Scaling Collapse Reveals Universal Dynamics in Compute-Optimally Trained Neural Networks

Shikai Qiu* SQ2129@NYU.EDU
New York University

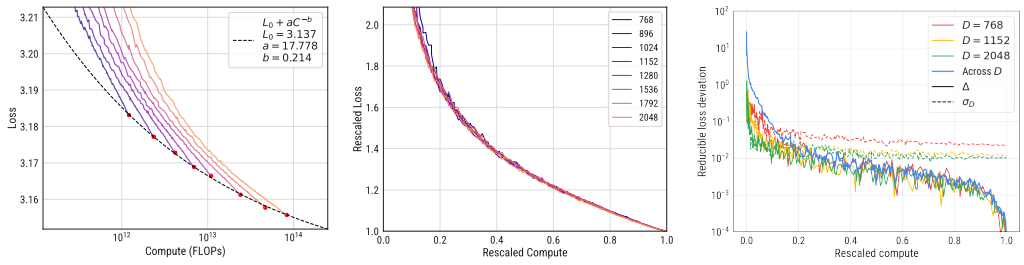
Atish Agarwala THETISH@GOOGLE.COM
Google DeepMind

Jeffrey Pennington JPENNIN@GOOGLE.COM
Google DeepMind

Lechao Xiao XLC@GOOGLE.COM
Google DeepMind

Abstract

Studies of scaling ladders have shown that the compute-optimal Pareto frontier of a family of loss curves can have a predictable shape, often a power law. We use a series of small transformer models to demonstrate that the full loss curves themselves have a consistent shape —— collapsing onto a single universal curve after an affine rescaling. Surprisingly, the deviations in the rescaled curves across model sizes are smaller than deviations induced by randomn initialization and data ordering in the raw loss curves, a phenomenon we call *supercollapse*. We recreate this phenomenon in a simplified setting of training MLPs on a synthetic regression dataset. By analyzing both the original model and our simplified model, we identify necessary conditions for supercollapse, including compute-optimal training, learning rate decay, and a power-law compute-loss Pareto frontier, and demonstrate its sensitivity to the estimate of the irreducible loss. Our study hints at a broader, dynamical universality induced by compute-optimal scaling procedures.



(a) Raw loss curves (b) Supercollapse (c) Precision

Figure 1: **Supercollapse of transformer loss curves for predicting next-pixel on CIFAR-5M.** (a) Raw cross-entropy loss curves for a series of compute-optimally scaled transformers with linearly decayed learning rate. (b) The loss curves collapse onto a single universal curve with high precision after an affine rescaling, a phenomenon we call *supercollapse*. (c) Deviation from the universal curve (Δ, blue) is within 1% for the majority of training and below the noise floor due to random initialization and data ordering measured on an equivalent scale (dashed red, yellow, and green).

* Work done during an internship at Google DeepMind.

1. Introduction

Recent progress in machine learning has been driven by understanding how the model performance scales with its parameter count, training data, and compute budget, especially in high-data settings. These empirical observations are captured by neural scaling laws — most notably the *compute-optimal* scaling laws, which predict the minimal loss achievable for a given compute budget [4, 6, 8]. While most explorations of scaling laws focus on the final loss value, in this work we show that the *full loss curve*, not just the final loss, can follow a highly predictable trend as we scale model, data, and training compute. Our main results are as follows:

- We use a next-pixel prediction transformer model trained on CIFAR-5M [10] to show there is an affine transformation of loss curves at different model sizes that makes them all coincide for a large fraction of training (Figure 1b). We dub this phenomenon *supercollapse* because the deviation between the resulting rescaled loss curves is *less* than the inherent random fluctuation in the loss curves for any particular model by varying seeds (Figure 1c).
- We show that supercollapse requires compute-optimal training horizons and learning rate decay, and is highly sensitive to the estimate of the irreducible loss (Figure 2).
- We recreate supercollapse in MLPs on an synthetic regression dataset (Figure 3) with a power-law Fourier spectrum, showing it is a general phenomenon not limited to transformers or modeling natural data.

