SIMULATING TRAINING SPIKES OF DEEP NEURAL NETWORKS BY LAYER-WISE RANDOM FEATURE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Intermittent "spikes" in loss, during the pre-training of Large Language Models (LLMs), remains a mysterious phenomenon both theoretically and empirically. In this paper, we show that simple quadratic models with heavy-tailed, power-law distributed inputs can replicate these spikes. We introduce a theoretical framework and simulation algorithm to model per-layer training dynamics of Deep Neural Networks (DNNs), using such quadratic models. In a case study on an Adam hyperparameter configuration (where β_2 is set close to 1) known to induce spikes in Transformer LMs, our simulation successfully predicts bumps indicating spiky behavior from a given checkpoint. Furthermore, we derive an appropriate β_2 schedule from observed metrics, which empirically improves performance.

1 INTRODUCTION

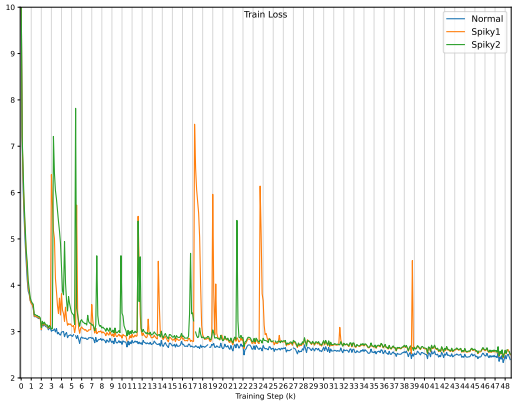
"Training spikes", the temporary, sharp surges in loss (Figure 1.1), are a phenomenon often observed during the pre-training of Large Language Models (LLMs) (Chowdhery et al., 2023; Molybog et al., 2023). Empirically, these spikes are considered pathological because the increase in loss may indicate training instability, the hyperparameters being sub-optimal, or the final model less predictive. The wild fluctuations in training metrics can trigger numerical issues as well, since large-scale Deep Neural Networks (DNNs) are usually trained with low-precision arithmetic. While their empirical occurrence is well documented, the underlying causes of these training spikes are not yet fully understood, partly because their occurrences can be affected by many factors, sometimes difficult to reliably reproduce in controlled settings (Nishida et al., 2024; Takase et al., 2025).

Stochastic optimizers, such as SGD and Adam (Kingma & Ba, 2015), inherently produce fluctuations due to the random sampling of training data. However, the "training spikes" discussed in this paper are distinct in their sensitivity to *both* data shuffling and model initialization. Typically, the trajectory of training loss fluctuations (the "up-and-downs" of the curve) remains consistent under an identical data shuffle (Figure 1.1b), irrespective of model initialization or minor architectural modifications. However, training spikes do not follow this pattern. As shown in Figure 1.1a, two training trajectories, which differ only by their random initialization seeds, can spike at different steps. This extra randomness complicates the study and mitigation of training spikes. For example, simply removing "data outliers" as a remedy to training spikes will only partially solve the problem, since one can often still observe random surges in the *validation loss* (Figure 1.1c). This suggests that training spikes are usually not caused by a specific occurrence of one single data point.

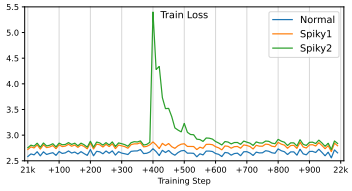
In § 3.4, we will detail our theoretical framework, which posits that normal fluctuations in loss are driven solely by data shuffling because high model dimensionality typically cancels out model-originated randomness. This concentration of measure property is conditional; it breaks down when the data distribution is heavy-tailed, resulting in training spikes. We demonstrate this mechanism using Power-Law Random Feature (PLRF) models – a class of simple linear-regression models with quadratic loss. Furthermore, we introduce an algorithm that uses PLRF-like dynamics to simulate the training behavior of individual layers within a DNN.

Remarkably, even simple models like PLRF can exhibit training spikes. Figure 2.1 compares various PLRF training trajectories – with and without spikes – across different optimizers, learning rates, and power-law exponents. Our findings highlight that the power-law exponent α, which quantifies the "heavy-tailedness" of the data distribution, is critical; a larger α is a necessary condition for

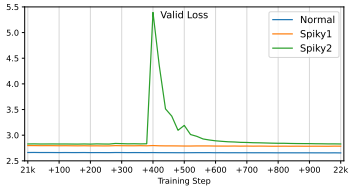
Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) Overall trajectories of the training loss.



(b) Training loss from 21k to 22k.



(c) Validation loss from 21k to 22k.

Figure 1.1: Training of a Transformer LM. Comparing a "Normal" training with two "Spiky" trainings. "Normal" and "Spiky1" differ in the β_2 hyperparameter of Adam only, while "Spiky1" and "Spiky2" differ in the random-seed of model initialization only. Training spikes can occur at different steps between "Spiky1" and "Spiky2", while "normal fluctuations" in training loss remain consistent.

these spikes to occur. To our knowledge, this is the first study to explicitly link training spikes to heavy-tailed data distributions.

Our simulation framework is largely inspired by Lee et al. (2019), utilizing a Taylor expansion of the DNN at training step t to predict dynamics up to a future step $t + s$. We advance this approach by incorporating stochastic optimization algorithms like SGD and Adam, which are more representative of real-world practices than full-batch gradient descent. This shift requires a 2nd-order Taylor expansion to account for stochastic gradient noise; Fortunately, recent work by Paquette et al. (2021; 2024) provides the theoretical tools to precisely solve training dynamics under this setting. The 2nd-order analysis is essential; as we show in Figure 4.1, empirical evidence suggests that training spikes of DNNs likely originate from 2nd-order gradient noise within some layers.

To summarize our contributions:

1. We developed a framework aligning DNN layers with simple Random Feature models to enable lightweight simulations of training dynamics (§ 3.3). This approach facilitates the scientific study of elusive phenomena like training spikes in DNNs.

2. While spiky behavior is often linked to the complex, non-convex nature of feature learning in DNNs (as seen in the "Edge-of-Stability" regime, Cohen et al., 2021; 2022), it is surprising that even convex PLRF models exhibit similar training spikes (§ 2).

3. We conducted a case-study where the Adam hyperparameter β_2 causes training spikes in a Transformer LM. Our simulation successfully predicts the spiky behavior for a given model checkpoint (§ 4). We then analyze several training metrics in details, and offer optimization recipes such as an appropriate β_2 schedule, which can avoid spikes and empirically improve performance (§ D.5).

2 Background

A Random Feature model assumes a non-trainable $\mathtt{D} \times \mathtt{V}$ matrix $\mathbf{A}$ which maps a $\mathtt{V}$-dimensional random input $\mathbf{x}$ to $\mathtt{D}$-dimensional features. Then, the features are fed to linear regression with a quadratic loss

$$\ell[\mathbf{x}; \boldsymbol{w}] \stackrel{\text{def}}{=} \frac{c}{2}\big(\boldsymbol{w}^\top \mathbf{A}\mathbf{x} - b[\mathbf{x}]\big)^2. \tag{1}$$

Here, $\boldsymbol{w}$ is the $\mathtt{D}$-dimensional trainable model weights, the label $b[\mathbf{x}]$ a scalar function of $\mathbf{x}$, and c a constant. The constant matrix $\mathbf{A}$ is usually set to random Gaussian if not else specified.

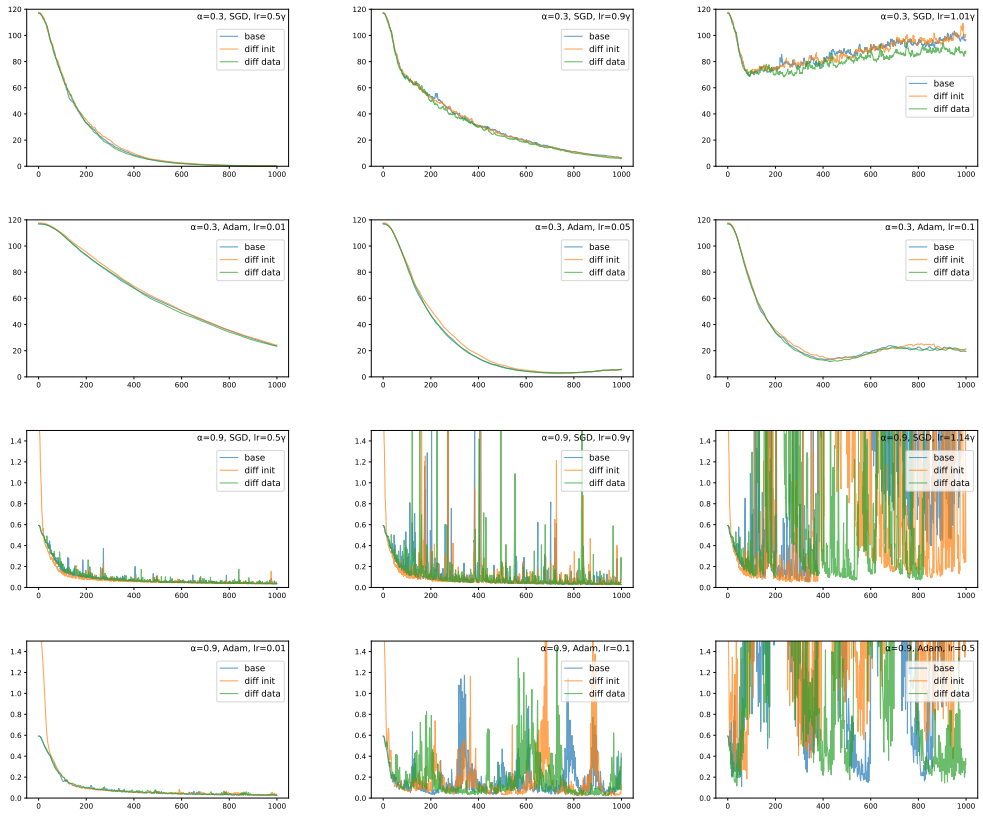


Figure 2.1: The validation loss of PLRF, from training step 0 to 1k. We compare $\alpha \in \{0.3, 0.9\}$, for SGD and Adam, with different learning-rates. Each plot shows 3 random seeds (to distinguish initialization and data-shuffle separately), and learning-rates are chosen such that training converges, nearly diverges, or diverges, respectively. For SGD, the required learning-rates are calculated from the threshold γ given in formula (2). For Adam, the learning-rates are found by manual grid search.

In addition, the Power-Law Random Feature model (Maloney et al., 2022; Bahri et al., 2024, PLRF) assumes that $\mathbf{x} = (i^{-\alpha} x_i)_{1 \le i \le \mathtt{V}}$, where each x_i is Gaussian i.i.d. drawn from $\mathcal{N}(0, 1)$, and α is the power-law exponent of input data. In this work (§ 3.2), we will use $\mathbf{A}\mathbf{x}$ to approximate hidden vectors in a Deep Neural Network (DNN), so $\mathtt{D}$ is the model dimension of the DNN, while $\mathtt{V}$ is a very large ($\approx \infty$) number intended for a latent space that can linearize the sub-DNN which produces the hidden vector, as inspired by Neural Tangent Kernel (Jacot et al., 2018; Bordelon et al., 2024).

When Stochastic Gradient Descent (SGD) is applied to the optimization of model (1), the training dynamics can be analytically solved (Paquette et al., 2021; 2024). Moreover, since the power-law data distribution in PLRF implies neural scaling laws (Maloney et al., 2022; Bahri et al., 2024), solving the dynamics allows for numerical predictions of *compute-optimal* scaling (Paquette et al., 2024). In this study, we further demonstrate that a large α can cause training spikes as well.

In Figure 2.1, we compare the training dynamics of PLRF for $\alpha \in \{0.3, 0.9\}$. We set $\mathtt{D} = 1024$, $\mathtt{V} = 20000$, $c = 1$, and apply SGD and Adam across different learning-rates. The batch-size $\mathtt{B} = 8$. When $\alpha = 0.3$, as the learning-rate increases, validation loss diverges without showing any spikes. In contrast, when $\alpha = 0.9$, both optimizers exhibit training spikes before divergence. This suggests two conditions for training spikes in PLRF: (1) heavy-tailed data (large α) and (2) near divergence (large learning-rate). We provide the Python code for these experiments in supplementary materials.

The spiky behavior of PLRF is surprising for two reasons. First, while SGD-based training is a random process influenced by data shuffling, Paquette et al. (2021) proves that the validation loss of Random Feature models typically converges to a smooth, *deterministic* curve – a phenomenon known as the "Concentration of SGD" – when $\mathtt{D}$ and $\mathtt{V}$ are sufficiently large. Random training spikes

of PLRF seemingly contradict this concentration effect – As it turns out, for PLRF with $\alpha > 0.5$, while the *expected* validation loss remains smooth, and the scaling analysis of Paquette et al. (2024); Ferbach et al. (2025) still valid, it must be noted that these results describe *average-case* behavior; But one can no longer prove the *convergence* to expectation (see § 3.4 and § B.6). In other words, at $\alpha > 0.5$, training can exhibit random spikes that temporarily deviate from the expectation, even at massive scales.

Second, while similar spiky behavior has been observed in DNNs under *full-batch gradient descent*, it was previously considered impossible for quadratic models (Cohen et al., 2021; Damian et al., 2023; Cohen et al., 2025). This is because full-batch gradient descent on quadratic loss has straightforward dynamics: It diverges if the learning rate exceeds $2/\lambda_{\text{max}}$ (where λ_{max} is the largest eigenvalue of the Hessian). Typically, only DNNs with 3rd-order terms exhibit spikes and the subsequent stabilization characteristic of the so-called "Edge-of-Stability" regime, when the learning rate nears this threshold.

However, this theory does not translate directly to stochastic optimization. Due to stochastic gradient noise captured by Hessian, training spikes can occur in quadratic models when the input data is heavy-tailed; 3rd-order terms are not necessary. Additionally, the learning-rate threshold γ of divergence is not $2/\lambda_{\text{max}}$ for SGD; instead, when the batch-size $\mathtt{B}$ is small ($\mathtt{B} \ll \sqrt{\mathtt{D}}$), γ is determined by the following equation (see Paquette et al., 2024, C.2, or § B.5 of this paper):

$$\frac{1}{\mathtt{B}} \sum_i \frac{\gamma \lambda_i}{2 - \gamma \lambda_i} = 1, \tag{2}$$

where $\{\lambda_i\}$ is the eigenvalues of the Hessian. In particular, $\gamma < \min\left(2\mathtt{B}/\sum_i \lambda_i,\, 2/\lambda_{\text{max}}\right)$.

3 Theory

Having demonstrated the rich dynamics of PLRF including training spikes, in this section we introduce a theoretical framework that aligns each DNN layer with a Random Feature model.

3.1 Layer-wise Quadratic Loss

Let $\mathcal{W}$ be the model parameters of a DNN. Given a loss function $\ell[\mathfrak{x}; \mathcal{W}]$, where $\mathfrak{x}$ is a data point, the DNN is trained by minimizing the loss over a data distribution $\mathfrak{x} \sim \mathcal{D}$. We are interested in the dynamics of the validation loss $\mathbb{E}_{\mathfrak{x}\sim\mathcal{D}}\big(\ell[\mathfrak{x}; \mathcal{W}_t]\big)$, where $\mathcal{W}_t$ is the model checkpoint at training step t.

Definition 1. Instead of directly investigating $\mathbb{E}_{\mathfrak{x}\sim\mathcal{D}}\, \ell[\mathfrak{x}; \mathcal{W}_t]$, we take its Taylor expansion to the 2nd order at training step t, for a fixed layer $\boldsymbol{w} \subset \mathcal{W}$:

$$\delta^{\boldsymbol{w}}_{t,s} \stackrel{\text{def}}{=} \mathbb{E}_{\mathfrak{x}\sim\mathcal{D}}\Big(\frac{\partial \ell}{\partial \boldsymbol{w}}[\mathfrak{x}; \mathcal{W}_t]\Big)(\boldsymbol{w}_{t+s} - \boldsymbol{w}_t) + \frac{1}{2}(\boldsymbol{w}_{t+s} - \boldsymbol{w}_t)^\top \mathbb{E}_{\mathfrak{x}\sim\mathcal{D}}\Big(\frac{\partial^2 \ell}{\partial \boldsymbol{w}^\top \partial \boldsymbol{w}}[\mathfrak{x}; \mathcal{W}_t]\Big)(\boldsymbol{w}_{t+s} - \boldsymbol{w}_t).$$

This is the approximate "amount of change" contributed by $\boldsymbol{w}$ to the validation loss, when $\boldsymbol{w}$ is trained to step $t + s$ while other parameters are fixed at step t. One subtle detail is that $\boldsymbol{w}_{t+s}$ evolves by the *original* updates calculated from the gradients of $\ell[\mathfrak{x}_{t+s}; \mathcal{W}_{t+s}]$ (where $\mathfrak{x}_{t+s} \sim \mathcal{D}$ is drawn from the training data-shuffle), without freezing any model parameters. We only freeze model parameters when calculating the evaluation metric $\delta^{\boldsymbol{w}}_{t,s}$.

The metric $\delta^{\boldsymbol{w}}_{t,s}$ is a quadratic function of $\boldsymbol{w}_{t+s}$, which can be written as "$L[\boldsymbol{w}_{t+s}] + \text{Constant}$" where

$$L[\boldsymbol{w}] \stackrel{\text{def}}{=} \frac{1}{2}(\boldsymbol{w} - \mathbf{w}^*)^\top \mathbb{E}_{\mathfrak{x}\sim\mathcal{D}}\Big(\frac{\partial^2 \ell}{\partial \boldsymbol{w}^\top \partial \boldsymbol{w}}[\mathfrak{x}; \mathcal{W}_t]\Big)(\boldsymbol{w} - \mathbf{w}^*). \tag{3}$$

Here, $\mathbf{w}^*$ is calculated such that the gradient of L can match that of the validation loss at $\boldsymbol{w}_t$:

$$\frac{\partial L}{\partial \boldsymbol{w}}[\boldsymbol{w}_t] = \mathbb{E}_{\mathfrak{x}\sim\mathcal{D}}\Big(\frac{\partial \ell}{\partial \boldsymbol{w}}[\mathfrak{x}; \mathcal{W}_t]\Big).$$

Our goal is to simulate the dynamics of L, the *layer-wise quadratic loss*. In this paper, we rigorously define "*layers*" of a DNN as below.

Definition 2. A *layer* is a subset $\boldsymbol{W} \subset \mathcal{W}$ of model parameters such that:

(1) $\boldsymbol{W}$ acts as a matrix multiplication, i.e., it takes a vector $\mathbf{h}$ as input, and returns $\boldsymbol{y} \stackrel{\text{def}}{=} \boldsymbol{W}\mathbf{h}$ as the layer output.

(2a) The forward pass to the input $\mathbf{h}$ does not depend on $\boldsymbol{W}$. In other words, $\mathbf{h}$ can be written as $\mathbf{h} = \mathbf{h}[\mathfrak{x}; \mathcal{W}^{\mathrm{F}}]$ where $\mathcal{W}^{\mathrm{F}} \cap \boldsymbol{W} = \emptyset$.

