^{t^*}\right)$. Further, because $\nabla^2_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t^*}}(\mathbf{x}) \sum_{t=t^*+1}^{T} \Delta\boldsymbol{\theta}_{tj}^{t^*} \approx \mathbf{0}$ for each $j \in \{0, 1\}$, we also have that

$$\nabla^2_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t^*}}(\mathbf{x}) \left(\sum_{t=t^*+1}^{T} \alpha\Delta\boldsymbol{\theta}_{t1} + (1-\alpha)\Delta\boldsymbol{\theta}_{t2}\right) \approx \alpha\mathbf{0} + (1-\alpha)\mathbf{0} = \mathbf{0} \tag{20}$$

Then, the second-order Taylor approximation of $f_{\alpha\boldsymbol{\theta}_{1T}^{t'} + (1-\alpha)\boldsymbol{\theta}_{2T}^{t'}}(\mathbf{x})$ around $\boldsymbol{\theta}_{t^*}$ gives

$$\begin{aligned} f_{\alpha\boldsymbol{\theta}_{1T}^{t'} + (1-\alpha)\boldsymbol{\theta}_{2T}^{t'}}(\mathbf{x}) \approx f_{\boldsymbol{\theta}_{t^*}}(\mathbf{x}) + \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_{t^*}}(\mathbf{x})^\top \sum_{t=t^*+1}^{T} (\alpha\Delta\boldsymbol{\theta}_{t1} + (1-\alpha)\Delta\boldsymbol{\theta}_{t2}) \\ + R_2(\sum_{t=t^*+1}^{T} \Delta\boldsymbol{\theta}_{t1} + (1-\alpha)\Delta\boldsymbol{\theta}_{t2}) \end{aligned} \tag{21}$$

Thus, $f_{\alpha\boldsymbol{\theta}_{1T}^{t'} + (1-\alpha)\boldsymbol{\theta}_{2T}^{t'}}(\mathbf{x}) \approx \bar{f}^{\alpha}(\mathbf{x})$ up to remainder terms of third order and above.

Appendix E. Additional Experimental details

In this section, we provide a complete description of the experimental details throughout this work. Each section also reports their respective required compute which was performed on either Azure VMs powered by 4 $\times$ NVIDIA A100 GPUs or an NVIDIA RTX A4000 GPU.

E.1. Case study 1 (Sec. 4.1) and approximation quality experiment (Sec. 3, Fig. 1)

Double descent experiments. In Fig. 2 (1), we replicate [3, Sec. S.3.3]'s only binary classification experiment which used fully connected ReLU networks with a single hidden layer trained using the squared loss, *without* sigmoid activation, on cat and dog images from CIFAR-10 [87]. Like [3], we grayscale and downsize images to $d = 8 \times 8$ format and use $n = 1000$ training examples and use SGD with momentum $\beta_1 = 0.95$. We use batch size 100 (resulting in $B = 10$ batches), learning rate $\gamma = 0.0025$, and test on $n_{test} = 1000$ held out examples. We train for up to $e = 30000$ epochs, but stop when training accuracy reaches 100% or when the training squared loss does not improve by more than 10^{-4} for 500 consecutive epochs (the former strategy was also employed in [3], we additionally employ the latter to detect converged networks). We report results using $\{1, 2, 5, 7, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 70, 85, 100, 200, 500, 1000, 2000, 5000\}$ hidden units. We repeat the experiment for 4 random seeds and report mean and standard errors in all figures.

In Appendix F.2, we additionally repeat this experiment with the same hyperparameters using MNIST images [88]. To create a binary classification task, we similarly train the model to distinguish 3-vs-5 from $n = 1000$ images downsampled to $d = 8 \times 8$ format and test on 1000 examples. Likely because the task is very simple, we observe no deterioration in test error in this setting for any hidden size (see Fig. 7). Because [49] found that double descent can be more apparent in the presence of label noise, we repeat this experiment while adding 20% label noise to the training data, in which case the double descent shape in test error indeed emerges. As above, we repeat both experiments for 4 random seeds and report mean and standard errors in all figures.