Our work suggests that there may be *universal training dynamics* of models trained with compute-optimal hyperparameter choices. Moreover, much of the phenomenology may hold generally and doesn't require specific datasets or industry-scale models to explore. From a practical perspective, our result suggests analyzing loss curve collapse may be a cost-efficient and high-precision tool for studying certain scaling behaviors of neural networks.

2. Experimentally Tractable Scaling Ladders via Next-Pixel Prediction

To study compute-optimal training dynamics, we need an experimentally tractable *scaling ladder* — a procedure for training a family of models with increasing compute, with hyperparameters that ensure compute-optimal performance. We introduce such a scaling ladder for a family of small transformers on a next-pixel prediction task, which enables quick experimental turnaround. Small scale proxies for behavior in larger transformer models have been used successfully to understand large-scale training instabilities [13], suggesting that some findings can generalize to larger systems.

Model and Dataset Details. We train decoder-only transformer models on a next-pixel prediction task using the CIFAR-5M dataset [10] of 6 million CIFAR-like images. We convert the $32 \times 32 \times 3$ images to greyscale and flatten them into sequences of length 1024. The model autoregressively predicts the pixel intensities by minimizing the cross-entropy loss. On the larger dataset YFCC-100M [12], this task has been found to generate power-law scaling of the loss on the compute-optimal Pareto frontier [4] — a key property in neural scaling laws, especially those found in language.

Scaling Ladder. We used 3 transformer blocks and scaled the model dimension D from 768 to 2048, fixing the attention head dimension at 64 and MLP hidden dimension equal to D instead of the usual $4D$ to reduce parameters. The resulting model sizes ranged from 12M to 79M parameters,

which are sufficiently small to be trained compute-optimally on CIFAR-5M without a noticable train-test gap. We trained with the Adam [2] optimizer at a batch size of 256 images for a width-dependent *training horizon* of $S(D)$ steps, linearly decaying the learning rate to 0 at the end. All models were parameterized in μP [14, 16] to use suitably scaled initialization and learning rates (Appendix A). We trained with qk-layernorm [5], gradient clipping at global norm of 1, and a linear warmup of 1000 steps, with no weight decay. With a fixed batch size, a P parameter transformer model trained for S steps uses a compute budget (FLOPs) proportional to $6SP$ for large P [8]. Therefore we take $C = 6SP$ to be the "compute cost" for all our analyses. Note that $P \propto D^2$.

We chose the training horizon by estimating its compute-optimal value $S^*(D)$, which we will show is critical for supercollapse. To estimate the compute-optimal $S^*(D)$, we carry out the following procedure detailed in Appendix A: 1) We trained models with varying D for $7 \cdot 10^5$ steps *without* learning rate decay. 2) We then numerically computed the compute-loss Pareto frontier to obtain an estimate of $D^*(C)$ - the optimal width for a fixed compute budget. 3) Finally, we fit a power law $D^*(C) \propto C^{\alpha_D}$ and inverted it to find an optimal number of steps $S^*(D)$. We then used $S^*(D)$ as our training horizon function for our experiments with learning rate decay. This procedure assumes $S^*(D)$ does not change too much when adding learning rate decay, in order to avoid an additional sweep over S^* for each D as done in Hoffmann et al. [6].

The loss curves of the models trained with learning rate decay generate a Pareto frontier which is well-described by the form $L_0 + aC^{-b}$ (Figure 1a), as expected from prior works on compute-optimal scaling laws [6, 8]. We refer to L_0 as the *irreducible loss*. Moreover, the loss curves have remarkably similar shapes, which we explore in the next section. We found the train-test gap to be negligible and report the test loss throughout this work.