(2b) The backward pass to the output gradient $\frac{\partial \ell}{\partial \boldsymbol{y}}$ does not depend on $\boldsymbol{W}$. In other words, $\frac{\partial \ell}{\partial \boldsymbol{y}}$ can be written as $\frac{\partial \ell}{\partial \boldsymbol{y}} = \frac{\partial \ell}{\partial \boldsymbol{y}}[\mathfrak{x}; \boldsymbol{y}, \mathcal{W}^{\mathrm{B}}]$, where $\mathcal{W}^{\mathrm{B}} \cap \boldsymbol{W} = \emptyset$.

Intuitively, Definition 2 ensures that $\boldsymbol{W}$ acts as a "standalone linear unit" in the DNN. In practice, most DNNs can be decomposed into layers under this definition[1]. Because $\boldsymbol{W}$ is generally a *matrix*, we flatten it to a *vector* $\boldsymbol{w}$ in Definition 1.

We can calculate the *per-example* gradient of $\boldsymbol{W}$ from Definition 2:

$$\frac{\partial \ell}{\partial \boldsymbol{W}}[\mathfrak{x}; \mathcal{W}] = \mathbf{h}[\mathfrak{x}; \mathcal{W}^{\mathrm{F}}] \frac{\partial \ell}{\partial \boldsymbol{y}}[\mathfrak{x}; \boldsymbol{y}, \mathcal{W}^{\mathrm{B}}]. \tag{4}$$

In the RHS above, the only component that depends on $\boldsymbol{W}$ is $\boldsymbol{y}$. Therefore, if we focus on the evolution of $\boldsymbol{W}_{t+s}$, everything except the linear regression $\boldsymbol{y}_{t+s}$ can be treated as "data given". This is the main idea for the next section.

3.2 Per-Layer Approximation by Random Feature Models

Let $\boldsymbol{W}$ be a $\mathtt{C} \times \mathtt{D}$ matrix. The next theorem suggests that one can approximate the per-example gradients of $\boldsymbol{W}$, as if they come from a (generalized) Random Feature Model.

Theorem 3. *Fix a training step t. There exists a map $(\mathfrak{x}, s) \mapsto \mathbf{x}_s$, such that for any training example $\mathfrak{x}$ and step $s \geq 0$, a $\mathtt{V}$-dimensional vector $\mathbf{x}_s$ satisfies the following:*

(a) The $\mathtt{D}$-dimensional vector $\mathbf{h}_{t+s} \stackrel{\text{def}}{=} \mathbf{h}[\mathfrak{x}; \mathcal{W}^{\mathrm{F}}_{t+s}]$ can be approximated by $\mathbf{A}\mathbf{x}_s$, where $\mathbf{A}$ is a constant $\mathtt{D} \times \mathtt{V}$ matrix that does not depend on $\mathfrak{x}$ or s.

(b) The $\mathtt{C}$-dimensional vector $\nabla \boldsymbol{y}_{t+s} \stackrel{\text{def}}{=} \frac{\partial \ell}{\partial \boldsymbol{y}^\top}[\mathfrak{x}; \boldsymbol{y}_{t+s}, \mathcal{W}^{\mathrm{B}}_{t+s}]$ (where $\boldsymbol{y}_{t+s} \stackrel{\text{def}}{=} \boldsymbol{W}_{t+s}\mathbf{h}_{t+s}$), can be approximated by $\boldsymbol{G}_{t+s}\mathbf{x}_s$, where $\boldsymbol{G}_{t+s}$ is an affine function of $\boldsymbol{W}_{t+s}$ and a $\mathtt{C} \times \mathtt{V}$ matrix that does not depend on $\mathfrak{x}$.

At first glance, the *linear* approximation as required by Theorem 3 may seem unreasonably strong, because DNNs are highly non-linear – which is why our proof (given in § B) uses a latent space of extremely high dimensionality ($\mathtt{V} \approx$ #training-examples $\times$ #model-parameters) to "linearize" the DNN, as inspired by Neural Tangent Kernel. Therefore, the vector $\mathbf{x}_s$ and matrices $\mathbf{A}$, $\boldsymbol{G}_{t+s}$ are infeasible to calculate in practice – However, they provide the necessary framework to apply powerful assumptions (e.g. the independence of entries of $\mathbf{x}_s$ under suitable change of basis) and theoretical tools (e.g. SVD decomposition of $\mathbf{A}$), enabling predictions that can be empirically verified.

Under stochastic optimization, $\mathfrak{x} \sim \mathcal{D}$ is randomly drawn from the training data, so its corresponding representation $\mathbf{x}_s$ is a random vector. Although the distribution of $\mathfrak{x}$ does not depend on the training step, the construction of $\mathbf{x}_s$ in Theorem 3 does. In other words, there is generally *distribution drift* of $\mathbf{x}_s$ across different s. However, in this work, we impose a strong assumption that the distribution drift is negligible when s is small. In fact, we assume that the distribution of $\mathbf{x}_s$ is explicitly given as i.i.d. Gaussian, so that we can align with a PLRF model:

Assumption 4. One can replace the $\mathbf{x}_s$ in Theorem 3 by a suitable linear transformation $\mathbf{T}\mathbf{x}$, where $\mathbf{T}$ does not depend on $\mathfrak{x}$ or s, and the entries of $\mathbf{x}$ are i.i.d. Gaussian of mean 0 and variance 1.

When Assumption 4 is applied, we can omit the subscript of $\mathbf{x}_s$ and absorb the linear transformation $\mathbf{T}$ into $\mathbf{A}$ and $\boldsymbol{G}_{t+s}$. So $\mathbf{h}_{t+s} \approx \mathbf{A}\mathbf{x}$ and $\nabla \boldsymbol{y}_{t+s} \approx \boldsymbol{G}_{t+s}\mathbf{x}$, with the entries of $\mathbf{x}$ being i.i.d. Gaussian of mean 0 and variance 1. This assumption enables explicit calculation of the covariance of the per-example gradient, $\nabla \boldsymbol{W}_{t+s} \stackrel{\text{def}}{=} \frac{\partial \ell}{\partial \boldsymbol{W}^\top}[\mathfrak{x}; \mathcal{W}_{t+s}] \approx \boldsymbol{G}_{t+s}\mathbf{x}\,\mathbf{x}^\top\mathbf{A}^\top$, as below.

[1] In certain model designs, "shared weights" can break the conditions. For example, some LMs reuse the input embedding layer as the output logits head, which breaks condition (2a) and (2b) because the same weights are used at different sites of the model. We will leave such cases to future work.

Prediction 5. *Using Assumption 4, and assume that* $\mathtt{D}$ *is sufficiently large. Let* $\mathbb{E}(\mathbf{h}_t\mathbf{h}_t^\top)$ *be the correlation matrix of input hidden vectors, where* $\mathbb{E}$ *is averaging over the training data* $\mathfrak{x} \sim \mathcal{D}$*. Let* $\mathbb{E}(\mathbf{h}_t\mathbf{h}_t^\top) = \mathbf{U}\,\mathbf{Diag}(\boldsymbol{\lambda})\mathbf{U}^\top$ *be the eigen decomposition. Then,*

$$\mathbb{E}(\boldsymbol{g}_i\boldsymbol{g}_j{}^\top) \approx \begin{cases} \lambda_i\,\mathbb{E}(\nabla \boldsymbol{y}_{t+s}\nabla \boldsymbol{y}_{t+s}^\top) & \text{if } i = j \\ \mathbf{0} & \text{if } i \neq j \end{cases} \tag{5}$$

where $\boldsymbol{g}_i$*,* $\boldsymbol{g}_j$ *are the* i, j*-th column vectors of the* $\mathtt{C} \times \mathtt{D}$ *matrix* $(\nabla \boldsymbol{W}_{t+s})\mathbf{U}$*. In particular, we have*

$$\mathbb{E}(\|\nabla \boldsymbol{W}_{t+s}\|^2) \approx \sum_{i=1}^{\mathtt{D}} \lambda_i\,\mathbb{E}(\|\nabla \boldsymbol{y}_{t+s}\|^2) \tag{6}$$

by taking the trace of (5)*.*

We theoretically prove Prediction 5 in § B.2 and empirically verify in § D.3. This is a non-trivial result relying on the high-dimensionality of $\mathtt{D}$: Intuitively, it claims that for $\nabla \boldsymbol{W}_{t+s} = (\nabla \boldsymbol{y}_{t+s})\,\mathbf{h}_{t+s}^\top$, one can calculate the expected gradient norm $\mathbb{E}(\|\nabla \boldsymbol{W}_{t+s}\|^2)$ as if $\|\nabla \boldsymbol{y}_{t+s}\|$ and $\|\mathbf{h}_{t+s}\|$ are independent – Of course, $\nabla \boldsymbol{y}_{t+s} \approx \boldsymbol{G}_{t+s}\mathbf{x}$ and $\mathbf{h}_{t+s} \approx \mathbf{A}\mathbf{x}$ are *not* independent because they share a common factor $\mathbf{x}$ – But the "dependent part" turns out to be negligible when $\mathtt{D}$ is large. In our case-study of a Transformer LM, we show that the empirical error of (6) is mostly within 5%, even though the two sides simultaneously vary multiple magnitudes across different layers and training steps.

3.3 Simulation Algorithm

From the layer-wise quadratic loss $L[\boldsymbol{w}]$ one can only obtain the *average* gradient (which matches the approximate average gradient of the DNN, as calculated from the layer-wise Taylor expansion to the 2nd order); But in order to simulate stochastic optimization, one has to synthesize *per-example* gradients, or more generally the *mini-batched* gradients of the DNN.

Our idea is to use Prediction 5, which provides the 2nd moment of per-example gradients. Roughly speaking, we can synthesize "pseudo gradients" by randomly sampling Gaussians[2] to match the average and 2nd moment, at each step of the simulation.

In Prediction 5, one can directly measure $\mathbb{E}(\mathbf{h}_t\mathbf{h}_t^\top)$ from forward activations on a validation set (and Assumption 4 implies it does not change over time), but $\mathbb{E}(\nabla \boldsymbol{y}_{t+s}\nabla \boldsymbol{y}_{t+s}^\top)$ is tricky because it depends on $\boldsymbol{W}_{t+s}$. However, for the Random Feature model given in (1), where

$$y_{t+s} \stackrel{\text{def}}{=} \boldsymbol{w}_{t+s}^\top \mathbf{A}\mathbf{x}$$

is a *scalar*, one has

$$\mathbb{E}\big((\nabla y_{t+s})^2\big) = \mathbb{E}\Big(c^2\big(\boldsymbol{w}_{t+s}^\top \mathbf{A}\mathbf{x} - b[\mathbf{x}]\big)^2\Big) = 2c\,\mathbb{E}\big(\ell[\mathbf{x}; \boldsymbol{w}_{t+s}]\big) \tag{7}$$

which is a *multiple of the validation loss*, a presumed quantity that we can handle. Essentially, this simplification (7) leads to the Volterra equation investigated in Paquette et al. (2021) and the analytic solution of SGD training dynamics for Random Feature models. In this paper, we further apply (7) to DNNs, by making the following assumption:

Assumption 6. $\mathbb{E}(\nabla \boldsymbol{y}_{t+s}\nabla \boldsymbol{y}_{t+s}^\top)$ can be roughly approximated by $\big(\mathbb{E}(\|\nabla \boldsymbol{y}_{t+s}\|^2)/\mathtt{C}\big)\mathbf{I}$, where $\mathbf{I}$ is the identity matrix.

This final assumption hypothesizes that most eigenvalues of $\mathbb{E}(\nabla \boldsymbol{y}_{t+s}\nabla \boldsymbol{y}_{t+s}^\top)$ are close to each other, so that one can approximately collapse it into a scalar operator (which equals the mean eigenvalue $\mathbb{E}(\|\nabla \boldsymbol{y}_{t+s}\|^2)/\mathtt{C}$). As a result, our simulation approximates the behavior of a $\mathtt{D}$-dimensional "average slice" of the $\mathtt{C} \times \mathtt{D}$ matrix $\boldsymbol{W}$, so that one can apply (7) and predict the dynamics.

The simulation pseudo-code is given in Algorithm 1. We choose the constant c in the Random Feature model (1) such that the trace of the Hessian matches that of the DNN Hessian in the layer-wise quadratic loss (3). This approximation is necessary also because when the DNN is large, the

[2]For Random Feature models, the distribution of the per-example gradient $\boldsymbol{G}_{t+s}\mathbf{x}\,\mathbf{x}^\top\mathbf{A}^\top$ is *not* Gaussian – it's a product of two Gaussian random variables. However, since the distribution is still of the exponential family, and the simulation is a long-term accumulation of gradients across multiple steps, Gaussian sampling at each step should be a good approximation for long-term, average-case analysis.

Algorithm 1 Simulating the approximate behavior of the layer-wise quadratic loss L from step t to $t+s$, where a $\mathtt{D}$-dimensional average slice $\boldsymbol{w}^\top$ of the $\mathtt{C} \times \mathtt{D}$ matrix $\boldsymbol{W}$ gets trained.

Require: $\boldsymbol{\lambda} = (\lambda_i)_{1 \le i \le \mathtt{D}}$ and $\mathbf{U}$, the eigenvalues and eigenvectors of $\mathbb{E}(\mathbf{h}_t \mathbf{h}_t^\top)$, respectively.
Require: $\mathbb{E}(\nabla \boldsymbol{W}_t)$, the average gradient of the DNN layer, at step t.
Require: $\mathbb{E}(\|\nabla_{\mathtt{B}} \boldsymbol{W}_t\|^2)$, the mean squared-norm of per-batch gradients.
Require: $\mathrm{Tr}\left(\mathbb{E}(\nabla^2 \boldsymbol{W}_t)\right)$, the trace of the layer-wise Hessian.
Require: $\phi[\bullet]$, an optimizer to map (synthetic) per-batch gradients to updates.
Ensure: $\{L_{t+j}\}_{1 \le j \le s}$, a trajectory of simulated layer-wise quadratic loss.

$c \leftarrow \mathrm{Tr}\left(\mathbb{E}(\nabla^2 \boldsymbol{W}_t)\right) / (\sum_i \lambda_i \mathtt{C})$ ▷ Approximate the constant c in (1).
$\mathfrak{B} \leftarrow \dfrac{(\sum_i \lambda_i)^2}{\sum_i \lambda_i^2} \dfrac{\|\mathbb{E}(\nabla \boldsymbol{W}_t)\|^2}{\mathbb{E}(\|\nabla_{\mathtt{B}} \boldsymbol{W}_t\|^2)}$ ▷ The equivalent batch factor.
$\boldsymbol{z} \leftarrow \mathrm{RMS}\big(\mathbb{E}(\nabla \boldsymbol{W}_t)\mathbf{U},\, \mathtt{axis} = 0\big) / \boldsymbol{\lambda}$ ▷ Initialize $c(\boldsymbol{w} - \mathbf{w}^*)$.
for $j = 0, \ldots, s$ **do**
  $L_{t+j} \leftarrow \dfrac{\mathtt{C}}{2c} \sum_i \lambda_i z_i^2$ ▷ Output.
  $n_k \sim \mathcal{N}\Big(0, \lambda_k \big| \sum_i \lambda_i z_i^2 / \mathfrak{B} - \lambda_k z_k^2 \big|\Big)$ ▷ Sample stochastic noise of per-batch gradient.
  $\boldsymbol{g} \leftarrow \boldsymbol{\lambda} \odot \boldsymbol{z} + (n_k)_{1 \le k \le \mathtt{D}}$ ▷ Synthetic per-batch gradient.
  $\boldsymbol{u} \leftarrow \mathbf{U}^\top \phi[\mathbf{U}\boldsymbol{g}]$ ▷ Call optimizer.
  $\boldsymbol{z} \leftarrow \boldsymbol{z} - c\boldsymbol{u}$ ▷ Update.
end for

full Hessian tensor of $(\mathtt{C} \times \mathtt{D})^2$ total dimensions is infeasible to calculate. Then, our algorithm tracks the $\mathtt{D}$-dimensional vector $\boldsymbol{z} = c(\boldsymbol{w} - \mathbf{w}^*)$, and initializes $\boldsymbol{z}$ such that the norms of the initial average gradients match in every eigenmode. Finally, we synthesize *per-batch* gradients instead of *per-example* gradients, depending on the batch-size. For this purpose, we define an *equivalent batch factor* $\mathfrak{B}$ which is empirically measured to match the initial mean squared-norm of per-batch gradients. Under certain conditions, $\mathfrak{B}$ coincides with the actual batch-size for Random Feature models (§ B.3), and we break down the details of $\mathfrak{B}$ for DNNs in § D.4. The theoretical derivation of Algorithm 1 is given in § B.4. When applied to PLRF models, we empirically verify that the algorithm correctly captures training dynamics, including spikes (Figure C.1).

Because our simulation does not call the backward pass of a DNN at each step – it synthesizes a random gradient instead – the algorithm is extremely lightweight. The initial inputs require $\mathbb{E}(\nabla \boldsymbol{W}_t)$, $\mathbb{E}(\|\nabla_{\mathtt{B}} \boldsymbol{W}_t\|^2)$, $\mathrm{Tr}\left(\mathbb{E}(\nabla^2 \boldsymbol{W}_t)\right)$ and (the eigen decomposition of) $\mathbb{E}(\mathbf{h}_t \mathbf{h}_t^\top)$, which cost one pass over a validation set. See Table 1 for a summary of the compute requirements.

3.4 A Sketch of why Training Spikes Occur

In this section, we briefly explain why power-law distributed inputs can lead to training spikes. For simplicity, we analyze the dynamics of $\boldsymbol{w}$ under SGD. (We expect optimizers with momentum and precondition, such as Adam, to follow a similar principle.) At any step t, let η_t be the learning-rate. The update of SGD is given by

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \nabla \boldsymbol{w}_t, \tag{8}$$

where $\nabla \boldsymbol{w}_t$ is a per-example gradient. This is a random process depending on the data sample at each step; We apply the decomposition $\nabla \boldsymbol{w}_t = \mathbb{E}(\nabla \boldsymbol{w}_t) + \boldsymbol{n}_t$, where $\mathbb{E}$ averages over the data distribution, and $\boldsymbol{n}_t$ is a random vector of mean $\mathbf{0}$, representing the stochastic noise of training samples.

Under a quadratic loss, the average gradient $\mathbb{E}(\nabla \boldsymbol{w}_t)$ is in the form $\mathbb{E}(\nabla \boldsymbol{w}_t) = \mathbf{H}(\boldsymbol{w}_t - \mathbf{w}^*)$, where $\mathbf{H}$ is the Hessian. Thus, we can decompose (8) into $\boldsymbol{w}_t = \overline{\boldsymbol{w}}_t + \boldsymbol{r}_t$, such that

$$\overline{\boldsymbol{w}}_{t+1} - \mathbf{w}^* = (\mathbf{I} - \eta_t \mathbf{H})(\overline{\boldsymbol{w}}_t - \mathbf{w}^*) \tag{9}$$

$$\text{and} \quad \boldsymbol{r}_{t+1} = (\mathbf{I} - \eta_t \mathbf{H})\, \boldsymbol{r}_t - \eta_t \boldsymbol{n}_t. \tag{10}$$

So $\overline{\boldsymbol{w}}_t$ is the predictable full-batch gradient descent, while $\boldsymbol{r}_t$ is a random walk with mean $\mathbf{0}$, accumulating the stochastic noise $\boldsymbol{n}_t$ across steps. When the Central Limit Theorem applies, $\boldsymbol{r}_t$ should approximately follow a (anisotropic) Gaussian distribution.