Compute: We train $\texttt{num\_settings} \times \texttt{num\_hidden\_sizes} \times \texttt{num\_seeds}$ $(3 \times 22 \times 4 = 264)$ models for up to $T = B \times e = 300000$ gradient steps. Training times, which included all gradient computations to create the telescoping approximation, depended on the dataset and hidden sizes, but completing a single seed for all hidden sizes for one setting took an average of 36 hours.

Grokking experiments. In panel (2) of Fig. 2, we replicate the polynomial regression experiment from [22, Sec. 5] exactly. [22] use a neural network with a single hidden layer, using custom nonlinearities, of width $n_h = 500$ in which the weights of the final layer are fixed, that is they use

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{1}{n_h} \sum_{j=1}^{n_h} \phi(\boldsymbol{\theta}_j^\top \mathbf{x}) \text{ where } \phi(h) = h + \frac{\epsilon}{2} h^2 \tag{22}$$

Inputs $x \in R^d$ are sampled from an isotropic Gaussian with variance $\frac{1}{d}$ and targets y are generated as $y(\mathbf{x}) = \frac{1}{2}(\boldsymbol{\beta}^\top \mathbf{x})^2$. In this setup, ϵ used in the activation function of the network controls how easy it is to fit the outcome function (the larger ϵ, the better aligned it is for the task at hand), which in turn controls whether grokking appears. In the main text, we present results using $\epsilon = .2$; in Appendix F.2 we additionally present results using $\epsilon = .05$ and $\epsilon = 0.5$. Like [22], we use $d = 100$, $n_{train} = 550$, $n_{test} = 500$, initialize all weights using standard normals, and train using full-batch gradient descent with $\gamma = B = 500$ on the squared loss. We repeat the experiment for 5 random seeds and report mean and standard errors in all figures.

In panel (3) of Fig. 2, we report an adapted version of [21]'s experiment reporting grokking on MNIST data. Here, we need to *adapt* [21]'s experiment into a binary classification task with lower learning rate γ to enable the use of $\tilde{f}_{\boldsymbol{\theta}_T}(\mathbf{x})$. The reduction of γ is needed here as the $\Delta\boldsymbol{\theta}_t$ are otherwise too large to obtain an accurate approximation and has as side effect that the grokking phenomenon appears visually less extreme as perfect training performance is achieved later in training. In particular, to enable the use of our model, we once more consider the binary classification task 3-vs-5 from $n = 1000$ images downsampled to $d = 8 \times 8$ features and test on 1000 held-out

examples. Like [21], we use a 3-layer fully connected ReLU network trained with squared loss (*without* sigmoid activation) and larger than usual initialization by using $\alpha\boldsymbol{\theta}_0$ instead of the default initialization $\boldsymbol{\theta}_0$. We report $\alpha = 6$ in the main text and include results with $\alpha = 5$ and $\alpha = 7$ in Appendix F.2. Like [21] we use the AdamW optimizer [86] with batches of size 200, $\beta_1 = .9$ and $\beta_2 = .99$, and use weight decay $\lambda = .1$. While [21] use learning rate 10^{-3}, we need to reduce this by factor 10 to $\gamma = 10^{-4}$ and additionally use linear learning rate warmup over the first 100 batches to ensure that weight updates are small enough to ensure the quality of the telescoping approximation; this is particularly critical because of the large initialization which otherwise results in instability in the approximation early in training. We repeat these experiments for 4 random seeds and report mean and standard errors in all figures.

Compute: Replicating [22]'s experiments required training `num_settings` $\times$ `num_seeds` $(3 \times 5 = 15)$ models for $T = 100,000$ gradient steps. Each training run including all gradient computations took less than 1 hour to complete. Replicating [21]'s experiments required training `num_settings` $\times$ `num_seeds` $(3 \times 4 = 12)$ for $T = 100,000$ gradient steps. Each training run including all gradient computations took around 5 hours to complete.