3. Observation of Supercollapse in Transformer Models

3.1. Loss Rescaling and Loss Curve Collapse

A natural way to compare the loss curves for different D is to plot the *relative progress* of training. We chose to plot the *reducible loss* vs compute (training FLOPS) as fractions of their final values. More formally, given an estimate of the irreducible loss L_0 our family of *rescaled loss curves* $\mathcal{L}(t; D)$ is derived from the original loss curves $L(C; D)$ as

$$\mathcal{L}(t; D) = \frac{L(t \cdot C^*(D); D) - L_0}{L(C^*(D); D) - L_0}, \tag{1}$$

where $C^*(D)$ is the total compute used to train a model of width D. Note that $\mathcal{L}(1; D) = 1$ always. We see that our family of rescaled curves $\mathcal{L}(t; D)$ appear to have the same shape for the last 80% of training (Figure 1b). We optimized the single parameter $L_0 = 3.137$ to achieve this degree of collapse across 8 curves over a broad range of t. Note that L_0 derived in this manner differs from the more common technique of fitting the Pareto frontier by only 0.005, but this small difference is key to the precision of the collapse as we discuss in Section 4.

This result suggests that merely predicting the endpoints of training is a low bar; in the right setup and with the appropriate rescaling, the *entire loss trajectory* is consistent across model scales. In the remainder of the paper we quantify the degree of consistency, and explore necessary conditions for this phenomenon.

3.2. Supercollapse: Consistency Below the Noise Floor

We dub the above phenomenon *supercollapse*, based on the following analysis of the deviations in $\mathcal{L}(t; D)$ across D for fixed t. One natural comparison of the deviations in $\mathcal{L}(t; D)$ are the deviations in the loss curves for fixed D induced by the randomness of the training process. We measure the *cross-seed fluctuation* σ_D defined as

$$\sigma_D^2(t) = \mathbb{V}\left\{\frac{L(tC^*(D), D) - L_0]}{\mathbb{E}[L(C^*(D), D) - L_0]}\right\} \tag{2}$$

where $\mathbb{E}$ and $\mathbb{V}$ denotes expectation and variance due to randomness in the training algorithm such as initialization and data ordering. That is, $\sigma_D(t)$ is the standard deviation of the reducible loss curves of a width D model as a fraction of its average final reducible loss. We use the estimate of L_0 found by the supercollapse fit, and perform the averaging over five seeds. We will compare $\sigma_D(t)$ to the *cross-model collapse deviation* $\Delta(t)$, defined as the standard deviation of the rescaled loss curves $\mathcal{L}(t; D)$ over the range of D we tested:

$$\Delta^2(t) = \mathbb{V}_D[\mathcal{L}(t; D)] = \mathbb{V}_D\left[\frac{L(tC^*(D), D) - L_0}{L(C^*(D), D) - L_0}\right] \tag{3}$$

which quantifies the degree of the collapse, where $\mathbb{V}_D$ denotes empirical variance over D in our experiments, and we use only *a single seed* for each D to mimic the realistic training setup.

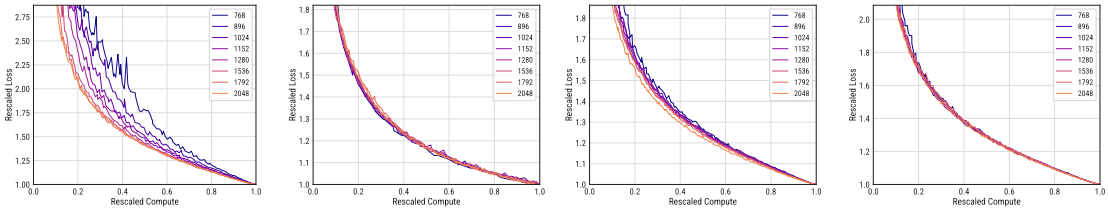
Plotting the two measures (Figure 1c) reveals that the cross-model collapse deviation (Δ, blue) is consistently *less than* the cross-seed fluctuation $\sigma_D(t)$ (dashed red, yellow, and green, computed over 5 seeds), for the majority of the training process and a variety of D. In other words, the collapse of $\mathcal{L}$ across model sizes D is quantitatively better than the *noise floor* introduced by the stochasticity of the training algorithm measuerd on an equivalent scale.