Experiment	Train	Eval $\delta^{\boldsymbol{w}}_{t,s}$	Eval $\mathbb{E}(\mathbf{h}_t\mathbf{h}_t^\top)$	$\mathbb{E}(\nabla \boldsymbol{W}_t)$ & $\mathbb{E}(\|\nabla_{\mathtt{B}} \boldsymbol{W}_t\|^2)$	$\mathrm{Tr}\left(\mathbb{E}(\nabla^2 \boldsymbol{W}_t)\right)$	Simulation
Data size	25.5B tokens	336M tokens	336M tokens	336M tokens	336M tokens	1500 steps
Model size	1.3B (24 blocks)	24 `ffn_0`	all layers	all layers	6 `ffn_0`	1 layer
Time	40 hrs	1.1 hrs	8 min	15 min	1.7 hrs	5 sec
Machine	TPUv4×32	TPUv4×32	TPUv4×32	TPUv4×32	TPUv4×32	CPU

Table 1: Summary of compute resources for running experiments in this paper.

Intuitively, one can imagine $\boldsymbol{r}_t$ as a high-dimension (isotropic) Gaussian with a given variance σ^2; so for any random $\boldsymbol{r}_t$, we know that $\|\boldsymbol{r}_t\|^2 \approx \mathtt{D}\sigma^2$ with high probability, while $\boldsymbol{r}_t$ can point to any direction. The fact that $\|\boldsymbol{r}_t\|^2 \approx \mathtt{D}\sigma^2$ holds (with high probability) is an example of the more general principle called "*concentration of measure*": the value of a *scalar* function of $\boldsymbol{r}_t$ "concentrates" to its expectation, because the random fluctuations of entries of $\boldsymbol{r}_t$ can cancel out each other, much like in the Law of Large Numbers.

The validation loss $\mathbb{E}\big(\ell[\mathbf{x}; \bullet]\big)$ is a scalar, quadratic function of $\boldsymbol{w}_t = \overline{\boldsymbol{w}}_t + \boldsymbol{r}_t$. So if this quadratic function is similar to $\|\bullet\|^2$, by analogy to the above, the concentration of measure principle applies and $\mathbb{E}\big(\ell[\mathbf{x}; \overline{\boldsymbol{w}}_t + \boldsymbol{r}_t]\big)$ should (with high probability) not depend on the random direction of $\boldsymbol{r}_t$. As a result, $\mathbb{E}\big(\ell[\mathbf{x}; \overline{\boldsymbol{w}}_t + \boldsymbol{r}_t]\big)$ converges to a deterministic trajectory at the high-dimension limit $\mathtt{D} \to \infty$.

However, in order for the "concentration of measure" to hold, the eigenvalues of the quadratic function $\mathbb{E}\big(\ell[\mathbf{x}; \bullet]\big)$ cannot be too heavy-tailed. If there are some very large eigenvalues that cannot be counter-weighted by all the small ones, the value of $\mathbb{E}\big(\ell[\mathbf{x}; \overline{\boldsymbol{w}}_t + \boldsymbol{r}_t]\big)$ will suddenly surge when $\boldsymbol{r}_t$ randomly points toward directions of the large eigenvalues – a "*spike*" will occur.

We also note that, the learning-rate has to be relatively large so that $\boldsymbol{r}_t$ can accumulate to a significant amount, for the training spikes to manifest. On the other hand, $\boldsymbol{r}_t$ starts to grow larger and larger if the learning-rate is too high – which means the training diverges. The threshold γ of divergence can be accurately calculated for SGD, which is given in (2). One observes large training spikes when the learning-rate is slightly below γ. For general optimizers, one can use our simulation to find the threshold learning-rate, because Algorithm 1 faithfully models the accumulation of random gradient noise. For more technical details, please refer to Appendix B.5 and B.6.

4 Simulation of Transformer LM Training

As a case-study, we train a vanilla Transformer decoder model of 1.3B parameters (24 Transformer blocks) on the C4 dataset, to the Chinchilla optimal number of tokens. We use the AdamW optimizer with hyperparameters inherited from previous research (Everett et al., 2024; Charles et al., 2025), which are tuned to near optimal. Then, we change β_2 of AdamW from 0.95 to 0.999 in order to produce training spikes. We obtain three training trajectories: Normal, Spiky1 and Spiky2, as illustrated in Figure 1.1. More details are given in § D.1.

We take the model checkpoints at step 21k, and run simulations for all three configurations. Optimizer states are initialized from model checkpoints, but we did not apply weight-decay during the simulation. Figure 4.1 shows the results of 6 feed-forward layers simulated (with 2 random seeds each) from step 21k to 22.5k. We see clear bumps in the simulation for *both* Spiky1 and Spiky2 (e.g., block_{0,4,8} at step ∼22k for Spiky1, and block_{0,12} at step ∼21.5k for Spiky2), while the simulation for Normal is comparatively more smooth. It suggests that our simulation has the predictive power of telling whether a training configuration can lead to spikes, from a given model checkpoint. It is noteworthy that in the actual training trajectory (Figure 1.1), only Spiky2 spikes between steps 21k and 22k; And the spiky positions in our simulations differ between random seeds, demonstrating the random nature of training spikes. In § D.2, we will also show that the distribution of eigenvalues of $\mathbb{E}(\mathbf{h}_t\mathbf{h}_t^\top)$ is significantly more heavy-tailed for spiky configurations.

On the other hand, we plot the 2nd-order term of $\delta^{\boldsymbol{w}}_{t,s}$ (Definition 1), from step 21k to 22k, in Figure 4.1 with a shifted origin but the same scale, in order to compare the largeness of fluctuations. We observe jumps in the 2nd-order terms from Spiky2 synchronized with the training spike, while the largeness of the jumps matches the fluctuation of simulations in some layers (e.g. block_{4,8,12}) if not all. The 1st-order term of $\delta^{\boldsymbol{w}}_{t,s}$, and more discussion can be found in Appendix D.6.

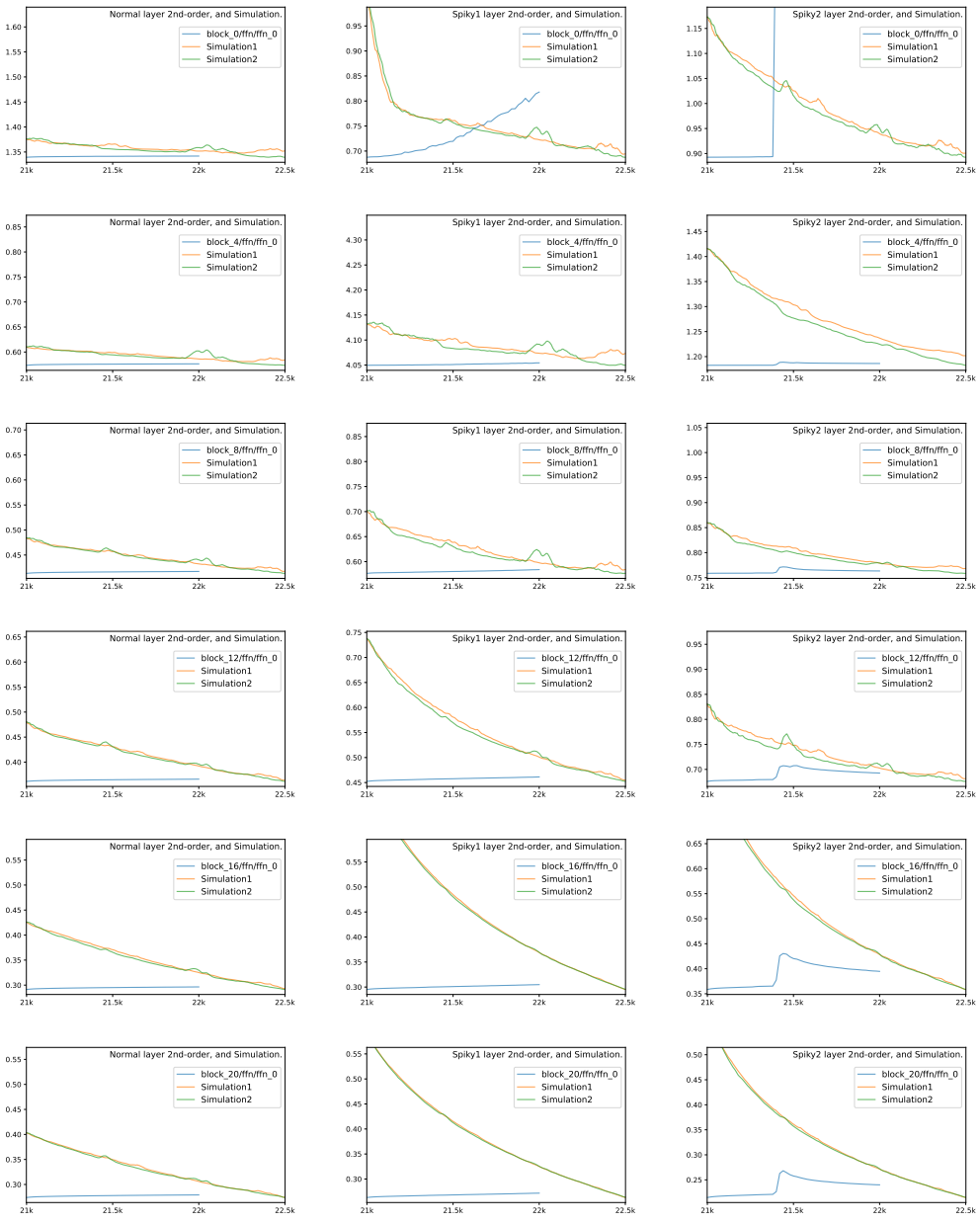


Figure 4.1: Simulation of the layer-wise quadratic approximation of validation loss at 6 layers (the feed-forward input of block_{0,4,8,12,16,20}), from training step 21k to 22.5k. We compare 3 settings of training (Normal, Spiky1 and Spiky2) as in Figure 1.1, and plot two simulations of different seeds, together with the actual 2nd-order term measured from corresponding training trajectories. The scale of the y-axis is kept the same for all plots, so that the fluctuations can be compared.

5 Conclusion

We have seen PLRF models exhibiting training spikes, and presented the theory, algorithm and experiments to align PLRF models with layerwise training dynamics of DNNs. Given the apparent similarity between empirical training spikes and our simulations, we believe quadratic models at least partially reveals the essence of underlying mechanism. Due to the page limit, we put more discussion and related works in Appendix A.

AI USE STATEMENT

We have not used generative AI tools for developing theory, formulating mathematical claims, designing experiments, or implementing methods. This is completely human work. During the writing of this paper, we have consulted AI for rephrasing some paragraphs of the draft, and summarizing some related papers; we have checked the outputs and manually rewrote the text. We take responsibility for the final content of this work, including text, claims, mathematical proofs and code.

REPRODUCIBILITY STATEMENT

Our experiments with PLRF models are easy to reproduce; we provide Python code in supplementary materials. Our Transformer LM experiments are based on a decoder-only Transformer with common components. All the hyperparameters are listed in § D.1, where we also describe how they were tuned. The training dynamics we present in this paper depends on the hyperparameters, especially the learning-rate schedule and weight-decay; But the phenomena we observe should be general across different modeling choices. For mathematical details of our theory, please refer to Appendix B.

REFERENCES

Ben Adlam and Jeffrey Pennington. Understanding double descent requires a fine-grained bias-variance decomposition. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Atish Agarwala and Jeffrey Pennington. High dimensional analysis reveals conservative sharpening and a stochastic edge of stability, 2025. URL https://arxiv.org/abs/2404.19261.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining neural scaling laws. *Proceedings of the National Academy of Sciences*, 121(27):e2311878121, 2024. doi: 10.1073/pnas.2311878121. URL https://www.pnas.org/doi/abs/10.1073/pnas.2311878121.

Zhiwei Bai, Zhangchen Zhou, Jiajie Zhao, Xiaolong Li, Zhiyu Li, Feiyu Xiong, Hongkang Yang, Yaoyu Zhang, and Zhi-Qin John Xu. Adaptive preconditioners trigger loss spikes in adam. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=STWQoscanw.

Blake Bordelon, Alexander B. Atanasov, and Cengiz Pehlevan. A dynamical model of neural scaling laws. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 4345–4382. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/bordelon24a.html.

Zachary Charles, Gabriel Teston, Lucio M. Dery, Keith Rush, Nova Fallen, Zachary Garrett, Arthur Szlam, and Arthur Douillard. Communication-efficient language model training scales reliably and robustly: Scaling laws for diloco. *CoRR*, abs/2503.09799, 2025. doi: 10.48550/ARXIV.2503.09799. URL https://doi.org/10.48550/arXiv.2503.09799.

Lizhang Chen, Jonathan Li, and Qiang Liu. Muon optimizes under spectral norm constraints. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=Blz4hjxLwU.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James

Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24:240:1–240:113, 2023. URL https://jmlr.org/papers/v24/22-1144.html.

Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, 2009. doi: 10.1137/070710111. URL https://doi.org/10.1137/070710111.

Jeremy Cohen, Simran Kaur, Yuanzhi Li, J. Zico Kolter, and Ameet Talwalkar. Gradient descent on neural networks typically occurs at the edge of stability. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=jh-rTtvkGeM.

Jeremy Cohen, Behrooz Ghorbani, Shankar Krishnan, Naman Agarwal, Sourabh Medapati, Michal Badura, Daniel Suo, David Cardoze, Zachary Nado, George E. Dahl, and Justin Gilmer. Adaptive gradient methods at the edge of stability. *CoRR*, abs/2207.14484, 2022. doi: 10.48550/ARXIV.2207.14484. URL https://doi.org/10.48550/arXiv.2207.14484.

Jeremy Cohen, Alex Damian, Ameet Talwalkar, J. Zico Kolter, and Jason D. Lee. Understanding optimization in deep learning with central flows. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=sIE2rI3ZPs.

Alex Damian, Eshaan Nichani, and Jason D. Lee. Self-stabilization: The implicit bias of gradient descent at the edge of stability. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=nhKHA59gXz.

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, Rodolphe Jenatton, Lucas Beyer, Michael Tschannen, Anurag Arnab, Xiao Wang, Carlos Riquelme Ruiz, Matthias Minderer, Joan Puigcerver, Utku Evci, Manoj Kumar, Sjoerd Van Steenkiste, Gamaleldin Fathy Elsayed, Aravindh Mahendran, Fisher Yu, Avital Oliver, Fantine Huot, Jasmijn Bastings, Mark Collier, Alexey A. Gritsenko, Vighnesh Birodkar, Cristina Nader Vasconcelos, Yi Tay, Thomas Mensink, Alexander Kolesnikov, Filip Pavetic, Dustin Tran, Thomas Kipf, Mario Lucic, Xiaohua Zhai, Daniel Keysers, Jeremiah J. Harmsen, and Neil Houlsby. Scaling vision transformers to 22 billion parameters. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 7480–7512. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/dehghani23a.html.

Aleksandr Dremov, Alexander Hägele, Atli Kosson, and Martin Jaggi. Training dynamics of the cooldown stage in warmup-stable-decay learning rate scheduler. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=ZnSYEcZod3.

Katie E. Everett, Lechao Xiao, Mitchell Wortsman, Alexander A. Alemi, Roman Novak, Peter J. Liu, Izzeddin Gur, Jascha Sohl-Dickstein, Leslie Pack Kaelbling, Jaehoon Lee, and Jeffrey Pennington. Scaling exponents across parameterizations and optimizers. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 12666–12700. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/everett24a.html.

Chen Fan, Mark Schmidt, and Christos Thrampoulidis. Implicit bias of spectral descent and muon on multiclass separable data. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=Zn2ajV1kTQ.

Damien Ferbach, Katie Everett, Gauthier Gidel, Elliot Paquette, and Courtney Paquette. Dimension-adapted momentum outscales SGD. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=t4aN2G7Ucc.

Damien Ferbach, Courtney Paquette, Gauthier Gidel, Katie Everett, and Elliot Paquette. Logarithmic-time schedules for scaling language models with momentum, 2026. URL https://arxiv.org/abs/2602.05298.

Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via hessian eigenvalue density. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2232–2241. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/ghorbani19b.html.

Justin Gilmer, Behrooz Ghorbani, Ankush Garg, Sneha Kudugunta, Behnam Neyshabur, David Cardoze, George Edward Dahl, Zachary Nado, and Orhan Firat. A loss curvature perspective on training instabilities of deep learning models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=OcKMT-36vUs.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/ioffe15.html.

Arthur Jacot, Clément Hongler, and Franck Gabriel. Neural tangent kernel: Convergence and generalization in neural networks. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 8580–8589, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/5a4be1fa34e62bb8a6ec6b91d2462f5a-Abstract.html.

Stanislaw Jastrzebski, Zachary Kenton, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amost Storkey. On the relation between the sharpest directions of DNN loss and the SGD step length. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SkgEaj05t7.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp.

8570–8581, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/0d1a9651497a38d8b1c3871c84528bd4-Abstract.html.

Aitor Lewkowycz, Yasaman Bahri, Ethan Dyer, Jascha Sohl-Dickstein, and Guy Gur-Ari. The large learning rate phase of deep learning: the catapult mechanism, 2020. URL https://arxiv.org/abs/2003.02218.

Qianxiao Li, Cheng Tai, and E. Weinan. Stochastic modified equations and dynamics of stochastic gradient algorithms i: mathematical foundations. *J. Mach. Learn. Res.*, 20(1):1474–1520, January 2019. ISSN 1532-4435.

Xiaolong Li, Zhi-Qin John Xu, and Zhongwang Zhang. Loss spike in training neural networks. *Journal of Computational Mathematics*, 44(2):369–393, 2026. doi: 10.4208/jcm.2412-m2024-0083. URL https://doi.org/10.4208/jcm.2412-m2024-0083.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019. URL https://arxiv.org/abs/1907.11692.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Alexander Maloney, Daniel A. Roberts, and James Sully. A solvable model of neural scaling laws. *CoRR*, abs/2210.16859, 2022. doi: 10.48550/ARXIV.2210.16859. URL https://doi.org/10.48550/arXiv.2210.16859.

Stephan Mandt, Matthew Hoffman, and David Blei. A variational analysis of stochastic gradient algorithms. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 354–363, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/mandt16.html.

Noah Marshall, Ke Liang Xiao, Atish Agarwala, and Elliot Paquette. To clip or not to clip: the dynamics of sgd with gradient clipping in high-dimensions, 2024. URL https://arxiv.org/abs/2406.11733.

Charles H. Martin and Michael W. Mahoney. Implicit self-regularization in deep neural networks: Evidence from random matrix theory and implications for learning. *Journal of Machine Learning Research*, 22(165):1–73, 2021. URL http://jmlr.org/papers/v22/20-410.html.

Igor Molybog, Peter Albert, Moya Chen, Zachary DeVito, David Esiobu, Naman Goyal, Punit Singh Koura, Sharan Narang, Andrew Poulton, Ruan Silva, Binh Tang, Diana Liskovich, Puxin Xu, Yuchen Zhang, Melanie Kambadur, Stephen Roller, and Susan Zhang. A theory on adam instability in large-scale machine learning. *CoRR*, abs/2304.09871, 2023. doi: 10.48550/ARXIV.2304.09871. URL https://doi.org/10.48550/arXiv.2304.09871.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=B1g5sA4twr.