Approximation quality experiment (Fig. 1) The approximation quality experiment uses the identical MNIST setup, training process and architecture as in the grokking experiments (differing only in that we use standard initialization α and no learning rate warmup). In addition to the vanilla SGD experiment presented in the main text, we present additional settings – using momentum alone, weight decay alone, AdamW [86] and using sigmoid activation – in Appendix F.1. In particular, we use the following hyperparameter settings for the different panels:

- *"SGD":* $\lambda = 0$, $\beta_1 = 0$, no sigmoid.
- *"AdamW":* $\lambda = 0.1$, $\beta_1 = 0.9$, $\beta_2 = .99$, no sigmoid.
- *"SGD + Momentum":* $\lambda = 0$, $\beta_1 = 0.9$, no sigmoid.
- *"SGD + Weight decay":* $\lambda = 0.1$, $\beta_1 = 0$, no sigmoid.
- *"SGD +* $\sigma(\cdot)$*":* $\lambda = 0$, $\beta_1 = 0$, with sigmoid activation.

We repeat the experiment for 4 random seeds and report mean and standard errors in all figures.

Compute: Creating Fig. 5 required training `num_settings` $\times$ `num_seeds` $(5 \times 4 = 20)$ for $T = 5,000$ gradient steps. Each training run including all gradient computations took approximately 15 minutes to complete.

E.2. Case study 2 (Sec. 4.2)

In Figs. 3 and 10 we provide results on tabular benchmark datasets from [23]. We select four datasets with $>$ 20,000 examples (`houses`, `superconduct`, `california`, `house_sales`) to ensure there is sufficient hold-out data for evaluation across irregularity proportions. We apply standard preprocessing including log transformations of skewed features and target rescaling. As discussed in the main text, irregular examples are defined by first projecting each (normalized) dataset's input features onto its first principal component and then calculating each example's absolute distance to the empirical median in this space. We note that several recent works have discussed metrics of an examples irregularity or "hardness" (e.g. [89, 90]) finding the choice of metric to be highly context-dependent. Therefore we select a principal component prototypicality approach based on its simplicity and transparency. The top K irregular examples are removed from the data (these form the "irregular examples at test-time") and the remainder (the "regular examples") is split into training

and testing. We then construct test datasets containing 4000 examples, constructed from a mixture of standard test examples and irregular examples according to each proportion p.

We train both a standard neural network (while computing its telescoping approximation as described in Eq. (5)) and a gradient boosted tree model (using [91]) on the training data. We select hyperparameters by further splitting the training data to obtain a validation set of size 2000 and applying a random search consisting of 25 runs. We use the search spaces suggested in [23]. Specifically, for GBTs we consider `learning_rate` $\in$ LogNormal$[\log(0.01), \log(10)]$, `num_estimators` $\in$ LogUniformInt$[10.5, 1000.5]$, and `max_depth` $\in$ [None, 2, 3, 4, 5] with respective probabilities $[0.1, 0.1, 0.6, 0.1, 0.1]$. For the neural network, we consider `learning_rate` $\in$ LogUniform$[1e-5, 1e-2]$ and set `batch_size` $= 128$, `num_layers` $= 3$, and `hidden_dim` $= 64$ with ReLU activations throughout. Each model is then trained on the full training set with its optimal parameters and is evaluated on each of test sets corresponding to the various proportions of irregular examples. All models are trained and evaluated for 4 random seeds and we report the mean and a standard error in our results.

As discussed in the main text, we report how the relative relative mean squared error of neural network and GBT (measured as $\frac{MSE^p_{NN} - MSE^p_{GBT}}{MSE^0_{NN} - MSE^0_{GBT}}$) changes as the proportion p of irregular examples increases and relate this to changes in $\frac{\frac{1}{T}\sum_{t=1}^{T} \max_{j \in \mathcal{I}^p_{test}} ||\mathbf{k}_t(x_j)||}{\frac{1}{T}\sum_{t=1}^{T} \max_{i \in \mathcal{I}_{train}} ||\mathbf{k}_t(\mathbf{x}_i)||}$, which measures how the kernels behave at their extreme during testing relative to the maximum of the equivalent values measured for the training examples such that the test values can be interpreted relative to the kernel at train time (i.e. values ¿ 1 can be interpreted as being larger than the largest value observed across the entire training set).