This phenomenon is remarkable; there is a way to make full loss curves across different training scales more consistent than the level of random fluctuations of any individual loss curve. While we can only predict the reducible loss curve of any one model to a relative error of 1%, we can rescale these curves so they match each other to within a relative error significantly below 1% after an initial transient period.

A key to this result is the fact that $\mathcal{L}$, and therefore $\Delta(t)$, depends on the exact final loss reached by each individual trajectory, not its average. In contrast, $\sigma_D(t)$ is rescaled by the *average* final loss across seeds. This shows that the end point of a specific loss curve efficiently encodes much of the randomness throughout the trajectory. Further evidence can be found by comparing cross-seed fluctuation $\sigma_D(t)$ with the *cross-seed collapse deviation* $\Delta_D(t)$ given by $\Delta_D^2(t) = \mathbb{V}[\mathcal{L}(t; D)]$. This measure is the analogue of the cross-model collapse deviation $\Delta(t)$ except with the "collapse" taken over random seeds while fixing D. We see that Δ_D is in fact numerically comparable to Δ (Figure 1c, solid red, yellow, and green), suggesting that the cross-model collapse and cross-seed collapse are closely related phenomona.

4. Conditions for Supercollapse

In this section, we empirically probe the conditions required for supercollapse. We first demonstrate that small differences in the scaling ladder and rescaling transform can significantly change the quality of the collapse, and then test whether supercollapse happens in other settings beyond our transformers expeirments.



(a) Not compute-optimal (b) No LR decay (c) Suboptimal choice of L_0 (d) Supercollapse

Figure 2: **Ablating necessary conditions for supercollapse.** (a) Compute-optimal horizon: underestimating the optimal compute exponent by 15% breaks scaling collapse. (b) Learning rate decay: Using a constant learning rate without decay yields only a noisy scaling collapse (no supercollapse). (c) Estimating L_0: fitting L_0 using only the final losses leads to a poor collapse. (d) Supercollapse with all conditions satisfied, shown for comparison.

4.1. Necessary Conditions for Supercollapse in Next-Pixel Prediction

Compute-Optimal Training. Suboptimally scaling the training horizon by underestimating the scaling exponent c in $S^*(D) \propto D^c$ from 2.31 to 2 breaks the collapse (Figure 2a). Note $c = 2$ corresponds to scaling parameter count in proportion to training examples, which is found to be compute-optimal in language modeling [6] and thus a reasonable alternative scaling. This result suggests that loss curve collapse can be used to test the optimality of a scaling ladder.

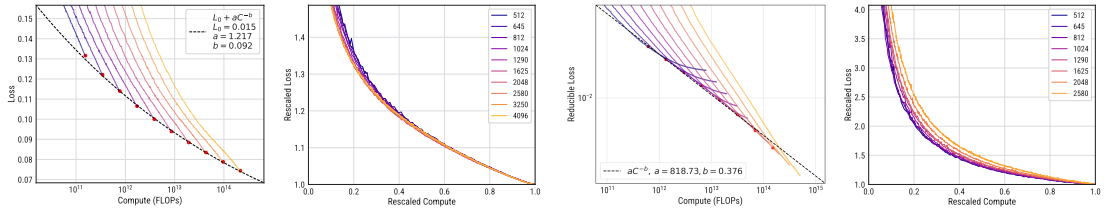
Learning Rate Decay. The variation within and between the rescaled curves at late times is noticeably larger when using a constant instead of decaying learning rate (Figure 2b, as compared to Figure 1b). This suggests that learning rate decay is essential for supercollapse.