Kosuke Nishida, Kyosuke Nishida, and Kuniko Saito. Initialization of large language models via reparameterization to mitigate loss spikes. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 22699–22714, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1264. URL https://aclanthology.org/2024.emnlp-main.1264/.

Courtney Paquette, Kiwon Lee, Fabian Pedregosa, and Elliot Paquette. SGD in the large: Average-case analysis, asymptotics, and stepsize criticality. In Mikhail Belkin and Samory Kpotufe (eds.), *Conference on Learning Theory, COLT 2021, 15-19 August 2021, Boulder, Colorado, USA*, volume 134 of *Proceedings of Machine Learning Research*, pp. 3548–3626. PMLR, 2021. URL http://proceedings.mlr.press/v134/paquette21a.html.

Elliot Paquette, Courtney Paquette, Lechao Xiao, and Jeffrey Pennington. 4+3 phases of compute-optimal neural scaling laws. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/1dccfc3ee01871d05e33457c61037d59-Abstract-Conference.html.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In Sanjoy Dasgupta and David McAllester (eds.), *Proceedings of the 30th International Conference on Machine Learning*, Proceedings of Machine Learning Research, pp. 1310–1318, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL https://proceedings.mlr.press/v28/pascanu13.html.

Shikai Qiu, Lechao Xiao, Andrew Gordon Wilson, Jeffrey Pennington, and Atish Agarwala. Scaling collapse reveals universal dynamics in compute-optimally trained neural networks. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=Fvq9ogLnLN.

Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=ryQu7f-RZ.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4596–4604. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/shazeer18a.html.

Jianlin Su, Murtadha H. M. Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. doi: 10.1016/J.NEUCOM.2023.127063. URL https://doi.org/10.1016/j.neucom.2023.127063.

Sho Takase, Shun Kiyono, Sosuke Kobayashi, and Jun Suzuki. Spike no more: Stabilizing the pre-training of large language models. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=52YBEzcI0l.

Kumiko Tanaka-Ishii. *Statistical Universals of Language: Mathematical Chance vs. Human Choice*. Springer, Switzerland, 2021. ISBN 978-3-030-59376-6. doi: 10.1007/978-3-030-59377-3.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Mitchell Wortsman, Tim Dettmers, Luke Zettlemoyer, Ari S. Morcos, Ali Farhadi, and Ludwig Schmidt. Stable and low-precision training for large-scale vision-language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=sqqASmpA2R.

Mitchell Wortsman, Peter J Liu, Lechao Xiao, Katie E Everett, Alexander A Alemi, Ben Adlam, John D Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, Jeffrey Pennington, Jascha Sohl-Dickstein, Kelvin Xu, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Small-scale proxies for large-scale transformer training instabilities. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=d8w0pmvXbZ.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture, 2020. URL https://arxiv.org/abs/2002.04745.

Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJgnXpVYwS.

Libin Zhu, Chaoyue Liu, Adityanarayanan Radhakrishnan, and Mikhail Belkin. Catapults in sgd: spikes in the training loss and their impact on generalization through feature learning, 2024. URL https://arxiv.org/abs/2306.04815.

George Kingsley Zipf. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*. Addison-Wesley Press, Cambridge, MA, 1949.

A Discussion and Related Works

Theoretical analysis of training spikes in most of previous works has focused on the concept of *sharpness*, defined as the largest eigenvalue $\lambda_{\max}$ of the Hessian. But the truth is, none of them has been able to create toy models with loss dynamics, that can match aesthetically the phenomenon we empirically documented as "training spikes", such as in Figure 1.1a. Some works show fluctuations with a log-scale in y-axis (which will amplify the largeness of the fluctuations in appearance), and try to connect them with training spikes – **I'm sorry, but they are not similar.**

Thus, we first want to emphasize the significance of our examples in PLRF models, solely from an aesthetics perspective: Please note the similarity of spikes between Figure 1.1a and Figure 2.1, and the fact that neither of them has log-scales in y-axis. We provide Python code of the PLRF examples in supplementary materials, which can run on Google Colab within 2 minutes. **Please just try it.**

In order to analyze the behavior of PLRF models under stochastic optimization, one has to investigate the gradient noise – a factor that does not exist in full-batch training and has been largely overlooked in previous theoretical study of training spikes. In practice, although recent LMs are trained with large numbers of tokens per batch, we especially investigated the *equivalent batch factor* in this paper and found that the gradient noise in training a Transformer LM is comparable to a Random Feature model with batch-size ≈ 1 (§ D.4). This is because the point-wise training loss of DNNs can increase gradient noise (§ B.3) – an effect specific to DNNs that can cancel the large batch-size. **In short, one cannot ignore gradient noise for training a Transformer LM.**

As a result, the sharpness $\lambda_{\max}$ is only distantly relevant to training dynamics of stochastic optimization. What really matters is the *distribution* of all eigenvalues $\{\lambda_i\}$ – That said, we would like to discuss several streams of previous works next, which have contributed to our understanding of training spikes, as well as the more general topic of training process of DNNs.

Feature Learning and Sharpness The Hessian is ever changing during the training of DNNs – a fact that cannot be captured by quadratic models like PLRF. Understanding how the Hessian evolves, which we call "feature learning", is an important topic to understand deep learning. It is reported by Jastrzebski et al. (2019); Cohen et al. (2021; 2022) that during the training of DNNs, the sharpness $\lambda_{\max}$ is increasing until it meets the threshold $2/\gamma$ (where γ is the learning-rate); And when γ decreases, $\lambda_{\max}$ increases further. This process, called "progressive sharpening", can be affected by several optimization hyperparameters (Gilmer et al., 2022), and it is connected to training spikes through Adam preconditioner (Bai et al., 2026), the "lower-loss-as-sharper structure" and condensation of model weights (Li et al., 2026). The phase where learning-rate is set to larger than $2/\lambda_{\max}$, called "catapult" (Lewkowycz et al., 2020), is directly connected to training spikes as well (Zhu et al., 2024). When the learning-rate temporarily exceeds $2/\lambda_{\max}$, Damian et al. (2023); Cohen et al. (2025) show that a 3rd-order term will automatically stabilize the sharpness and prevent it from increasing further – the so-called "Edge-of-Stability". We believe the insights regarding 3rd-order terms can be valuable in stochastic optimization as well: As we discussed in § 4 and § D.6, both the simulation and empirical measurement of layer-wise quadratic loss cannot fully explain the scale of spikes observed in the total validation loss; the random fluctuations driven by gradient noises in quadratic forms may get *amplified* and *stabilized* by higher-order terms.

Spectrum Analysis of Hessian and Model Weights Beyond the largest eigenvalue $\lambda_{\max}$, our empirical measurement of the whole spectrum of the layer-wise Hessian (especially the trace of the Hessian) is largely inspired by Ghorbani et al. (2019). Unfortunately, the computation conducted in Ghorbani et al. (2019) becomes infeasible for the model size of recent LMs; In this work, we take a shortcut by sticking to the *layer-wise* Hessian instead of the Hessian of the whole model, and we make Assumption 6 to largely reduce the computation to Hessian trace and eigen decomposition of correlation matrices of forward activations. Nevertheless, we note that Ghorbani et al. (2019) observed heavy-tailed distribution of eigenvalues similar to what we observe in Figure D.2. From another perspective, Martin & Mahoney (2021) investigated the spectrum of model weights, and recently the spectrum of gradients and moments become a topic of great interest (Fan et al., 2025; Chen et al., 2026), thanks to the popularity of Muon optimizer (Jordan et al., 2024). Our work is not directly connected to this stream because we simulate an average *slice* of the model weight, rather than the whole matrix. Based on the observation by Martin & Mahoney (2021) that the spectrum of model weights gets more heavy-tailed during training, we initially expected the same for Hessian;

However, we found that although the Hessian trace is increasing (Figure D.7 Third Row, in § D.5), the exponent α of the power-law fit is actually slightly *decreasing* for Normal training (Figure D.3, in § D.2). Still, optimization hyperparameters such as β_2 of Adam clearly affect the distribution of eigenvalues of Hessian; how to understand its mechanism remains a largely open problem.

Scaling Statistics of Language Universals It is not a coincidence that the eigenvalues of correlation matrices of forward activations of an LM are power-law distributed, because natural language data generally follow power-law distributions like Zipf's Law (Zipf, 1949). In fact, if we view the input embedding layer of a Transformer LM as a linear map from one-hot vectors to token embeddings, the correlation matrix of the inputs is $\mathbf{Diag}((p_i))$ where p_i is the unigram probability of the i-th token. So the distribution of the eigenvalues of $\mathbf{Diag}((p_i))$ is exactly Zipf's Law, and the power-law exponent is exactly $\alpha = 0.5$ in terms of the definition of α in PLRF (§ 2). Scaling statistics of language universals have long been studied (Tanaka-Ishii, 2021), and we believe they are generally behind various neural scaling laws for training LLMs (Bahri et al., 2024).

Double Descent The idea of distinguishing data-oriented and model-oriented randomness, as we did for training spikes, has also appeared in the explanation of Double Descent by Random Feature models (Adlam & Pennington, 2020). From a very general point of view, this is not a coincidence because Double Descent can be seen as some kind of "spikes" as well. Indeed, Nakkiran et al. (2020) has coined the term "epoch-wise double descent" and used the concept to explain temporary increase of validation loss at training time.

The β_2 of Adam Optimizer It has been widely known to practitioners that setting β_2 close to 1 can cause training spikes in Transformer LMs (e.g. Liu et al., 2019). Molybog et al. (2023) and Bai et al. (2026) specifically build their theory of training spikes upon this observation. In this paper, we also propose our recipe for choosing the appropriate β_2 schedule (§ D.5), and we show it improves performance in our case. The observation that an increasing schedule of β_2 can improve performance has been reported before (e.g. Reddi et al., 2018; Shazeer & Stern, 2018; Dremov et al., 2025; Ferbach et al., 2026). In practice, however, model performance and the occurrence of training spikes are affected by a large number of other factors as well. Nevertheless, the exact mechanism of how β_2 can affect the training dynamics in such a seemingly drastic way, especially how it affects the distribution of eigenvalues of the Hessian, is a curious mystery.

Optimization and Modeling Techniques to Improve Training Stability A huge amount of modeling and optimization techniques, such as normalization (Ioffe & Szegedy, 2015; Ba et al., 2016; Xiong et al., 2020; Dehghani et al., 2023; Wortsman et al., 2024) and clipping (Pascanu et al., 2013; Wortsman et al., 2023; Zhang et al., 2020; Marshall et al., 2024), *inter alia*, have been proposed to improve the training stability of DNNs. From the perspective of (avoiding) training spikes, our theory suggests essential metrics, such as the power-law exponent of the distribution of eigenvalues of the correlation matrices of forward activations, to be monitored as an indicator of healthy training. Our lightweight simulation algorithm can be applied directly to testing optimization techniques as well. That said, the empirical recipes explored in this work such as β_2 schedule, are intended as "proof of concept" for the theory, rather than alternatives to replace previous methods.

Gradient Noise and Average-case Analysis Even a simple model like PLRF can exhibit complicated dynamics such as training spikes. We urge the community to move from analyzing full-batch gradient descent and worst-case bounds, to the more realistic and challenging settings of stochastic optimization and average-case behavior of DNN training, with heavy-tailed input data and scaling-up of model sizes. Towards this direction, Mandt et al. (2016); Li et al. (2019) have developed stochastic differential equations to capture the training dynamics of SGD, which assume that the gradient noise is Gaussian (or more precisely, Brownian motion since the process is continuous), as similar to what we do for the simulation. What's different in this work is that the variance of gradient noise can be explicitly calculated for Random Feature models (i.e., Prediction 5), and we use that calculation instead of assuming the variance is a given constant (it's not constant, even for Random Feature models). On a different note, Agarwala & Pennington (2025) has coined the term "stochastic edge-of-stability", to emphasize the difference in learning-rate thresholds between full-batch gradient descent and SGD. In Qiu et al. (2025), it is shown that the gradient noise is a major factor controlled by learning-rate schedules, that can shape universally similar loss trajectories across model scales.

B THEORY AND PROOF

In this section, we give details of our theory and proof. For notation, we use bold lowercase (e.g. $\mathbf{h}$, $\boldsymbol{w}$, $\boldsymbol{y}$) to denote column vectors, and bold uppercase (e.g. $\mathbf{A}$, $\boldsymbol{W}$) for matrices. We also make the convention to denote trainable tensors (e.g., $\boldsymbol{w}$, $\boldsymbol{W}$, and $\boldsymbol{y}$ which depends on $\boldsymbol{W}$) by italic, and non-trainable constant tensors (e.g. $\mathbf{h}$, $\mathbf{A}$, $\mathbf{x}$) otherwise. All function-calls are denoted exclusively by the $[\bullet]$ bracket, in order to distinguish from the ordinary parentheses $()$. Derivatives (i.e., gradients) can be represented by either $\frac{\partial \ell}{\partial \boldsymbol{y}}$ or $\nabla \boldsymbol{y}$, but we impose the convention that $\frac{\partial \ell}{\partial \boldsymbol{y}}$ is a *row* vector while $\nabla \boldsymbol{y}$ is a column vector. Therefore, we have

$$\frac{\partial \ell}{\partial \boldsymbol{y}^\top} = \nabla \boldsymbol{y}$$

where $\top$ is used to denote transpose. Under this convention, the 1st-order Taylor expansion can be written as

$$\ell[\boldsymbol{w}] \approx \ell[\mathbf{w}_0] + \frac{\partial \ell}{\partial \boldsymbol{w}}[\mathbf{w}_0]\big(\boldsymbol{w} - \mathbf{w}_0\big).$$

Also in this paper, an input data point to DNNs is exclusively denoted by $\mathfrak{x}$, and considered as having no internal structure (i.e., $\mathfrak{x}$ can be anything, not necessarily a tensor). And we assume that the training and validation datasets are both finite, and drawn from the same distribution $\mathcal{D}$.

B.1 PROOF OF THEOREM 3

Technically, we prove Theorem 3 by taking Taylor expansions of the DNN. The "difficulty" is that Taylor expansions can only be taken for the *point-wise* loss $\ell[\mathfrak{x}; \mathcal{W}]$, for a given data point $\mathfrak{x}$; While the Random Feature model requires a universal constant matrix $\mathbf{A}$. Our proof shows that this is still possible, by "concatenating" all the data points in the training set into a huge linear map, exploiting the high dimensionality $\mathtt{V}$ of the latent space.

Proof of Theorem 3. For *(a)*, since the Jacobian matrix $\mathbf{J}[\mathfrak{x}] \overset{\text{def}}{=} \frac{\partial \mathbf{h}}{\partial \mathcal{W}^{\mathrm{F}}}[\mathfrak{x}; \mathcal{W}_t^{\mathrm{F}}]$ depends on $\mathfrak{x}$ but does not depend on s, we can construct a matrix $\tilde{\mathbf{A}}$ by concatenating the $\mathbf{J}[\mathfrak{x}]$ (on axis 1) for all training examples $\mathfrak{x} \in \mathcal{D}$, and a vector $\tilde{\mathbf{x}}$ by padding the flattened vector $(\mathcal{W}_{t+s}^{\mathrm{F}} - \mathcal{W}_t^{\mathrm{F}})$ with zeros, to obtain

$$\Big(\frac{\partial \mathbf{h}}{\partial \mathcal{W}^{\mathrm{F}}}[\mathfrak{x}; \mathcal{W}_t^{\mathrm{F}}]\Big)(\mathcal{W}_{t+s}^{\mathrm{F}} - \mathcal{W}_t^{\mathrm{F}}) = \tilde{\mathbf{A}}\tilde{\mathbf{x}}$$

where $\tilde{\mathbf{A}}$ does not depend on $\mathfrak{x}$ or s. Then, we define $\mathbf{A}$ by further concatenating $\tilde{\mathbf{A}}$ with an identity matrix (on axis 1), and $\mathbf{x}$ by concatenating $\tilde{\mathbf{x}}$ with $\mathbf{h}_t$, so that $\mathbf{h}_{t+s}$ has the 1st-order Taylor expansion

$$\mathbf{h}_{t+s} \approx \mathbf{h}_t + \Big(\frac{\partial \mathbf{h}}{\partial \mathcal{W}^{\mathrm{F}}}[\mathfrak{x}; \mathcal{W}_t^{\mathrm{F}}]\Big)(\mathcal{W}_{t+s}^{\mathrm{F}} - \mathcal{W}_t^{\mathrm{F}}) = \mathbf{A}\mathbf{x},$$

as required.

For *(b)*, recall from the previous paragraph that we have $\tilde{\mathbf{A}}\mathbf{x} \approx \mathbf{h}_{t+s} - \mathbf{h}_t$, and $\mathbf{h}_t$ is contained in $\mathbf{x}$, so we can write

$$\boldsymbol{y}_{t+s} - \boldsymbol{y}_t = \boldsymbol{W}_{t+s}(\mathbf{h}_{t+s} - \mathbf{h}_t) + (\boldsymbol{W}_{t+s} - \boldsymbol{W}_t)\mathbf{h}_t \approx (\boldsymbol{W}_{t+s}\tilde{\mathbf{A}} + \widetilde{\Delta \boldsymbol{W}})\mathbf{x}$$

where $\widetilde{\Delta \boldsymbol{W}}$ is $(\boldsymbol{W}_{t+s} - \boldsymbol{W}_t)$ padded with zeros (on axis 1), such that $\widetilde{\Delta \boldsymbol{W}}\mathbf{x} = (\boldsymbol{W}_{t+s} - \boldsymbol{W}_t)\mathbf{h}_t$. Without changing the calculation so far, we can further modify $\mathbf{x}$ by concatenating $(\mathcal{W}_{t+s}^{\mathrm{B}} - \mathcal{W}_t^{\mathrm{B}})$. Then, since both $\frac{\partial^2 \ell}{\partial \boldsymbol{y}^\top \partial \boldsymbol{y}}[\mathfrak{x}; \boldsymbol{y}_t, \mathcal{W}_t^{\mathrm{B}}]$ and $\frac{\partial^2 \ell}{\partial \boldsymbol{y}^\top \partial \mathcal{W}^{\mathrm{B}}}[\mathfrak{x}; \boldsymbol{y}_t, \mathcal{W}_t^{\mathrm{B}}]$ are matrices that depend on $\mathfrak{x}$ but do not depend on s, we concatenate them (on axis 1) for all training examples $\mathfrak{x} \in \mathcal{D}$ and define

$$\tilde{\boldsymbol{G}}_{t+s} \overset{\text{def}}{=} \mathbf{B}\Big(\bigoplus\nolimits_{k=1}^{\mathtt{n}}(\boldsymbol{W}_{t+s}\widetilde{\mathbf{J}_k} + \widetilde{\Delta \boldsymbol{W}})\Big) + \tilde{\mathbf{C}}$$

where $\mathbf{B}$ is the concatenation of $\frac{\partial^2 \ell}{\partial \boldsymbol{y}^\top \partial \boldsymbol{y}}[\mathfrak{x}; \boldsymbol{y}_t, \mathcal{W}_t^{\mathrm{B}}]$, and $\tilde{\mathbf{C}}$ the concatenation of $\frac{\partial^2 \ell}{\partial \boldsymbol{y}^\top \partial \mathcal{W}^{\mathrm{B}}}[\mathfrak{x}; \boldsymbol{y}_t, \mathcal{W}_t^{\mathrm{B}}]$. The $\bigoplus_{k=1}^{\mathtt{n}}$ denotes concatenation of matrices on axis 0, with $\mathtt{n} = \#\text{training-examples}$, and $\widetilde{\mathbf{J}_k}$ denotes the Jacobian matrix $\mathbf{J}[\mathfrak{x}^{(k)}]$ padded with zeros, for the k-th training example $\mathfrak{x}^{(k)} \in \mathcal{D}$. (Such

that, $\sum_{k=1}^{\mathtt{n}} \widetilde{\mathbf{J}_k} = \tilde{\mathbf{A}}$ for $\tilde{\mathbf{A}}$ in the previous paragraph). The matrices $\mathbf{B}$, $\tilde{\mathbf{C}}$, $\widetilde{\mathbf{J}_k}$ and $\widetilde{\Delta \boldsymbol{W}}$ do not depend on $\mathfrak{x}$ or s, while the construction of $\tilde{\boldsymbol{G}}_{t+s}$ implies

$$\Big(\frac{\partial^2 \ell}{\partial \boldsymbol{y}^\top \partial \boldsymbol{y}}[\mathfrak{x}; \boldsymbol{y}_t, \mathcal{W}_t^{\mathrm{B}}]\Big)(\boldsymbol{y}_{t+s} - \boldsymbol{y}_t) + \Big(\frac{\partial^2 \ell}{\partial \boldsymbol{y}^\top \partial \mathcal{W}^{\mathrm{B}}}[\mathfrak{x}; \boldsymbol{y}_t, \mathcal{W}_t^{\mathrm{B}}]\Big)(\mathcal{W}_{t+s}^{\mathrm{B}} - \mathcal{W}_t^{\mathrm{B}}) \approx \tilde{\boldsymbol{G}}_{t+s}\mathbf{x}.$$