Compute: The hyperparameter search results in `num_searches`×`num_datasets`×`num_models` ($25 \times 4 \times 2 = 200$) training runs and evaluations. Then the main experiment requires `num_seeds` × `num_datasets`×`num_models` ($4 \times 4 \times 2 = 32$) training runs and `num_seeds`×`num_datasets`× `num_models` × `num_proportions` ($4 \times 4 \times 2 \times 5 = 160$) evaluations. This results in a total of 232 training runs and 360 evaluations. Individual training and evaluation times depend on the model and dataset but generally require ¡ 1 hour.

E.3. Case study 3 (Appendix B)

In Fig. 4 we follow the experimental setup described in [5]. Specifically, for each model we train for a total of 63,000 iterations over batches of size 128 with stochastic gradient descent. At a predetermined set of checkpoints ($t' \in [0, 4, 25, 50, 100, 224, 500, 1000, 2000, 4472, 10000, 25100]$) we create two copies of the current state of the network and train until completion with different batch orderings, where linear mode connectivity measurements are calculated. This process sometimes also referred to as *spawning* [7] and is repeated for 3 seeds at each t'. The entire process is repeated for 3 seeds resulting in a total of $3 \times 3 = 9$ total values over which we report the mean and a standard error. Momentum is set to 0.9 and a stepwise learning rate is applied beginning at 0.1 and decreasing by a factor of 10 at iterations 32,000 and 48,000. For the ResNet-20 architecture [92], we use an implementation from [93]. Experiments are conducted on CIFAR-10 [87] where the inputs are normalized with random crops and random horizontal flips used as data augmentations.

Pretraining of the finetuned model model is performed on the SVHN dataset [94] which is also an image classification task with identically shaped input and output dimensions as CIFAR-10. We use a training setup similar to that of the CIFAR-10 model but set the number of training iterations to 30,000 and perform the stepwise decrease in learning rate at iterations 15,000 and 25,000 decaying by a factor of 5. Three models are trained following this protocol which achieve validation accuracy of 95.5%, 95.5%, and 95.4% on SVHN. We then repeat the CIFAR-10 training protocol for finetuning

but parameterize the three initialization with the respective pretrained weights rather than random initialization. We also find that a shorter finetuning period is sufficient and therefore finetune for 12,800 steps with the learning rate decaying by a factor of 5 at steps 6,400 and 9,600.

Also following the protocol of [5], for each pair of trained spawned networks ($f_{\boldsymbol{\theta}_1}$&$f_{\boldsymbol{\theta}_2}$) we consider interpolating their losses (i.e. $\ell_\alpha^{\text{avg}} := \alpha \cdot \ell(f_{\boldsymbol{\theta}_1}(\mathbf{x}), y) + (1-\alpha) \cdot \ell(f_{\boldsymbol{\theta}_2}(\mathbf{x}), y)$) and parameters (i.e. $\ell_\alpha^{\text{lmc}} := \ell(f_{\boldsymbol{\theta}^{\text{lmc}}}(\mathbf{x}), y)$ where $\boldsymbol{\theta}^{\text{lmc}} = \alpha\boldsymbol{\theta}_1 + (1-\alpha)\boldsymbol{\theta}_2$) for 30 equally spaced values of $\alpha \in [0, 1]$. In the upper panel of Fig. 4 we plot the accuracy gap at each checkpoint t' (i.e. the point from which two identical copies of the model are made and independently trained to completion) which is simply defined as the average final validation accuracy of the two individual child models minus the final validation accuracy of the weight averaged version of these two child models. Beyond the original experiment, we also wish to evaluate how the gradients $\nabla f_{\boldsymbol{\theta}_t}(\cdot)$ evolve throughout training. Therefore, in the lower panel of Fig. 4, at each checkpoint we also measure the mean squared difference in the gradient vectors lagged between the current iteration t and those at the next epoch $t + 390$ over a set of $n = 256$ test examples, $\frac{1}{n}\sum_{i=1}^{n} \frac{||\nabla f_{\boldsymbol{\theta}_t}(\mathbf{x}_i) - \nabla f_{\boldsymbol{\theta}_{t+390}}(\mathbf{x}_i)||_2}{||\nabla f_{\boldsymbol{\theta}_t}(\mathbf{x}_i)||_2}$.