Estimating L_0 from Full Loss Curves. When the models achieve near irreducible loss, the rescaled curves are highly sensitive to the estimate of L_0. A standard approach to estimate L_0 is by simplying fitting a power-law plus constant form $L^*(C) = aC^{-b} + L_0$ to the compute-optimal loss frontier $L^*(C) = \min_D L(C; D)$ [4, 8]. Compared to our earlier estimate of L_0 using full loss curves based on the collapse, this estimate leads to significantly worse collapse (Figure 2c), even though the estimates only differ by 0.005 (power-law fit $L_0 = 3.132$ vs. full curve $L_0 = 3.137$). However, as we do not know the true L_0, we cannot conclude which estimate is more accurate.

4.2. How General is Supercollapse?

Is supercollapse specific to the next-pixel prediction transformer models? We show the answer is no, by recreating supercollapse in MLPs trained on a synthetic regression dataset.

Supercollapse in MLPs on Random Fourier Features. We train 6-layer MLPs with varying widths from 512 to 4096 on a target function $\phi(x) = \sum_{i=1}^{M} w_i \sqrt{2} \cos\left(2\pi k_i^\top x + b_i\right)$, with $x \in \mathbb{R}^8, w_i \sim \mathcal{N}(0,1), b_i \sim \frac{\pi}{2}\text{Bernoulli}(0.5)$, $k_i = \text{quantize}(|k_i| v_i)$ where $|k_i|$ is a scalar sampled from a power law with support $[1, 10^6]$ and exponent -2, v_i is a random unit vector, and quantize rounds to the nearest point in $\mathbb{Z}^8$. During training, x is sampled uniformly from $[-0.5, 0.5]^8$, making the fourier featrues orthonormal over the data distribution. We include experiment details in Appendix B. Carrying out the experimental procedure in Section 2, we find qualitatively the same supercollapse shown in Figure 3a and Figure 3b in this setup. This result indicates supercollapse may be a fairly generic phenomenon in scaling neural networks.



(a) Loss (b) Supercollapse (c) No power law (d) No Collapse

Figure 3: **Supercollapse in MLPs trained on random Fourier featurues data.** (a,b) Raw loss and rescaled loss cureves with a linearly decaying learning rate, showing supercollapse. (c,d) Raw loss and rescaled loss cureves with a constant learning rate but with an expoential instead of power-law frequency distribution, showing a lack of collapse.

Power-Law Compute-Loss Frontier. Supercollapse is a statement about a precise scale invariance in the loss dynamics. This suggests that the task itself needs to have some scale invariance as well, such as a power-law compute-loss Pareto frontier. We ablate this property in the MLP setting by changing the distribution of the scalar frequency $|k_i|$ from a power law to an exponential. With this change, the compute-loss frontier no longer obeys a power law as shown in Figure 3c, where the best power-law fit is shown as a dashed line and L_0 is estimated to be 0. The lack of a power-law pareto frontier is in line with recent works establishing a connection between the power-law spectrum of the task kernel and the power-law scaling of loss with compute [1, 11]. In Figure 3d, we see this change also destroys the collapse in the rescaled curves, suggesting that a power-law compute-optimal frontier is necessary.

Effect of Feature Learning. In Appendix C, we discuss the effect of feature learning on supercollapse. We train versions of the transformer and MLP models with no or weak feature learning (curvature evolution), and find that the collapse of the rescaled loss curves look qualitatively different than those in Figure 1b and Figure 3. This suggests that the amount of feature-learning can impact the shape and quality of the collapse significantly, an interesting topic for further study.