Finally, we can modify $\mathbf{x}$ by concatenating with $\nabla \boldsymbol{y}_t$, and define $\mathbf{C}$ by concatenating $\tilde{\mathbf{C}}$ with an identity matrix (on axis 1) to absorb the constant term, and obtain

$$\boldsymbol{G}_{t+s} \stackrel{\text{def}}{=} \mathbf{B}\left(\bigoplus_{k=1}^{\mathtt{n}} (\boldsymbol{W}_{t+s}\widetilde{\mathbf{J}_k} + \widetilde{\Delta \boldsymbol{W}})\right) + \mathbf{C}$$

such that the 1st-order Taylor expansion of $\nabla \boldsymbol{y}_{t+s}$ is given by $\boldsymbol{G}_{t+s}\mathbf{x}$. By construction, $\boldsymbol{G}_{t+s}$ is an affine function of $\boldsymbol{W}_{t+s}$. □

B.2 Proof of Prediction 5

We have $\mathbf{h}_t \approx \mathbf{A}\mathbf{x}$ where the entries of $\mathbf{x}$ are i.i.d. Gaussian of mean 0 and variance 1, by Assumption 4. So

$$\mathbb{E}(\mathbf{h}_t \mathbf{h}_t^\top) \approx \mathbb{E}(\mathbf{A}\mathbf{x}\mathbf{x}^\top \mathbf{A}^\top) = \mathbf{A}\mathbf{A}^\top.$$

Therefore, the eigen decomposition of $\mathbb{E}(\mathbf{h}_t \mathbf{h}_t^\top)$ comes from the singular value decomposition

$$\mathbf{A} = \mathbf{U}\,\mathbf{Diag}(\sqrt{\boldsymbol{\lambda}})\mathbf{V}^\top,$$

where $\sqrt{\boldsymbol{\lambda}}$ is the entry-wise square-root of $\boldsymbol{\lambda}$, and $\mathbf{V}$ is a $\mathtt{V} \times \mathtt{D}$ matrix with orthonormal columns.

Now, $\boldsymbol{g}_i$ is the i-th column vector of the $\mathtt{C} \times \mathtt{D}$ matrix

$$(\nabla \boldsymbol{W}_{t+s})\mathbf{U} = (\nabla \boldsymbol{y}_{t+s})\,\mathbf{h}_{t+s}^\top \mathbf{U} \approx \boldsymbol{G}_{t+s}\mathbf{x}\,\mathbf{x}^\top \mathbf{A}^\top \mathbf{U} = \boldsymbol{G}_{t+s}\mathbf{x}\,\mathbf{x}^\top \mathbf{V}\,\mathbf{Diag}(\sqrt{\boldsymbol{\lambda}}),$$

So

$$\boldsymbol{g}_i \approx \sqrt{\lambda_i}\boldsymbol{G}_{t+s}\mathbf{x}\,\mathbf{x}^\top \mathbf{V}\mathbf{e}_i \tag{11}$$

where λ_i is the i-th entry of $\boldsymbol{\lambda}$ and $\mathbf{e}_i$ is the i-th unit vector ($1 \le i \le \mathtt{D}$).

On the other hand, we have $\nabla \boldsymbol{y}_{t+s} \approx \boldsymbol{G}_{t+s}\mathbf{x}$ so

$$\mathbb{E}(\nabla \boldsymbol{y}_{t+s} \nabla \boldsymbol{y}_{t+s}^\top) \approx \mathbb{E}(\boldsymbol{G}_{t+s}\mathbf{x}\mathbf{x}^\top \boldsymbol{G}_{t+s}^\top) = \boldsymbol{G}_{t+s}\boldsymbol{G}_{t+s}^\top. \tag{12}$$

Thus, Prediction 5 follows by combining the following lemma with (11) and (12).

Lemma 7. *Assume $\mathbf{V}$ is "general enough". Then, we have*

$$\mathbb{E}\big(\mathbf{x}\,\mathbf{x}^\top \mathbf{V}\mathbf{e}_i \mathbf{e}_j{}^\top \mathbf{V}^\top \mathbf{x}\,\mathbf{x}^\top\big) = \delta_{ij}\mathbf{I}_{\mathtt{V}}$$

at the $\mathtt{D} \to \infty$ limit. Here, δ_{ij} is the Kronecker delta and $\mathbf{I}_{\mathtt{V}}$ is the $\mathtt{V} \times \mathtt{V}$ identity matrix.

Proof. The (k, n)-entry of the matrix $\mathbf{x}\,\mathbf{x}^\top \mathbf{V}\mathbf{e}_i \mathbf{e}_j{}^\top \mathbf{V}^\top \mathbf{x}\,\mathbf{x}^\top$ is a linear combination of random variables of the form

$$x_k x_l x_m x_n,$$

where x_k, x_l, x_m, x_n are entries of $\mathbf{x}$. Now recall Wick's formula (Isserlis's theorem), which implies that in order to have

$$\mathbb{E}(x_k x_l x_m x_n) \neq 0,$$

there are only three possibilities:

1. $k = n, l = m$;
2. $k = m, l = n$;
3. $k = l, m = n$.

When restricted to Case 1, only the diagonal entries (such that $k = n$) in $\mathbb{E}\big(\mathbf{x}\,\mathbf{x}^\top \mathbf{V}\mathbf{e}_i \mathbf{e}_j{}^\top \mathbf{V}^\top \mathbf{x}\,\mathbf{x}^\top\big)$ are nonzero, and they equal to $\mathbf{e}_j{}^\top \mathbf{V}^\top \mathbf{V}\mathbf{e}_i$ which is δ_{ij} because the column vectors of $\mathbf{V}$ are orthonormal. So this case results in a $\delta_{ij}\mathbf{I}_{\mathtt{V}}$ term.

For both Cases 2 and 3, they result in terms of the form $\mathbf{V}\mathbf{e}_i \mathbf{e}_j{}^\top \mathbf{V}^\top$, which are projectors that map any $\mathtt{V}$-dimensional vector into a 1-dimensional subspace (i.e., $\mathbf{V}\mathbf{e}_i$) of the $\mathtt{D}$-dimensional space spanned by the column vectors of $\mathbf{V}$. When $\mathbf{V}$ is "general enough", this is like a random orthogonal projection from a $\mathtt{D}$-dimensional space onto its 1-dimensional subspace; So "on average" these terms are of the magnitude $1/\sqrt{\mathtt{D}}$, which become negligible compared to $\mathbf{I}_{\mathtt{V}}$ when $\mathtt{D}$ is sufficiently large.

Rigorously, this "general enough" argument can be conveyed by assuming a Haar measure over the space of $\mathbf{V}$ and bounds the error with "overwhelming probability", as similar to Paquette et al. (2021, Appendix C). □

Although our proof of Lemma 7 relies on specifics of the Gaussian distribution (such as Wick's formula), the conclusion on high dimension limit is likely to hold for other random vectors (asymptotically) of mean 0 and variance 1.

B.3 Equivalent Batch Factor

In this section, we discuss how to apply Prediction 5 to the calculation of 2nd moment of *mini-batched* gradients of *DNN*s.

For stochastic training with batch-size $\mathtt{B}$, we use $\nabla_{\mathtt{B}}$ to denote the *per-batch* gradient. Then, instead of (11), we have

$$\boldsymbol{g}_i^{\mathtt{B}} \approx \frac{1}{\mathtt{B}} \sum_{k=1}^{\mathtt{B}} \sqrt{\lambda_i} \boldsymbol{G}_{t+s} \mathbf{x}^{(k)}\, \mathbf{x}^{(k)^\top} \mathbf{V}\mathbf{e}_i \tag{13}$$

where $\boldsymbol{g}_i^{\mathtt{B}}$ is the i-th column vector of the $\mathtt{C} \times \mathtt{D}$ matrix $(\nabla_{\mathtt{B}} \boldsymbol{W}_{t+s})\mathbf{U}$, and $\mathbf{x}^{(k)}$ the k-th data point in a batch. Note the normalization factor $1/\mathtt{B}$ which is to make $\mathbb{E}(\nabla \boldsymbol{W}) = \mathbb{E}(\nabla_{\mathtt{B}} \boldsymbol{W})$. We can generalize Lemma 7 to this mini-batched setting, as below.

Lemma 8. *Assume $\mathbf{V}$ is "general enough", and $\mathtt{B}^2 \ll \mathtt{D}$. Then, we have*

$$\mathbb{E}\Big(\sum_{k=1}^{\mathtt{B}} \big(\mathbf{x}^{(k)}\, \mathbf{x}^{(k)^\top}\big) \mathbf{V}\mathbf{e}_i \mathbf{e}_j{}^\top \mathbf{V}^\top \sum_{k=1}^{\mathtt{B}} \big(\mathbf{x}^{(k)}\, \mathbf{x}^{(k)^\top}\big)\Big) = \mathtt{B}\, \delta_{ij} \mathbf{I}_{\mathtt{V}}$$

at the $\mathtt{D} \to \infty$ limit. Here, δ_{ij} is the Kronecker delta and $\mathbf{I}_{\mathtt{V}}$ is the $\mathtt{V} \times \mathtt{V}$ identity matrix.

Proof. The proof is mostly similar to Lemma 7, except that for Case 3, we have $\mathtt{B}^2$ many terms of the form $\mathbf{V}\mathbf{e}_i \mathbf{e}_j{}^\top \mathbf{V}^\top$, due to cross-interaction inside the batch. Therefore, an additional condition $\mathtt{B}^2 \ll \mathtt{D}$ is required to ensure that the batch-size is not too large, so that Case 3 is still negligible. □

Therefore, we can calculate

$$\mathbb{E}(\boldsymbol{g}_i^{\mathtt{B}} \boldsymbol{g}_j^{\mathtt{B}^\top}) \approx \frac{1}{\mathtt{B}}\, \mathbb{E}(\boldsymbol{g}_i \boldsymbol{g}_j^\top) \tag{14}$$

as long as the batch-size $\mathtt{B}$ is not too large ($\mathtt{B} \ll \sqrt{\mathtt{D}}$). This formula is compatible with the intuition that increasing the batch-size will decrease gradient noise.

Now, regarding $\mathbb{E}(\nabla \boldsymbol{y}_{t+s} \nabla \boldsymbol{y}_{t+s}^\top)$, for Random Feature models where y_{t+s} is a scalar we have

$$\nabla y_{t+s} = c\big(\boldsymbol{w}_{t+s}^\top \mathbf{A}\mathbf{x} - b[\mathbf{x}]\big)$$

hence

$$\mathbb{E}\big((\nabla y_{t+s})^2\big) = \mathbb{E}\Big(c^2 \big(\boldsymbol{w}_{t+s}^\top \mathbf{A}\mathbf{x} - b[\mathbf{x}]\big)^2\Big) = 2c\, \mathbb{E}\big(\ell[\mathbf{x}; \boldsymbol{w}_{t+s}]\big),$$

which is a multiple of the validation loss.

However, the above simplification does *not* hold for DNNs in general. Even if we use Assumption 6 to reduce y_{t+s} to a scalar, the *point-wise* loss of DNNs is of the form

$$\ell[\mathfrak{x}; \boldsymbol{w}] \approx \frac{c[\mathfrak{x}]}{2} \big(\boldsymbol{w}^\top \mathbf{A}\mathbf{x} - b[\mathfrak{x}]\big)^2 \quad \text{and} \quad \nabla y_{t+s} \approx c[\mathfrak{x}] \big(\boldsymbol{w}_{t+s}^\top \mathbf{A}\mathbf{x} - b[\mathfrak{x}]\big). \tag{15}$$

Especially, the coefficient c in (15) is different across data points $\mathfrak{x}$.

Nevertheless, in our simulation we estimate c as an average of $c[\mathfrak{x}]$, such that the Hessian trace of the layer-wise quadratic loss $L[\boldsymbol{w}_t]$ matches the empirical measurement $\mathrm{Tr}\left(\mathbb{E}(\nabla^2 \boldsymbol{W}_t)\right)$. And we still assume that $\mathbb{E}\big((\nabla y_{t+s})^2\big)$ is a multiple of $2cL[\boldsymbol{w}_{t+s}]$, as a rough approximation. But the coefficient of that multiple needs to be empirically measured, and it would be much larger than 1 (as in contrast to the case of Random Feature models) because $\mathbb{E}(c[\mathfrak{x}]^2)$ is usually much larger than $\mathbb{E}(c[\mathfrak{x}])^2$. In this work, we absorb this multiple into a "batch-size factor", because it affects the norm of per-batch gradients as similar to a batch-size.

In practice, we do not directly measure $L[\boldsymbol{w}_t]$ either, because the measurement of the Hessian tensor is costly if not infeasible. Instead, we have $\|\frac{\partial L}{\partial \boldsymbol{w}}[\boldsymbol{w}_t]\|^2 = \|\mathbb{E}(\nabla \boldsymbol{W}_t)\|^2$ by definition, and $\|\mathbb{E}(\nabla \boldsymbol{W}_t)\|^2$ is directly measurable. So it is favorable if $L[\boldsymbol{w}_t]$ can be approximated by $\|\frac{\partial L}{\partial \boldsymbol{w}}[\boldsymbol{w}_t]\|^2$.

Recall that $L[\boldsymbol{w}]$ is a quadratic function of $\boldsymbol{w}$ which can be decomposed into eigenmodes. Let

$$L[\boldsymbol{w}] = \frac{c}{2}\sum_{i=1}^{\mathtt{D}} \lambda_i (w_i - \mathtt{w}_i^*)^2$$

be the eigen decomposition, where $(w_i)_{1\le i\le \mathtt{D}}$ and $(\mathtt{w}_i^*)_{1\le i\le \mathtt{D}}$ are representations of $\boldsymbol{w}$ and $\mathbf{w}^*$ under the eigen basis, respectively. Then, we have

$$\frac{\partial L}{\partial \boldsymbol{w}^\top}[\boldsymbol{w}] = \big(c\lambda_i(w_i - \mathtt{w}_i^*)\big)_{1\le i\le \mathtt{D}} \quad \text{and} \quad \Big\|\frac{\partial L}{\partial \boldsymbol{w}}[\boldsymbol{w}]\Big\|^2 = c^2\sum_{i=1}^{\mathtt{D}} \lambda_i^2 (w_i - \mathtt{w}_i^*)^2.$$

Thus, the following quantity

$$\frac{\sum_{i=1}^{\mathtt{D}} \lambda_i}{\sum_{i=1}^{\mathtt{D}} \lambda_i^2}\big\|\mathbb{E}(\nabla \boldsymbol{W}_t)\big\|^2 \tag{16}$$

would be an easy-to-measure approximation for $2cL[\boldsymbol{w}_t]$. This approximation should work well if either of the conditions below is satisfied:

(1) For most i, values of $(w_i - \mathtt{w}_i^*)^2$ are approximately of the same magnitude;
(2) Or, the values of $(w_i - \mathtt{w}_i^*)^2$ are "random enough", the distribution of (λ_i) not too heavy-tailed, and $\mathtt{D}$ sufficiently large.

To summarize, we have assumed that $\mathbb{E}\big((\nabla y_{t+s})^2\big)$ is a multiple of $2cL[\boldsymbol{w}_{t+s}]$, and propose to use (16) as an approximation for $2cL[\boldsymbol{w}_t]$. Combining this with (14) and Prediction 5, we predict

$$\mathfrak{B}\,\mathbb{E}(\|\nabla_{\mathtt{B}} \boldsymbol{W}_{t+s}\|^2) \approx \sum_{i=1}^{\mathtt{D}} \lambda_i\, 2cL[\boldsymbol{w}_{t+s}] \tag{17}$$

where $\mathfrak{B}$ is empirically measured as

$$\mathfrak{B} \overset{\text{def}}{=} \frac{(\sum_{i=1}^{\mathtt{D}} \lambda_i)^2}{\sum_{i=1}^{\mathtt{D}} \lambda_i^2}\frac{\|\mathbb{E}(\nabla \boldsymbol{W}_t)\|^2}{\mathbb{E}(\|\nabla_{\mathtt{B}} \boldsymbol{W}_t\|^2)}. \tag{18}$$

We call $\mathfrak{B}$ the *equivalent batch factor*, because it should coincide with the batch-size $\mathtt{B}$ for Random Feature models.

B.4 Detailed Derivation of Algorithm 1

We track the vector $\boldsymbol{z} = (z_i)_{1\le i\le \mathtt{D}}$ where $z_i \overset{\text{def}}{=} c(w_i - \mathtt{w}_i^*)$, with $(w_i)_{1\le i\le \mathtt{D}}$ and $(\mathtt{w}_i^*)_{1\le i\le \mathtt{D}}$ being the representations of $\boldsymbol{w}$ and $\mathbf{w}^*$ under the eigen basis, respectively. Since

$$L[\boldsymbol{w}] = \frac{1}{2c}\sum_{i=1}^{\mathtt{D}} \lambda_i z_i^2,$$

the sign of each z_i does not matter – our simulation only depends on the absolute value $|z_i|$. Initially, $|z_i|$ is set to match the Root Mean Square (RMS) of the gradient $\mathbb{E}(\nabla \boldsymbol{W}_t)$:

- The coordinates of $\mathbb{E}(\nabla \boldsymbol{W}_t)$ are changed to the eigen basis by $\mathbb{E}(\nabla \boldsymbol{W}_t)\mathbf{U}$;
- The shape of $\mathbb{E}(\nabla \boldsymbol{W}_t)\mathbf{U}$ is $\mathtt{C} \times \mathtt{D}$; We reduce the $\mathtt{C}$-dimensional output axis by taking $\mathrm{RMS}\big(\mathbb{E}(\nabla \boldsymbol{W}_t)\mathbf{U},\, \mathtt{axis} = 0\big)$;
- The i-th entry of $\mathrm{RMS}\big(\mathbb{E}(\nabla \boldsymbol{W}_t)\mathbf{U},\, \mathtt{axis} = 0\big)$ should be the same as

$$\Big|\frac{\partial L}{\partial w_i}[\boldsymbol{w}]\Big| = \lambda_i |z_i|,$$

  in order to match the gradient at each eigenmode.
- Therefore, we set

$$\boldsymbol{z} \leftarrow \mathrm{RMS}\big(\mathbb{E}(\nabla \boldsymbol{W}_t)\mathbf{U},\, \mathtt{axis} = 0\big)/\boldsymbol{\lambda}$$

  where "/" denotes element-wise division.