Compute: We train `num_outer_seeds` × `num_inner_seeds` × `num_child_models` × `num_checkpoints` ($3 \times 3 \times 2 \times 12 = 216$) networks for the randomly initialized model. For the finetuned model this results in $3 \times 3 \times 2 \times 10 = 180$ training runs. Additionally, we require the pertaining of the 3 base models on SVHN. Combined this results in a total of $216 + 180 + 3 = 399$ training runs. Training each ResNet-20 on CIFAR-10 required ¡1 hour including additional gradient computations.

E.4. Data licenses

All image experiments are performed on CIFAR-10 [87], MNIST [88], or SVHN [94]. Tabular experiments are run on `houses`, `superconduct`, `california`, and `house_sales` from OpenML [95] as described in [23]. CIFAR-10 is released with an MIT license. MNIST is released with a Creative Commons Attribution-Share Alike 3.0 license. SVHN is released with a CC0:Public Domain license. OpenML datasets are released with a 3-Clause BSD License. All the datasets used in this work are publicly available.

Appendix F. Additional results

F.1. Additional results on approximation quality (supplementing Fig. 1)

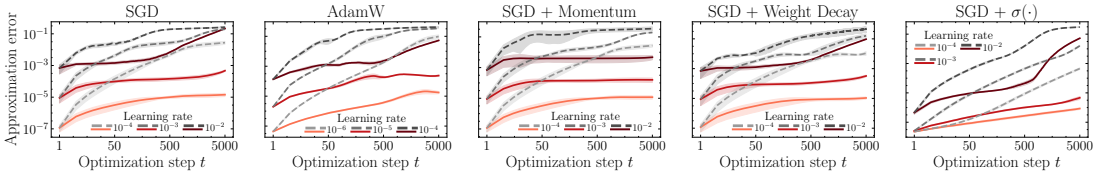

Figure 5: **Approximation error** of the telescoping ($\tilde{f}_{\boldsymbol{\theta}_t}(\mathbf{x})$, red) and the lazy model ($f^{lazy}_{\boldsymbol{\theta}_t}(\mathbf{x})$, gray) by optimization step for different optimization strategies and other design choices. Iteratively telescoping out the updates using $\tilde{f}_{\boldsymbol{\theta}_t}(\mathbf{x})$ improves upon the lazy approximation around the initialization by orders of magnitude.

In Fig. 5, we present results investigating the evolution of approximation errors of the telescoping and lazy approximation during training using additional configurations compared to the results presented in Fig. 1 in the main text (replicated in the first column of Fig. 5). We observe the same