5. Discussion

Going beyond the existing studies on scaling laws that focus primarily on the final loss, our result points to a high degree of universality in the overall training dynamics of neural networks as they scale. This dynamical universality makes it possible to study the scaling behaviors of the entire loss curves after a rescaling, with equal or even better precision compared to the final loss. This finding potentially enables a more efficient and precise method for understanding and predicting neural network performance as they scale.
We list several exicting extensions of this work:

- *Scaling supercollapse*: Can we produce supercollapse in much larger-scale models with more sophisticated and realistic scaling ladders (e.g. co-scaling width, depth, and batch size)?
- *Explaining supercollapse*: Can we understand why the loss curves at different scales collapse onto a single universal curve with unexpectedly high consistency?
- *Guiding optimal scaling*: Can we use supercollapse, or lack thereof, to identify sub-optimality in our scaling ladder other than the training horizon?

References

[1] Blake Bordelon, Alexander Atanasov, and Cengiz Pehlevan. A dynamical model of neural scaling laws. *arXiv preprint arXiv:2402.01092*, 2024.

[2] Jimmy Ba Diederik P. Kingma. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations (ICLR)*, 2015.

[3] Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs). *Preprint arXiv 1606.08415*, 2016.

[4] Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B Brown, Prafulla Dhariwal, Scott Gray, et al. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*, 2020.

[5] Alex Henry, Prudhvi Raj Dachapally, Shubham Pawar, and Yuxuan Chen. Query-key normalization for transformers. *arXiv preprint arXiv:2010.04245*, 2020.

[6] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

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

[8] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[9] 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.

[10] Preetum Nakkiran, Behnam Neyshabur, and Hanie Sedghi. The deep bootstrap framework: Good online learners are good offline generalizers. *arXiv preprint arXiv:2010.08127*, 2020.

[11] Elliot Paquette, Courtney Paquette, Lechao Xiao, and Jeffrey Pennington. 4+ 3 phases of compute-optimal neural scaling laws. *arXiv preprint arXiv:2405.15074*, 2024.

[12] Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. Yfcc100m: The new data in multimedia research. *Communications of the ACM*, 59(2):64–73, 2016.

[13] Mitchell Wortsman, Peter J Liu, Lechao Xiao, Katie Everett, Alex Alemi, Ben Adlam, John D Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, et al. Small-scale proxies for large-scale transformer training instabilities. *arXiv preprint arXiv:2309.14322*, 2023.

[14] Greg Yang and Edward J. Hu. Feature Learning in Infinite-Width Neural Networks. *International Conference on Machine Learning (ICML)*, 2021.

[15] Greg Yang and Etai Littwin. Tensor Programs IVb: Adaptive Optimization in the Infinite-Width Limit. *International Conference on Learning Representations (ICLR)*, 2023.

[16] Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[17] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

Appendix A. Transformer Experiment Details

A.1. Architectural Details

Our transformer uses GeLU activations [3], RMSNorm [17], and learned positional embeddings. We untie the embedding matrix from the output head and do not use bias anywhere. The vocabulary is the set of pixel intensities $\{0, \ldots, 255\}$.

A.2. Hyperparameters

We parameterize the learning rate for each weight matrix as $\eta = \eta_{\text{base}}/D$ where D is the model dimension, except for the embedding matrix which has $\eta = \eta_{\text{base}}$. We use a parameter multiplier a on the embedding matrix. We use $\eta_{\text{base}} = 4$ and $a = 0.1$ as they led to good performance in our early experiments. We initialize the embedding matrix as $W_{ij}^{\text{emb}} \sim \mathcal{N}(0, 1)$, the output head as $W^{\text{head}} = 0$, all other matrices W as $W_{ij} \sim \mathcal{N}(0, 1/D)$.