At each simulation step, let $\boldsymbol{g} = (g_k)_{1 \le k \le \mathtt{D}}$ be the synthetic per-batch gradient under the eigen basis. The mean $\mathbb{E}(g_k)$ is given by

$$\mathbb{E}(g_k) = \frac{\partial L}{\partial w_k}[\boldsymbol{w}] = \lambda_k z_k, \tag{19}$$

and the 2nd moment

$$\mathbb{E}(g_k^2) \approx \lambda_k \, 2cL[\boldsymbol{w}]/\mathfrak{B} = \lambda_k \sum_{i=1}^{\mathtt{D}} \lambda_i z_i^2/\mathfrak{B} \tag{20}$$

according to Prediction 5 and (17). Therefore, the variance is calculated as

$$\mathbb{VAR}[g_k] = \mathbb{E}(g_k^2) - \big(\mathbb{E}(g_k)\big)^2 = \lambda_k \Big(\sum_{i=1}^{\mathtt{D}} \lambda_i z_i^2/\mathfrak{B} - \lambda_k z_k^2 \Big). \tag{21}$$

Our algorithm simply draws g_k as Gaussian of mean $\mathbb{E}(g_k)$ and variance $\mathbb{VAR}[g_k]$ as given in (19) and (21), respectively.

Once the synthetic per-batch gradient $\boldsymbol{g}$ is obtained, we convert it to the superficial coordinates by $\mathbf{U}\boldsymbol{g}$. Then we apply the optimizer $\phi[\bullet]$, and convert back to the coordinates under eigen basis again:

$$(\text{update to } \boldsymbol{w}) = \boldsymbol{u} = \mathbf{U}^\top \phi[\mathbf{U}\boldsymbol{g}].$$

Our algorithm finishes this simulation step by updating $\boldsymbol{z} = c(\boldsymbol{w} - \mathbf{w}^*)$ accordingly.

B.5 Derivation of Learning-rate Threshold

We analyze the dynamics of validation loss $\mathbb{E}\big(\ell[\mathbf{x}; \boldsymbol{w}_t]\big)$ for Random Feature models defined in (1), by taking eigen decomposition of the Hessian

$$\mathbf{H} \overset{\text{def}}{=} \mathbb{E}\Big(\frac{\partial^2 \ell}{\partial \boldsymbol{w}^\top \partial \boldsymbol{w}}[\mathbf{x}; \boldsymbol{w}_t]\Big) = c\,\mathbb{E}(\mathbf{A}\mathbf{x}\mathbf{x}^\top \mathbf{A}^\top).$$

Let $\nabla_{\mathtt{B}} \boldsymbol{w}_t$ be the mini-batched gradient of batch-size $\mathtt{B}$. Let $g_i^{\mathtt{B}}$, $w_{i,t}$ and $\mathtt{w}_i^*$ be the projections of $\nabla_{\mathtt{B}} \boldsymbol{w}_t$, $\boldsymbol{w}_t$ and $\mathbf{w}^*$ onto the i-th eigenmode, respectively. The SGD update is given by

$$w_{i,t+1} = w_{i,t} - \gamma\, g_i^{\mathtt{B}},$$

where γ is the learning-rate. Now, recall that $\mathbb{E}(\nabla_{\mathtt{B}} \boldsymbol{w}_t) = \mathbb{E}(\nabla \boldsymbol{w}_t) = \mathbf{H}(\boldsymbol{w}_t - \mathbf{w}^*)$; so we have

$$(w_{i,t+1} - \mathtt{w}_i^*)^2 = (w_{i,t} - \mathtt{w}_i^*)^2 - 2\gamma\,(w_{i,t} - \mathtt{w}_i^*) g_i^{\mathtt{B}} + \gamma^2 (g_i^{\mathtt{B}})^2 \tag{22}$$

$$\text{and} \quad \mathbb{E}\big((w_{i,t+1} - \mathtt{w}_i^*)^2\big) = (1 - \gamma\lambda_i)^2 (w_{i,t} - \mathtt{w}_i^*)^2 + \gamma^2 \Big(\mathbb{E}\big((g_i^{\mathtt{B}})^2\big) - \big(\mathbb{E}(g_i^{\mathtt{B}})\big)^2 \Big), \tag{23}$$

where $\{\lambda_i\}$ is the eigenvalues of $\mathbf{H}$. In order to derive the threshold of learning-rate between convergence and divergence, we make the intuitive assumption that when the learning-rate γ is at the threshold, the variance of gradients is neither decreasing nor increasing, i.e., it remains a constant independent of the step t:

$$\mathbb{E}\big((g_i^{\mathtt{B}})^2\big) - \big(\mathbb{E}(g_i^{\mathtt{B}})\big)^2 \equiv V_i \neq 0. \tag{24}$$

For more formal and rigorous treatment using the kernel of a Volterra equation, we refer to Paquette et al. (2024, C.2). Substituting (24) into (23), we immediately get

$$\lim_{s\to\infty} \mathbb{E}\big((w_{i,t+s} - \mathtt{w}_i^*)^2\big) = \sum_{s=0}^{\infty} (1-\gamma\lambda_i)^{2s}\gamma^2 V_i = \frac{\gamma}{\lambda_i(2-\gamma\lambda_i)} V_i. \tag{25}$$

Furthermore, when $\mathtt{D} = \dim(\boldsymbol{w})$ is sufficiently large and $\mathtt{B} \ll \sqrt{\mathtt{D}}$, we have

$$\mathbb{E}\big((g_i^{\mathtt{B}})^2\big) = \frac{2}{\mathtt{B}}\lambda_i\, \mathbb{E}\big(\ell[\mathbf{x}; \boldsymbol{w}_t]\big) = \frac{1}{\mathtt{B}}\lambda_i \sum_k \lambda_k (w_{k,t} - \mathtt{w}_k^*)^2 \tag{26}$$

by combining (14), (5) and (7). We also have $\big(\mathbb{E}(g_i^{\mathtt{B}})\big)^2 \ll \mathbb{E}\big((g_i^{\mathtt{B}})^2\big)$ under this condition, based on a similar argument as in Lemma 8. So V_i can be explicitly calculated, therefore (25) implies

$$\lim_{s\to\infty} \mathbb{E}\big((w_{i,t+s} - \mathtt{w}_i^*)^2\big) = \frac{\gamma}{\lambda_i(2-\gamma\lambda_i)} V_i = \frac{\gamma}{\mathtt{B}(2-\gamma\lambda_i)} \sum_k \lambda_k (w_{k,t} - \mathtt{w}_k^*)^2. \tag{27}$$

Then, by summing across all i-th eigenmodes in (27), we obtain:

$$\lim_{s\to\infty} \mathbb{E}\big(\sum_i \lambda_i (w_{i,t+s} - \mathtt{w}_i^*)^2\big) = \sum_i \frac{\gamma\lambda_i}{\mathtt{B}(2-\gamma\lambda_i)} \sum_k \lambda_k (w_{k,t} - \mathtt{w}_k^*)^2. \tag{28}$$

Thus the following equation regarding the threshold γ, by taking the limit at $t \to \infty$:

$$1 = \sum_i \frac{\gamma\lambda_i}{\mathtt{B}(2-\gamma\lambda_i)}. \tag{29}$$

B.6 Concentration Effect Only Holds for $\alpha \le 0.5$

We need to analyze the random process (22), instead of the average-case behavior (23). The difference between (22) and (23) is a martingale; When one sums the martingales from all eigenmodes, the "*concentration effect*" states that the sum converges to 0 with "overwhelming probability", under a suitable limit at $\mathtt{D} = \dim(\boldsymbol{w}) \to \infty$.

We refer to Paquette et al. (2021) for the formal tools and rigorous arguments regarding convergence of martingales. In this section, we give an intuitive sketch of the essence of the idea.

Recall the arguments in § 3.4, that $\boldsymbol{w}_t$ can be decomposed into $\boldsymbol{w}_t = \overline{\boldsymbol{w}}_t + \boldsymbol{r}_t$ where $\overline{\boldsymbol{w}}_t$ follows the deterministic gradient descent, while $\boldsymbol{r}_t$ is an accumulation of random noises of mean $\mathbf{0}$. At each step, a new draw of gradient noise is added into $\boldsymbol{r}_t$, and *on average* it contributes to the i-th eigenmode of $\mathbb{E}\big((w_{i,t+1} - \mathtt{w}_i^*)^2\big)$ by $\gamma^2\Big(\mathbb{E}\big((g_i^{\mathtt{B}})^2\big) - \big(\mathbb{E}(g_i^{\mathtt{B}})\big)^2\Big)$, according to (23).

From the calculation of (26), the variance $\Big(\mathbb{E}\big((g_i^{\mathtt{B}})^2\big) - \big(\mathbb{E}(g_i^{\mathtt{B}})\big)^2\Big)$ is proportional to λ_i; so we can write the *random* contribution of the gradient noise to the i-th eigenmode of $(w_{i,t+1} - \mathtt{w}_i^*)^2$, by $\lambda_i X_i^2$ where X_i is a random variable, independent and of the same variance across all i-th eigenmodes.

Now, by summing across all eigenmodes, the gradient noise contributes to the random validation loss $\sum_i \lambda_i (w_{i,t+1} - \mathtt{w}_i^*)^2$ by $\gamma^2 \sum_i \lambda_i^2 X_i^2$. The concentration effect states that

$$\gamma^2 \sum_{i=1}^{\mathtt{D}} \lambda_i^2 X_i^2 \approx \gamma^2 \sum_{i=1}^{\mathtt{D}} \lambda_i^2 \mathbb{E}(X_i^2)$$

with high probability, at the limit $\mathtt{D} \to \infty$.

Note that, in order for the model to converge, the learning-rate γ should be less than the threshold which satisfies Equation (29), meaning $\gamma < 2\mathtt{B}/\sum_i \lambda_i$. To make the limit $\mathtt{D} \to \infty$ properly defined, γ should actually scale in proportion to $1/\sum_i \lambda_i$. Thus, we obtain the following

$$\frac{\sum_{i=1}^{\mathtt{D}} \lambda_i^2 X_i^2}{\big(\sum_{i=1}^{\mathtt{D}} \lambda_i\big)^2} \tag{30}$$

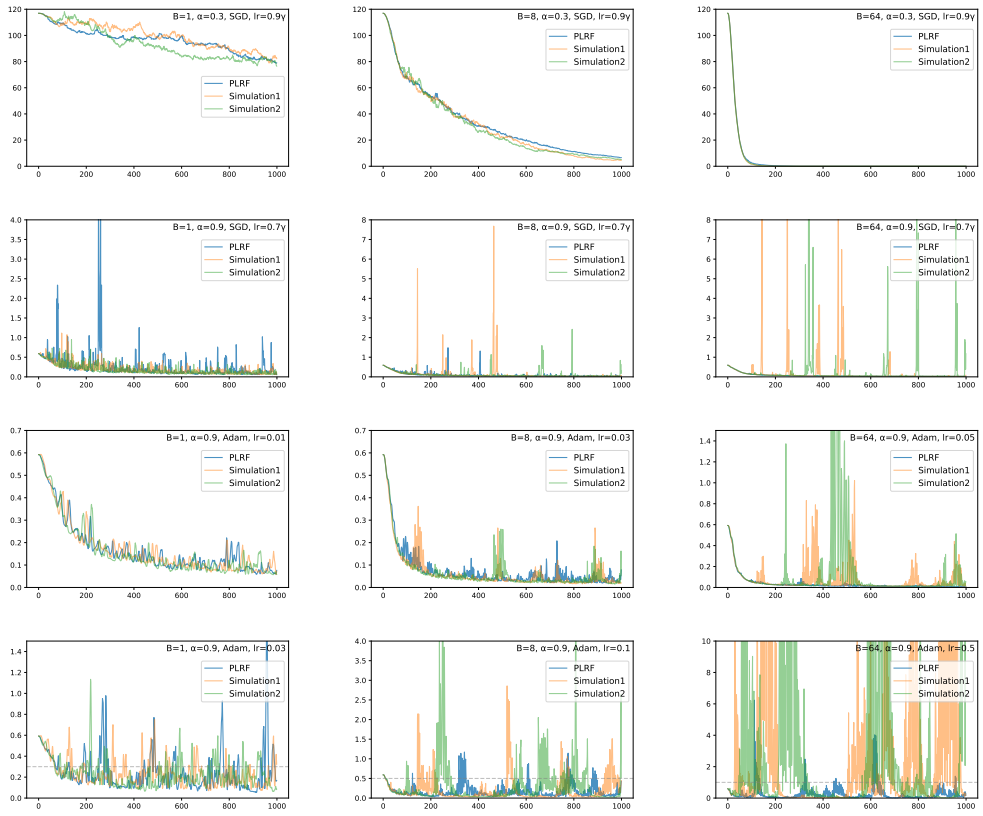


Figure C.1: Simulating PLRF by Algorithm 1. We apply SGD and Adam at different batch-sizes. Each plot includes one PLRF training and two simulation runs (with different random seeds).

and note that $\mathbb{E}(X_i^2)$ scales approximately in proportion to $\sum_{i=1}^{\mathtt{D}} \lambda_i$. So (30) can be viewed as an average of $\{\lambda_i X_i^2/(\sum_{i=1}^{\mathtt{D}} \lambda_i)\}$, weighted by the probability measure $\big(\lambda_i/(\sum_{i=1}^{\mathtt{D}} \lambda_i)\big)$.

For PLRF models with power-law exponent α, the eigenvalues (λ_i) are asymptotically proportional to $(i^{-2\alpha})$. Then, assuming $\{X_i\}$ are i.i.d. random variables, and $X_i^2/(\sum_{i=1}^{\mathtt{D}} \lambda_i)$ has finite expectation and variance, the concentration to expectation is a standard argument by the Law of Large Numbers, that the random fluctuations of (30) will cancel out, if and only if

$$\lim_{\mathtt{D}\to\infty} \sum_{i=1}^{\mathtt{D}} \lambda_i = \infty.$$

This is equivalent to $\alpha \leq 0.5$.

C PLRF Simulation Experiments

As a sanity check, we apply our simulation algorithm to PLRF models, and provide the Python code in supplementary materials. The PLRF models are defined the same as in Figure 2.1, while we train with SGD and Adam, at batch-sizes $\mathtt{B} = \{1, 8, 64\}$. Each configuration is simulated by two runs of different random seeds.

Figure C.1 shows the results. We observe that the simulation correctly captures the average-case behavior of the validation loss: the underlying "flows" match each other as long as we ignore the spikes. Moreover, the simulation qualitatively captures the spiky behavior: Simulation is spiky if and only if the actual PLRF training trajectory is spiky.

	B=1,LR=0.03,TH=0.3			B=8,LR=0.1,TH=0.5			B=64,LR=0.5,TH=1.0		
	Q90	RAT	MRLAT	Q90	RAT	MRLAT	Q90	RAT	MRLAT
PLRF	0.89	0.18	8.2	1.1	0.051	2.2	3.8	0.035	1.3
Simulation1	0.59	0.14	4.9	1.4	0.096	2.7	22.	0.38	4.6
Simulation2	0.53	0.18	5.1	2.5	0.19	3.7	44.	0.36	6.9

Table 2: Statistics of training spikes, for three configurations of batch-sizes $\texttt{B} = \{1, 8, 64\}$, with respectively picked learning-rate (LR) and threshold (TH).

However, the simulation does not always correctly predict every statistics of training spikes: It seems that the simulation tends to underestimate the "largeness" of spikes when $\texttt{B} = 1$, while significantly overestimate it when B is large.

This tendency is quantitatively documented in Table 2: For each of the three configurations in the last row of Figure C.1, we pick a threshold (shown as dashed lines in the figure) and empirically measure the following statistics to reflect the nature of spikes:

- **Quantile at 90% (Q90)**: The 90th percentile of loss values larger than the threshold at steps ≥ 200. This measures how large the spikes are.
- **Ratio Above Threshold (RAT)**: The ratio of loss values larger than the threshold, for steps ≥ 200. This measures how frequently the spikes occur.
- **Mean Run Length Above Threshold (MRLAT)**: Count the number of consecutive steps (≥ 200) where loss values exceed the threshold before they drop back down. Then, take the average of these lengths. This measures the average duration of a spike.

We observe that for $\texttt{B} = 1$, simulations have slightly lower Q90; while for $\texttt{B} = 64$, all statistics (Q90, RAT and MRLAT) become much larger, indicating that the simulations are overestimating spikes. We believe the reasons are due to: (1) The distribution of *per-example* gradients of PLRF is not Gaussian; (2) Our estimate of the 2nd moment of *per-batch* gradients, as used in the simulation, is only asymptotically accurate when the batch-size is small.

D Transformer LM Experiments

In this section, we give details of our case-study of a Transformer LM training.

D.1 Model and Training Settings

The model is a vanilla decoder-only Transformer, with Self-Attention (Vaswani et al., 2017), Pre-Layernorm (Xiong et al., 2020), Rotary Position Encoding (Su et al., 2024) and QK-layernorm (Dehghani et al., 2023) for each layer. We build our experiments upon an existing series of model sizes used in previous research (Everett et al., 2024; Charles et al., 2025), where the sizes of model components are parameterized by a single factor k, as below:

- Sequence length: 2048.
- Vocab size: 32000.
- Number of blocks: $12 * k$.
- Number of attention heads: $16 * k$.
- Dimension of attention heads: 64.
- Dimension of model hidden vectors = (Attention head dimension) $*$ (Number of attention heads) = $64 * 16 * k = 1024 * k$.
- MLP intermediate dimension = $4 *$ (Hidden dimension) = $4096 * k$.

As detailed in Everett et al. (2024), the training hyperparameters are found by sweeping smaller models and verified by fitting a power-law curve. We take one point $k = 2$ for our case-study.

For this point, the hyperparameters are given as below.

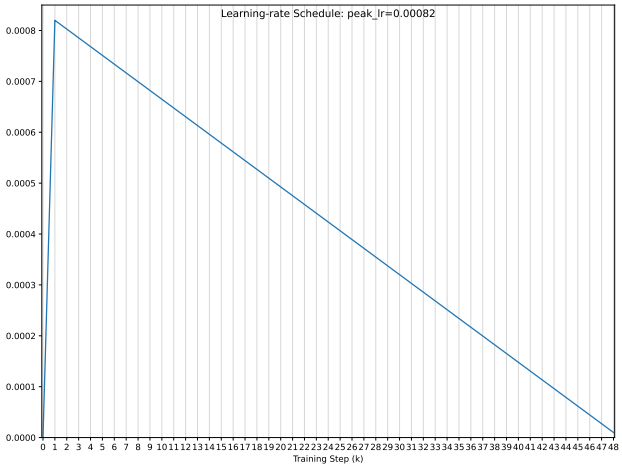


Figure D.1: The learning-rate schedule used for our Transformer LM training.