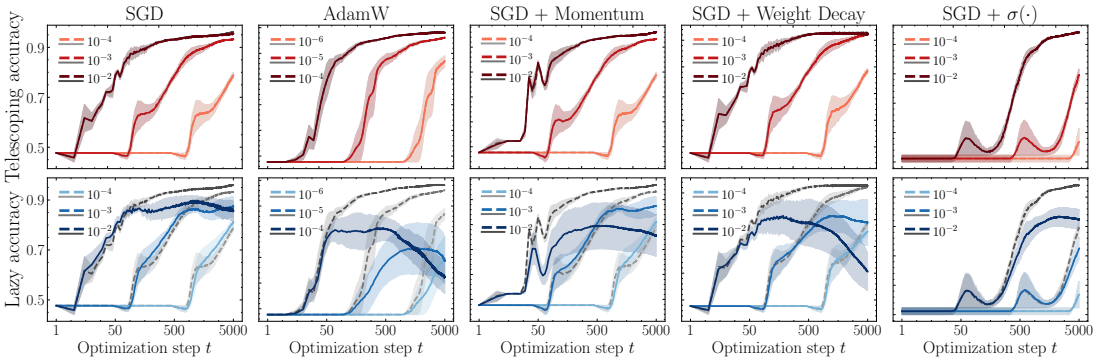

Figure 6: **Test accuracy** of the telescoping ($\tilde{f}_{\boldsymbol{\theta}_t}(\mathbf{x})$, red, top row) and the lazy model ($f^{lazy}_{\boldsymbol{\theta}_t}(\mathbf{x})$, blue, bottom row) against accuracy of the actual neural network (gray) by optimization step for different optimization strategies and other design choices. While the telescoping model visibly matches the accuracy of the actual neural network, the lazy approximation around the initialization leads to substantial differences in accuracy later in training.

trends as in the main text, where the telescoping approximation matches the predictions by the neural network by orders of magnitudes better than the lazy approximation around the initialization. Importantly, we highlight in Fig. 6 that this is also reflected in how well each approximation matches the accuracy of the predictions of the real neural network: while the small errors of the telescoping model lead to no visible differences in accuracy compared to the real neural network, using the Taylor expansion around the initialization leads to significantly different accuracy later in training. Finally, note that the learning rate γ *interacts* with the optimizer choice – e.g. Adam(W) [30, 86] naturally makes larger updates due rescaling (see Appendix A) and therefore requires smaller γ to ensure approximation quality than SGD.

F.2. Additional results for case study 1: Exploring surprising generalization curves and benign overfitting

Double descent on MNIST. In Fig. 7, we replicate the CIFAR-10 experiment from the main text while training models to distinguish 3-vs-5 on MNIST. We find that in the absence of label noise, no problematic overfitting occurs for any hidden size; both train and test error monotonically improve with increased width. Only when we add label noise to the training data, do we observe the characteristic double descent behavior in error – this is in line with [49]'s observation that double descent can be more pronounced when there is noise in the data. Importantly, we observe that as in the main text, the improvement of test error past the interpolation threshold is associated with the divergence of effective parameters used on train and test data.

Additional grokking results. In Fig. 8, we replicate the polynomial grokking results of [22] with additional values of ϵ. Like [22], we observe that larger values of $\epsilon = 0.5$ lead to less delayed generalization. This is reflected in a gap between effective parameters on test and train emerging earlier. With very small $\epsilon = .05$, conversely, we even observe a double descent-like phenomenon where test error first worsens before it improves later in training. This is reflected also in the effective parameters, where $p^{test}_{\hat{\mathbf{S}}}$ first exceeds $p^{train}_{\hat{\mathbf{S}}}$ before dropping below it as benign overfitting sets in later in training. In Fig. 9, we replicate the MNIST results with additional values of α; like [21] we observe that grokking behavior is more extreme for larger α. This is indeed also reflected in the gap between $p^{test}_{\hat{\mathbf{S}}}$ and $p^{train}_{\hat{\mathbf{S}}}$ emerging later in training.