A.3. Estimating Compute-Optimal Training Horizon

We provide additional details for the 3-step procedure presented in Section 2:

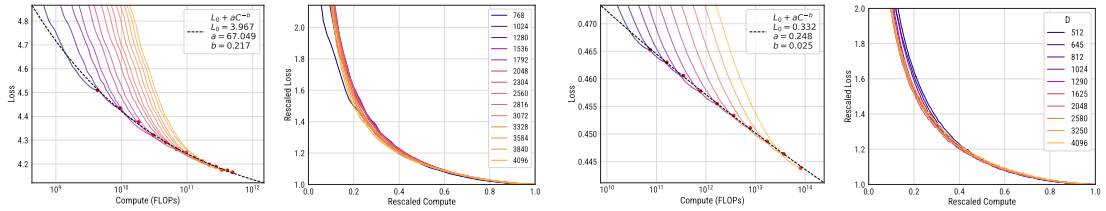
1. We trained models with varying D for $7 \cdot 10^5$ steps *without* learning rate decay but keeping the initial warmup. We chose a large enough number of steps so that the largest model could reach the compute-optimal loss frontier. We average the loss curves from 5 seeds for each D.

2. We numerically computed the compute-loss Pareto frontier to obtain an estimate of $D^*(C)$ - the optimal width for a fixed compute budget. We use 50 logarithically spaced points for C and find the D that achieves the best loss given that amount of compute.

3. We fit a power law $D^*(C) \propto C^{\alpha_D}$ and inverted it to find an optimal number of steps $S^*(D)$. Before obtaining the final fit, we iteratively remove the point with minimum or maximum C in the remaining points to maxize the R^2 of the fit. This step removes outlier points that do not belong on the true compute-optimal frontier but are present in our data due to over-training the largest model or under-training the smallest model.

Appendix B. MLP Experiment Details

We use the vanilla MLP where $f(x) = \left(W^{L+1} \circ g \circ W^L \circ g \circ \cdots \circ g \circ W^1\right)(x)$, where W^ℓ denotes linear layers without bias. We parameterize the learning rate for each weight matrix as $\eta = \eta_{\text{base}}/D$ where D is the width, except W^1 which has a learning rate $\eta_{\text{base}}/8$ (8 is the input dimension). We initialize all but the last layer as $W_{ij} \sim \mathcal{N}(0, 1/D)$ and zero-initialize the last layer. We use a base learning rate $\eta_{\text{base}} = 0.4$ and a batch size of 16384. We use gradient clipping at global norm of 1 and a linear warmup of 1000 steps. We repeat the same procedure outlined in Section 2 to estimate the compute-optimal training horizon and run our final experiments with a linear decay learning rate schedule to produce Figure 3a and Figure 3b.
For Figure 3c and Figure 3d, we sample the scalar frequencies $|k_i|$ from the distribution $p(k) \propto \exp(-k/k_0)$ where $k_0 = 1.1$. We did not experiment different values of k_0.



(a) Transformer loss (b) Transformer collapse (c) MLP loss (d) MLP collapse

Figure 4: **Loss and loss curve collapse in models with no or weak feature learning.** (a) Loss curves for transformers where only the last layer is trained, using a linearly decayed learning. (b) Rescaled loss curves. (c) Loss curves for MLPs using NTK parameterization with a linearly decayed learning. (d) Rescaled loss curves.

Appendix C. Effect of Feature Learning

In Figure 4, we show the raw and rescaled loss curves in models with no or weak feature learning, obtained by training only the last layer of the transformer (scaling learning rate as $1/D$) and training the MLP in the NTK parameteriztaion [7, 9]. As before, we train the models using the Adam optimizer. We use the version of the NTK parameteriztaion appropriate for the Adam optimizer [15]. Compared to the collapse observed in fully-trained feature-learning models, the collapse in Figure 4 show two qualitative differences: 1) the collapse is only good at late times ($t > 0.5$), and 2) the loss curve is flat, i.e. $\partial\mathcal{L}/\partial t \approx 0$, at late times, with a clear positive curvature, i.e. $\partial^2\mathcal{L}/\partial t^2 > 0$. Both phenomena are likely related to the effect of the learning rate decay on slowing down the loss decrease in linear models rather than accelerating it as in non-linear models. We leave further exploration of this topic to future work.