- Optimizer: AdamW (Loshchilov & Hutter, 2019) with $\beta_1 = 0.9$, $\beta_2 = 0.95$. We do not apply any gradient clipping to the training.
- Batch-size (number of sequences per batch): Set to 256.
- Number of training steps: Determined by the Chinchilla rule (Hoffmann et al., 2022), i.e., (Total number of tokens trained) / (Tokens per batch) = $(20 * \text{Number of model parameters})$ / $(256 \times 2048) \approx 48500$.
- Learning-rate schedule: We use a simple schedule with warm-up in 1000 steps, then linearly decay to 0. The peak learning-rate is 0.00082. The schedule is shown in Figure D.1.
- Weight-decay: Set to 0.2.

The model is trained with cross-entropy loss normalized by the number of tokens per batch. We note this normalization factor is applied to gradient – the loss should be considered *per-token* cross-entropy without any normalization factor; while the *per-batch* gradient is summed across all tokens in a batch, so we normalize it by the number of tokens per batch, in order to ensure that the *average gradient* over a dataset does not depend on the batch-size of the calculation.

This convention of normalization becomes critical when we discuss and distinguish between the squared norms of *per-example* gradient $\|\nabla \boldsymbol{W}_t\|^2$ and *per-batch* gradient $\|\nabla_{\mathtt{B}} \boldsymbol{W}_t\|^2$. Here, $\|\nabla \boldsymbol{W}_t\|^2$ denotes the squared norm of gradient over *one token*, and $\mathbb{E}(\|\nabla \boldsymbol{W}_t\|^2)$ is averaged across *all tokens* in a dataset. In contrast, $\|\nabla_{\mathtt{B}} \boldsymbol{W}_t\|^2$ denotes the squared norm of gradient of *a batch*, and $\mathbb{E}(\|\nabla_{\mathtt{B}} \boldsymbol{W}_t\|^2)$ takes average across *all batches* in a dataset. There is no obvious relation between $\mathbb{E}(\|\nabla \boldsymbol{W}_t\|^2)$ and $\mathbb{E}(\|\nabla_{\mathtt{B}} \boldsymbol{W}_t\|^2)$, because the square in $\|\nabla_{\mathtt{B}} \boldsymbol{W}_t\|^2$ is taken *after* summing over the batch! Throughout this paper we will carefully distinguish, prove and verify relations between $\mathbb{E}(\|\nabla \boldsymbol{W}_t\|^2)$ and $\mathbb{E}(\|\nabla_{\mathtt{B}} \boldsymbol{W}_t\|^2)$.

Empirically, the calculation of $\mathbb{E}(\|\nabla \boldsymbol{W}_t\|^2)$ can still be batched, because we have the following

$$(\nabla \boldsymbol{W}_t)^{\odot 2} = (\nabla \boldsymbol{y}_t)^{\odot 2} \, \mathbf{h}_t^{\odot 2^\top}, \tag{31}$$

where $^{\odot 2}$ denotes element-wise square. Therefore, by taking squares of $\nabla \boldsymbol{y}_t$ and $\mathbf{h}_t$ first, one can obtain results of squares *before* summing over the batch, while still batching the whole calculation and summing across all tokens.

In order to calculate the Hessian trace $\mathrm{Tr}\left(\mathbb{E}(\nabla^2 \boldsymbol{W}_t)\right)$, we use the fact that for any random vector $\boldsymbol{v} = (v_i)$, where $\{v_i\}$ are independent, of mean 0 and variance 1, we have

$$\mathrm{Tr}(\mathbf{H}) = \mathbb{E}(\boldsymbol{v}^\top \mathbf{H} \boldsymbol{v}). \tag{32}$$

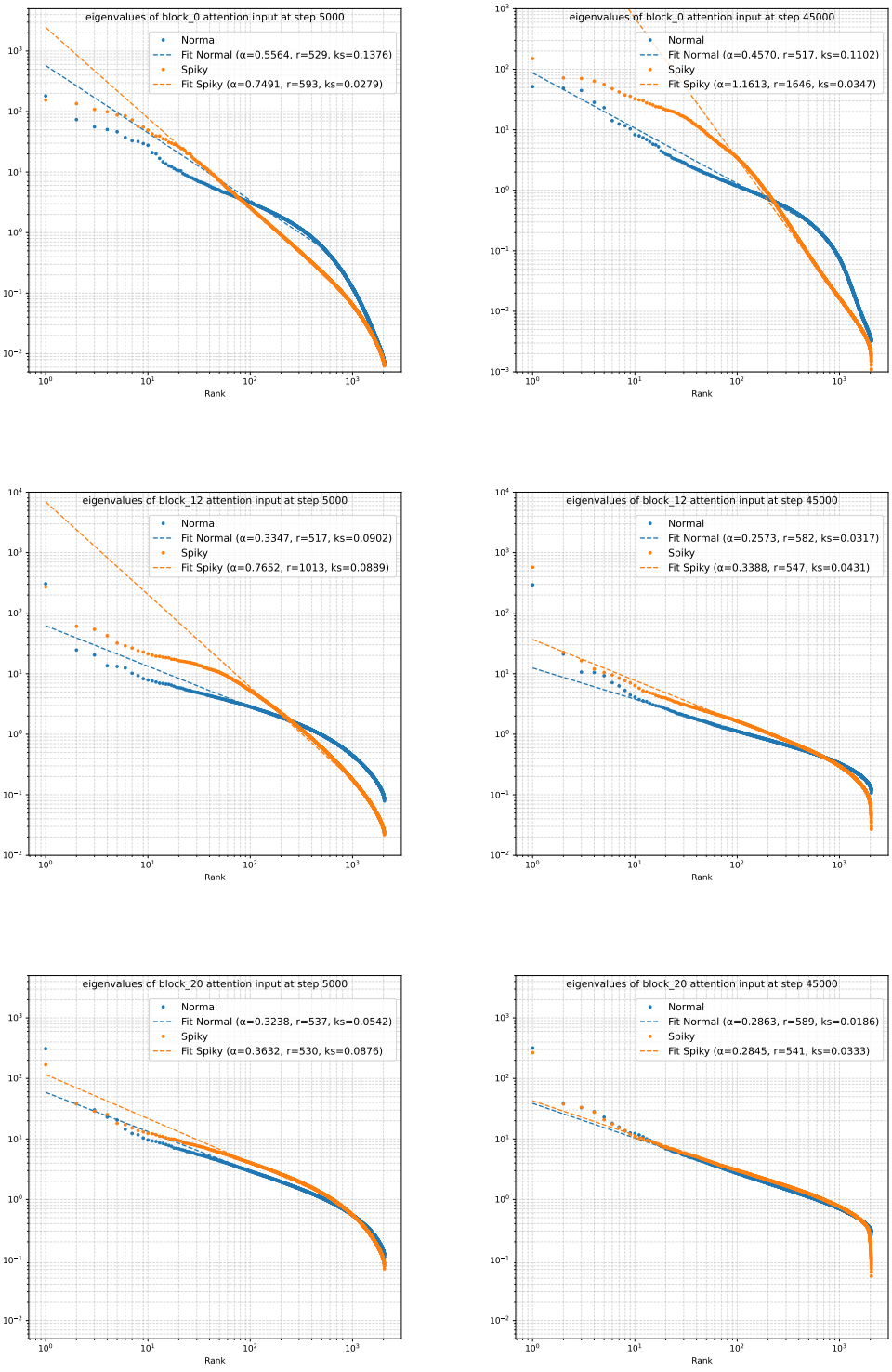


Figure D.2: Log-log plot with truncated power-law fit, for ranked eigenvalues of correlation matrices.

Therefore, instead of directly calculating the huge tensor $\mathbf{H}$, we only need the Hessian form $\boldsymbol{v}^\top \mathbf{H} \boldsymbol{v}$ of a few random vectors. In our experiments, we randomly sample 6 vectors, with each entry drawn from $\{\pm 1\}$. Preliminary experiments suggest this setting has relative error < 0.1.

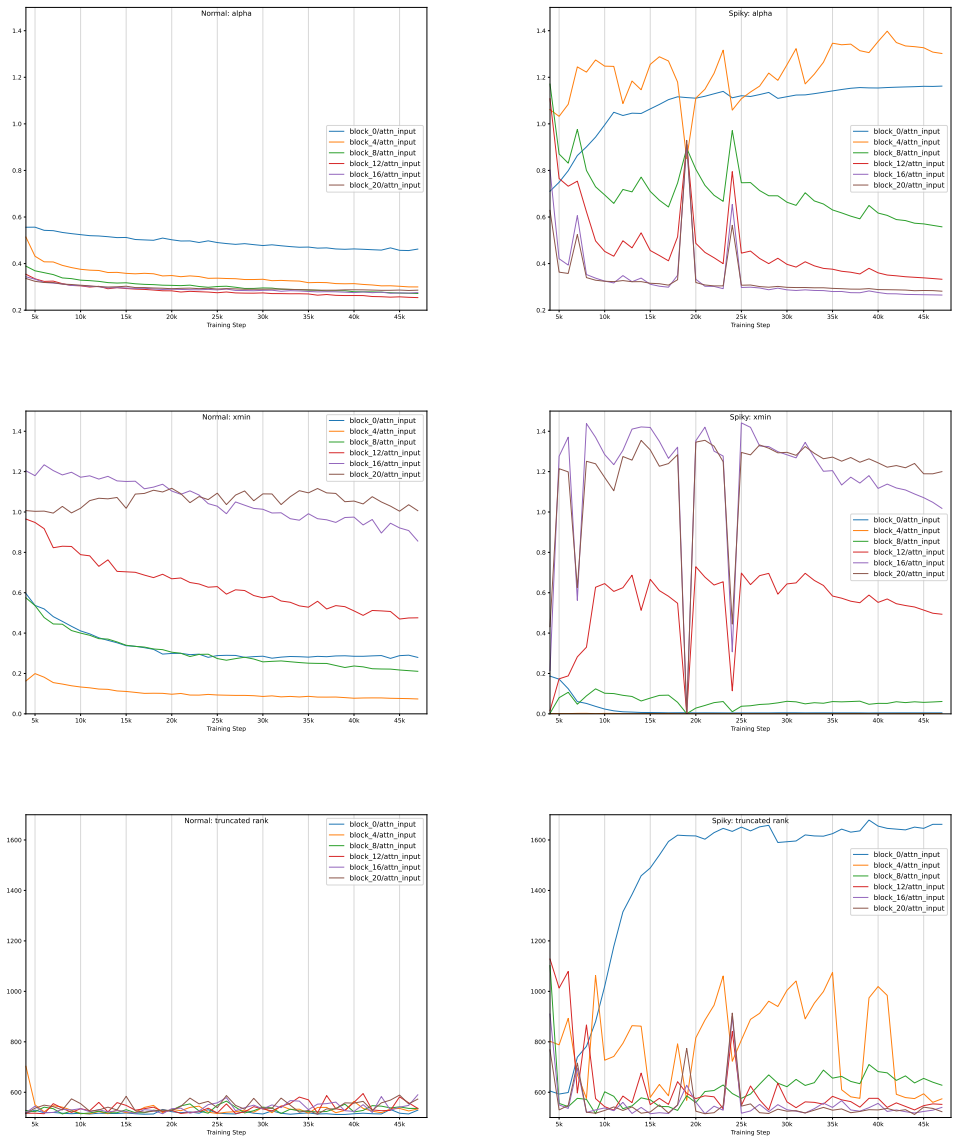


Figure D.3: Evolution of power-law fitting statistics, across layers and training steps.

The Hessian form $\boldsymbol{v}^\top \mathbf{H}\boldsymbol{v}$ can be calculated from Jacobian-Vector Products (a.k.a. forward-mode autodiff). With some implementation techniques, one can even calculate the layer-wise Hessian form for multiple layers in one forward pass, by concatenating intermediate results from each layer along the way of autodiff.

D.2 Power-Law Distribution of Forward Activations

Let $\mathbf{h}_t$ be the input hidden vector to a layer. In order for the layer to exhibit training spikes, our theory suggests that the eigenvalues of $\mathbb{E}(\mathbf{h}_t\mathbf{h}_t^\top)$ should follow a heavy-tailed distribution. In this section, we verify that it is indeed the case for our Transformer LM training.

In Figure D.2, we sort the eigenvalues of $\mathbb{E}(\mathbf{h}_t\mathbf{h}_t^\top)$ in descending order, and show the log-log plots of eigenvalues over ranks, for different layers and training steps. The Normal and Spiky1 training configurations are compared in each sub-plot.

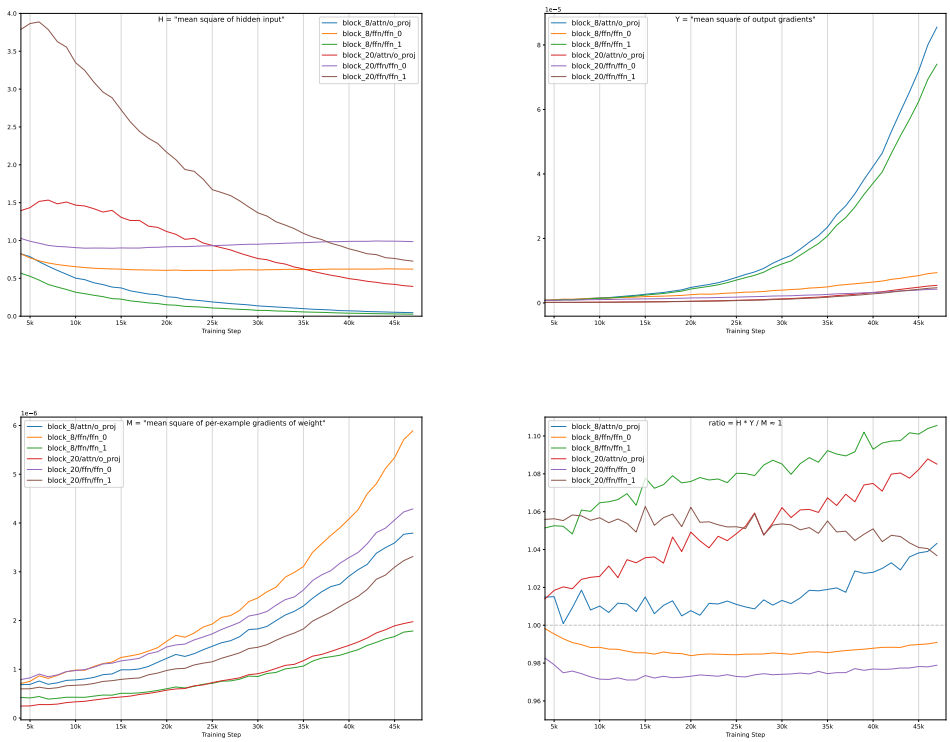


Figure D.4: Empirical verification of Equation (6).

In the log-log plots, eigenvalues indeed exhibit power-law like distributions, whereas the Spiky training generally shows steeper descending and crosses Normal. In other words, eigenvalues from Spiky are more heavy-tailed.

Quantitatively, we follow (Clauset et al., 2009) to fit truncated power-law distributions to the log-log plots, which estimates the power-law exponent by maximizing the likelihood, and estimates the truncation threshold by minimizing the Kolmogorov–Smirnov (KS) statistic. The estimated power-law exponent α, truncation rank r, and the KS statistic are shown in Figure D.2.

For lower layers (e.g., `block_0`), the eigenvalues distribution of Spiky seems significantly different from Normal, and the gap remains unclosed until the end. For upper layers (e.g., `block_20`) the distributions are more similar, and they seem to converge to the same towards the end of training (e.g., `block_{12,20}`).

This tendency is documented in Figure D.3, where we plot the evolution of power-law fitting statistics along the whole training process. Generally, Spiky training exhibits larger power-law exponent α, larger truncation threshold $x_{\min}$ for upper layers and smaller for lower layers, and larger truncated rank r especially for lower layers. The evolution of the statistics is also more unstable for Spiky.

D.3 Empirical Verification of Prediction 5

In Prediction 5, we have derived a formula to calculate the norm of per-example gradients, given as Equation (6). We empirically verify the formula in this section.

Recall that layer $\boldsymbol{W}$ is a $\mathtt{C} \times \mathtt{D}$ matrix, the input $\mathbf{h}_t$ a $\mathtt{D}$-dimensional, and the output gradient $\nabla \boldsymbol{y}_t$ a $\mathtt{C}$-dimensional vector. In Figure D.4, we plot:

(a) $\mathfrak{H} \overset{\text{def}}{=} \sum_{i=1}^{\mathtt{D}} \lambda_i / \mathtt{D} = \mathbb{E}(\|\mathbf{h}_t\|^2)/\mathtt{D}$, the mean square of hidden inputs.

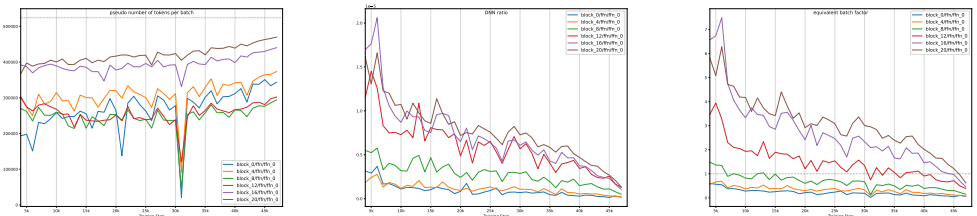


Figure D.5: Breakdown of the equivalent batch factor.

(b) $\mathfrak{Y} \stackrel{\text{def}}{=} \mathbb{E}(\|\nabla \boldsymbol{y}_t\|^2)/\mathtt{C}$, the mean square of output gradients.

(c) $\mathfrak{M} \stackrel{\text{def}}{=} \mathbb{E}(\|\nabla \boldsymbol{W}_t\|^2)/(\mathtt{C}\times\mathtt{D})$, the mean square of per-example gradients of the model weight.

(d) The ratio $\mathfrak{H}\mathfrak{Y}/\mathfrak{M}$, which should be close to 1 according to Equation (6).

We show different types of layers across training steps. The norm of gradients can vary multiple magnitudes during training, yet the ratio plot in Figure D.4 shows that the relative error of Equation (6) is less than 0.1, mostly within 5%.

D.4 Is the "Batch-size" Large or Small?

Specific to the training of Transformer LMs, there is the question of what should be considered as the "batch-size". Is it the number of tokens per batch, because each token is corresponding to a summand in the training loss function? Or is it the number of sequences per batch, because tokens within a sequence can have correlations, and the unit of sequence is what is guaranteed to be independent in a batch? Since recent LMs are trained with very long sequences ($\geq$10k), the answer to this question affects our understanding of whether the training of Transformer LMs should be theoretically considered as having a "large" or "small" batch-size.

The difference between "large" and "small" batch-sizes is drastic in theory: If the batch-size is small (compared to $\sqrt{\mathtt{D}}$, where $\mathtt{D}$ is the hidden dimension of the model), the gradient noise plays an important role in stochastic optimization, of which the effect can be accurately analyzed as we demonstrated in this work; On the other hand, our theoretical calculation becomes less accurate as the batch-size increases, and when the batch-size is extremely large as close to a full-batch setting, the theory regarding full-batch gradient descent becomes more relevant.

In this section, we provide a perspective from the *equivalent batch factor* concept. From (17), equivalent batch factor $\mathfrak{B}$ directly affects the largeness of gradient noise; In Figure D.5 (Right), we plot $\mathfrak{B}$ for different layers and training steps, as empirically measured according to (18). We observe that $\mathfrak{B}$ is small (close to 1), which means that we are in a typical stochastic optimization regime.