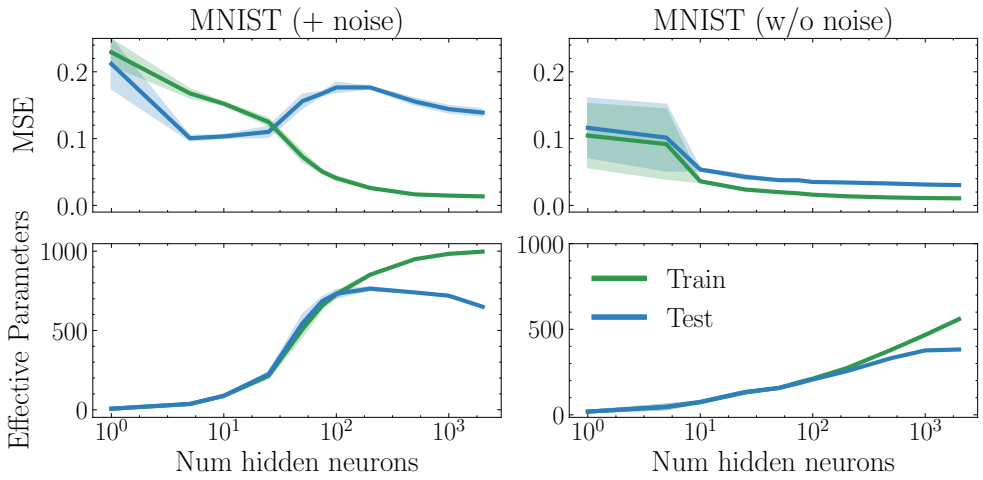

Figure 7: **Double descent** experiments using MNIST, distinguishing 3-vs-5, with 20% added label noise during training (left) and no added label noise (right). Without label noise, there is no double descent in error on this task; when label noise is added we observe the prototypical double descent shape in test error.

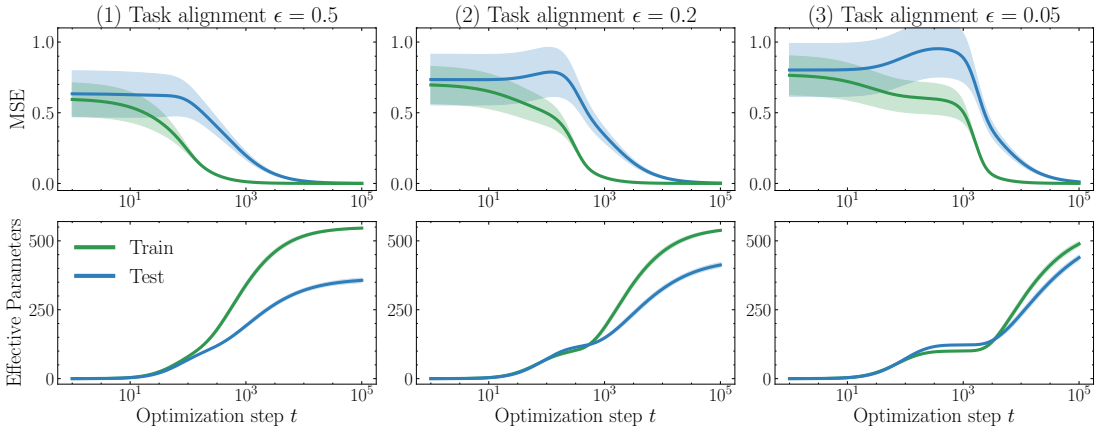

Figure 8: **Grokking** in mean squared error (top) on a polynomial regression task (replicated from [22]) against effective parameters (bottom) with different task alignment parameters ϵ.

F.3. Additional results for Case study 2: Understanding differences between gradient boosting and neural networks

In Fig. 10, we replicate the experiment from Sec. 4.2 on three further datasets from [23]'s tabular benchmark. We find that the results match the trends present in Fig. 3 in the main text: the neural network is outperformed by the GBTs already at baseline, and the performance gap grows as the test dataset becomes increasingly more irregular. The growth in the gap is tracked by the behavior of the normalized maximum kernel weight norm of the neural network's kernel. Only on the `california` dataset do we observe a slightly different behavior of the neural network's kernel: unlike the other three datasets, $\frac{\frac{1}{T}\sum_{t=1}^{T}\max_{j\in\mathcal{I}_{test}^{p}}||\mathbf{k}_t(x_j)||_2}{\frac{1}{T}\sum_{t=1}^{T}\max_{i\in\mathcal{I}_{train}}||\mathbf{k}_t(\mathbf{x}_i)||_2}$ stays substantially below 1 at all p; this indicates that there may have been examples in the training set that are irregular in ways not captured by our