Furthermore, $\mathfrak{B}$ can be decomposed into three factors as below.

$$\mathfrak{B} = \left(\frac{\mathbb{E}(\|\nabla \boldsymbol{W}_t\|^2)}{\mathbb{E}(\|\nabla_{\mathtt{B}} \boldsymbol{W}_t\|^2)}\right)\left(\frac{\sum_{i=1}^{\mathtt{D}} \lambda_i \mathbb{E}(\|\nabla \boldsymbol{y}_t\|^2)}{\mathbb{E}(\|\nabla \boldsymbol{W}_t\|^2)}\right)\left(\frac{\sum_{i=1}^{\mathtt{D}} \lambda_i}{\sum_{i=1}^{\mathtt{D}} \lambda_i^2}\frac{\|\mathbb{E}(\nabla \boldsymbol{W}_t)\|^2}{\mathbb{E}(\|\nabla \boldsymbol{y}_t\|^2)}\right).$$

The first factor, $\mathbb{E}(\|\nabla \boldsymbol{W}_t\|^2)/\mathbb{E}(\|\nabla_{\mathtt{B}} \boldsymbol{W}_t\|^2)$, is plotted in Figure D.5 (Left), which turns out to be slightly smaller than the number of tokens per batch (shown as dashed line in the figure), which is not completely a surprise in light of (14). For recent LMs, this is a very large number.

The second factor, $\sum_{i=1}^{\mathtt{D}} \lambda_i \mathbb{E}(\|\nabla \boldsymbol{y}_t\|^2)/\mathbb{E}(\|\nabla \boldsymbol{W}_t\|^2)$, is what we have investigated in § D.3 and Figure D.4, which is close to 1 as predicted.

The third factor is what makes $\mathfrak{B}$ small. It measures how far a DNN model deviates from (7) and (16), which is satisfied by Random Feature models and would make this factor ≈ 1. In contrast, the empirical measure in our case turns out to be very small, as shown in Figure D.5 (Middle). It suggests that the point-wise loss of DNNs brings very different gradients to each training example, and constitutes the major source of gradient noise in training of DNNs.

$\beta_2 = 0.95$	$\beta_2 = 0.982$	$\beta_2 = 0.992$	β_2 schedule
2.4421	2.4405	2.4502	**2.4404**

Table 3: Final validation loss for different β_2 settings.

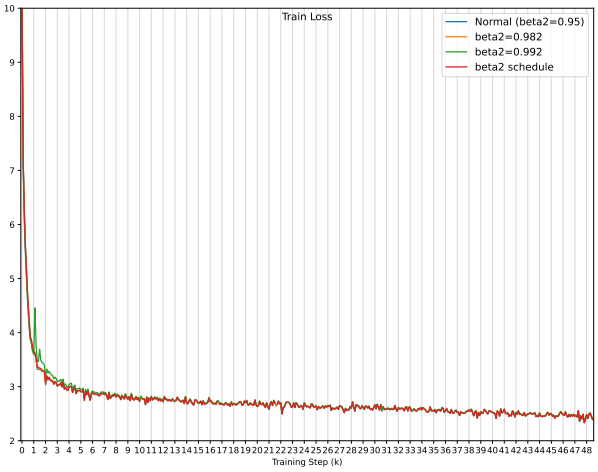


Figure D.6: Training loss for different β_2 settings.

Somehow magically, the first factor and the third factor cancel each other to make $\mathfrak{B}$ close to 1. If this is not a coincidence (since the training recipe we adopted in § D.1 has been tuned on batch-size as well), it might suggest heuristics for tuning the batch-size hyperparameter: Because the first factor depends on the batch-size while the third factor does not, one can empirically measure the third factor and tune the batch-size to make $\mathfrak{B} \approx 1$. We leave this direction to future work.

D.5 The Appropriate β_2

For our case-study, we know that the hyperparameter β_2 is causing training spikes. Is there a recipe to derive the appropriate β_2, possibly from empirical metrics, so that one can avoid the risk of spiky training? In this section, we propose and verify such a recipe.

The proposal is to set

$$\beta_2 \approx 1 - \sqrt{\frac{\eta_t \mathfrak{T}}{\sqrt{\mathfrak{N}}}} \quad \text{where} \quad \mathfrak{T} \overset{\text{def}}{=} \operatorname{Tr}\big(\mathbb{E}(\nabla^2 \boldsymbol{W}_t)\big)/(\mathtt{C} \times \mathtt{D}) \text{ and } \mathfrak{N} \overset{\text{def}}{=} \mathbb{E}(\|\nabla_{\mathtt{B}} \boldsymbol{W}_t\|^2)/(\mathtt{C} \times \mathtt{D}). \quad (33)$$

Here, η_t is the learning-rate schedule for Adam; We empirically measure $\mathfrak{T}$ the normalized Hessian trace and $\mathfrak{N}$ the mean square of per-batch gradients from validation data. In particular, (33) depends on the learning-rate schedule η_t; So if one applies a decaying schedule to the learning-rate, it usually suggests that β_2 should follow an increasing schedule.

The rationale for (33) comes from (10): Because the accumulated gradient noise $\boldsymbol{r}_t$ is decayed by $(\mathbf{I} - \eta_t \mathbf{H})$ at each training step, the random fluctuations of the loss dynamics (i.e. the source of training spikes) has self-correlation within $\approx 1/(\eta_t \mathbf{H})$ steps; Then, our proposal is to ensure that $\boldsymbol{\nu}$, the running average of gradient squares used as an optimizer state in Adam, has self-correlation within $\approx 1/\sqrt{\eta_t \mathbf{H}}$ steps. Since the learning-rate η_t in (10) is for SGD, we have to replace it by the adaptive learning-rate $\eta_t/\sqrt{\boldsymbol{\nu}}$ in Adam; We approximate the tensor $\sqrt{\boldsymbol{\nu}}$ by $\sqrt{\mathfrak{N}}$, and the tensor $\mathbf{H}$ by $\mathfrak{T} = \operatorname{Tr}(\mathbf{H})/\dim(\boldsymbol{w})$.

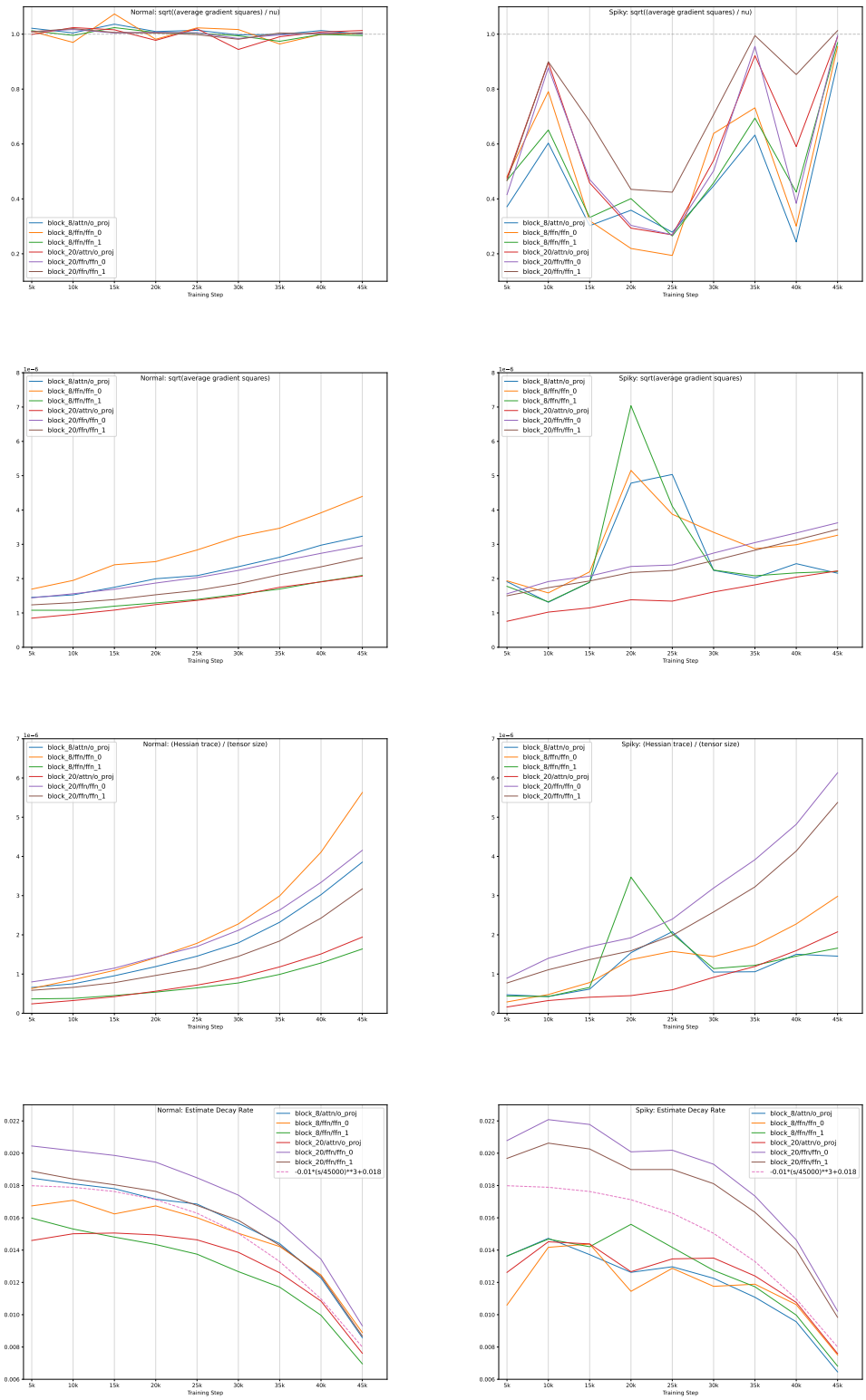


Figure D.7: Estimating an appropriate β_2 schedule.

In Figure D.7, we first ensure that $\sqrt{\mathfrak{N}}$ is actually a good approximation of $\sqrt{\boldsymbol{\nu}}$ (or vice versa): We plot the ratio $\sqrt{\mathfrak{N}/\operatorname{mean}(\boldsymbol{\nu})}$ in the First Row, for different layers and training steps, comparing the Normal and Spiky configurations. We found that the approximation works well for Normal training, but much worse for Spiky. This is not surprising because the large β_2 used by Spiky is likely to make any surge of gradient squares last longer in $\boldsymbol{\nu}$, causing it to overestimate the average.

Then, we plot $\sqrt{\mathfrak{N}}$, $\mathfrak{T}$ and finally $\sqrt{\eta_t \mathfrak{T}/\sqrt{\mathfrak{N}}}$ in the Second, Third and Fourth Row of Figure D.7, respectively. Although the plots of $\sqrt{\mathfrak{N}}$ and $\mathfrak{T}$ can be spiky, we observe that the overall tendency of the ratio $\sqrt{\eta_t \mathfrak{T}/\sqrt{\mathfrak{N}}}$ is comparatively stable and consistent across all layers, for both Normal and Spiky. In order to capture this tendency, we manually fit a schedule of the form below (shown as pink dashed curves in the Fourth Row of Figure D.7).

$$(\texttt{final_value} - \texttt{init_value}) \left(\frac{\text{step}}{\text{total_steps}} \right)^{\texttt{exponent}} + \texttt{init_value}, \tag{34}$$

with $\texttt{init_value} = 0.018$, $\texttt{final_value} = 0.008$ and $\texttt{exponent} = 3$.

We set $(1-\beta_2)$ to the schedule given in (34) and train a Transformer LM as in § D.1, without changing any other hyperparameters. We compare the final validation loss with the original setting $\beta_2 = 0.95$, and two additional settings $\beta_2 = (1 - \texttt{init_value}) = 0.982$ and $\beta_2 = (1 - \texttt{final_value}) = 0.992$. The results are shown in Table 3.

We observe that the proposed β_2 schedule can indeed improve performance in terms of validation loss. Remarkably, the performance significantly differs between $\beta_2 = (1 - \texttt{init_value})$ and $\beta_2 = (1 - \texttt{final_value})$, even if $\texttt{init_value}$ and $\texttt{final_value}$ only differ by 0.01. We also observe that the training of $\beta_2 = (1 - \texttt{final_value})$ is spiky, while $\beta_2 = (1 - \texttt{init_value})$ is not (Figure D.6). The proposed β_2 schedule can even improve a bit over $\beta_2 = (1 - \texttt{init_value})$, with the final value of β_2 actually becoming $(1 - \texttt{final_value})$.

D.6 Discussion about $\delta_{t,s}^{\boldsymbol{w}}$

We have plotted the 2nd-order term of $\delta_{t,s}^{\boldsymbol{w}}$, namely

$$\frac{1}{2}(\boldsymbol{w}_{t+s} - \boldsymbol{w}_t)^\top \mathbb{E}_{\mathfrak{x}\sim\mathcal{D}}\Big(\frac{\partial^2 \ell}{\partial \boldsymbol{w}^\top \partial \boldsymbol{w}}[\mathfrak{x}; \mathcal{W}_t] \Big)(\boldsymbol{w}_{t+s} - \boldsymbol{w}_t),$$

in Figure 4.1. Here, we note one big difference between our simulation and the empirical measurement of the 2nd-order terms of $\delta_{t,s}^{\boldsymbol{w}}$, is that the 2nd-order terms of $\delta_{t,s}^{\boldsymbol{w}}$ exhibit "jumps" rather than "spikes" (i.e., not resuming to the previous level after initial surge), which is also different from the behavior of the total validation loss as shown in Figure 1.1c. This suggests that the dynamics of the Taylor expansion $\delta_{t,s}^{\boldsymbol{w}}$ may not be fully explained by layer-wise quadratic models, as it cannot fully explain the total validation loss either. One may need to consider correlations across layers, and Taylor expansions beyond 2nd order.

In Figure D.8, we additionally plot the 1st order term of $\delta_{t,s}^{\boldsymbol{w}}$,

$$\mathbb{E}_{\mathfrak{x}\sim\mathcal{D}}\Big(\frac{\partial \ell}{\partial \boldsymbol{w}}[\mathfrak{x}; \mathcal{W}_t] \Big)(\boldsymbol{w}_{t+s} - \boldsymbol{w}_t),$$

as given in Definition 1. We use the same scale of y-axis (from -4 to 1) across all layers, and we compare the three training configurations (Normal, Spiky1 and Spiky2).

We found that the behavior of the 1st-order terms cannot be directly explained by layer-wise quadratic models either: First, note that the scale of y-axis in Figure D.8 is about $20\times$ the scale in Figure 4.1; the decrease of the 1st-order terms turns out to be *much faster* than one would expect from a layer-wise quadratic model – it seems that the highly non-linear nature of the DNN can somehow *accelerate* the feature-learning towards a loss-decreasing direction. On the other hand, the highly non-linear nature of the DNN must also produce *larger noises* than one would expect from a layer-wise quadratic model, because the overall validation loss of the DNN does not decrease this fast. Second, we observe that training spikes are reflected in the plots of 1st-order terms, as either "bumps" (block_12 of Spiky2), "jumps" (block_{8,16,20} of Spiky2), or "dives" (block_0 of Spiky2); but layer-wise quadratic models cannot exhibit similar behavior at this scale.

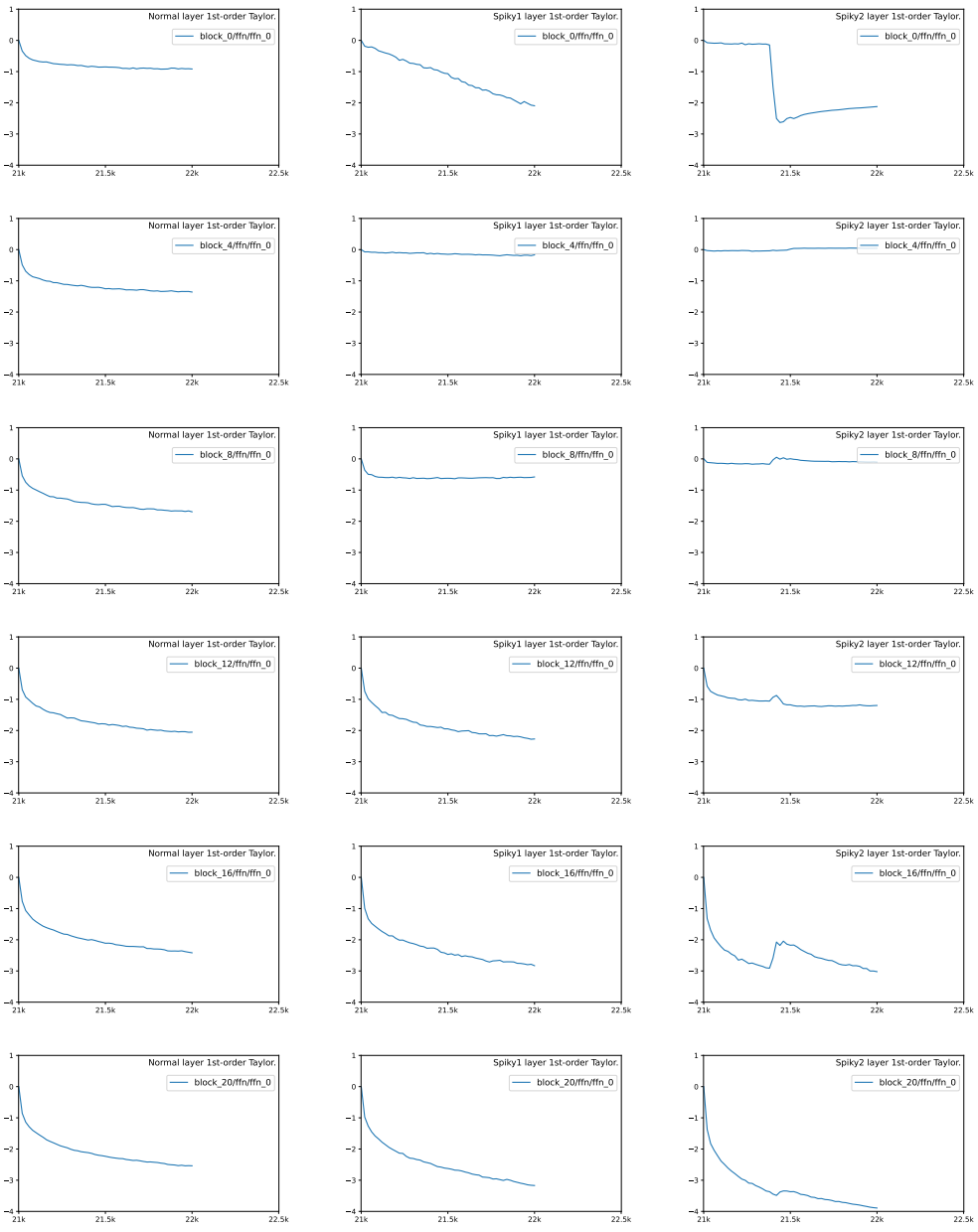


Figure D.8: Plots of the 1st-order term of $\delta^{\boldsymbol{w}}_{t,s}$.

How we can explain these discrepancies, and how the correlations across layers and Taylor expansions beyond 2nd order may play a role in the mechanism, are intriguing questions for future work.