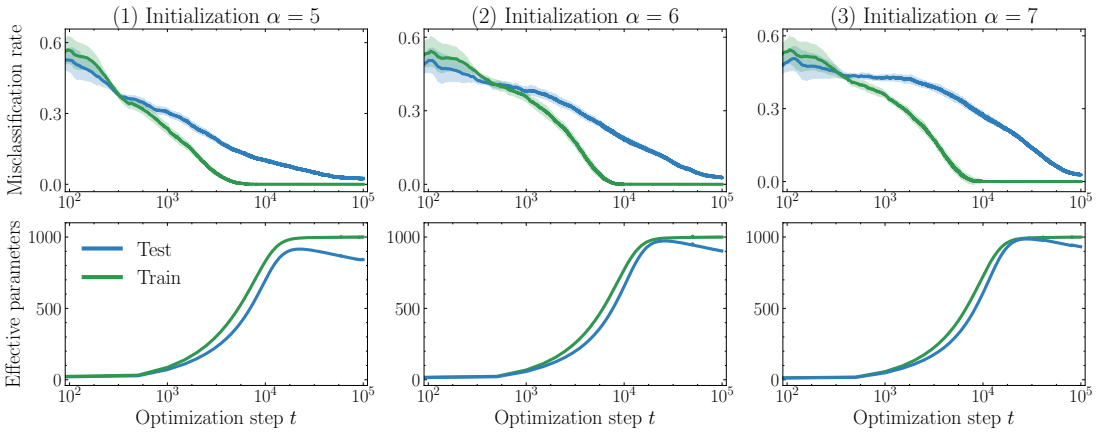

Figure 9: **Grokking** in misclassification error on MNIST using a network with large initialization (replicated from [21]) (top), against effective parameters (bottom) with different initialization scales α.

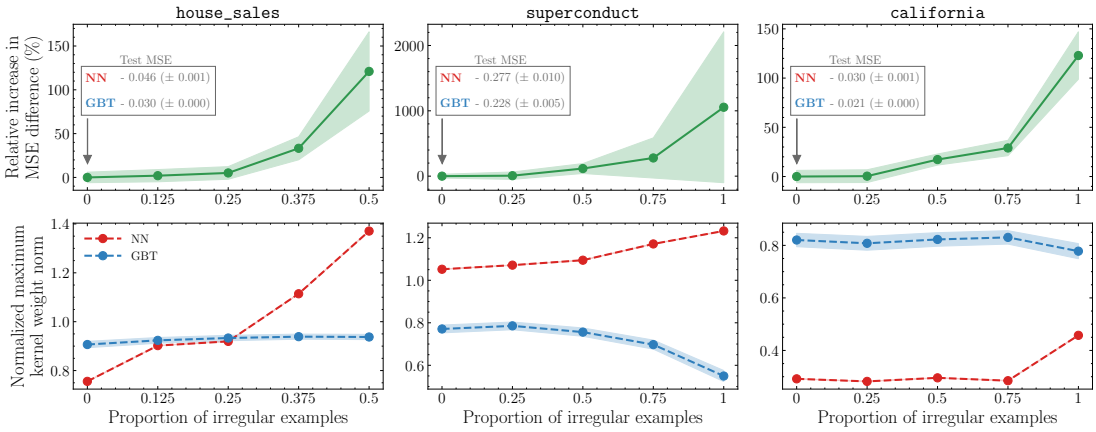

Figure 10: **Neural Networks vs GBTs:** Relative performance (top) and behavior of kernels (bottom) with increasing test data irregularity for three additional datasets.

experimental protocol. Nonetheless, we observe the same trend that $\frac{\frac{1}{T}\sum_{t=1}^{T}\max_{j\in\mathcal{I}^p_{test}}||\mathbf{k}_t(x_j)||_2}{\frac{1}{T}\sum_{t=1}^{T}\max_{i\in\mathcal{I}_{train}}||\mathbf{k}_t(\mathbf{x}_i)||_2}$ increases in relative terms as p increases.