What Does the Adam Optimizer Actually Adapt To?

by

Alan Milligan

B.Sc., The University of British Columbia, 2023

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)

September 2025

© Alan Milligan 2025

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

What Does the Adam Optimizer Actually Adapt To?

submitted by **Alan Milligan** in partial fulfillment of the requirements for the degree of **Master of Science** in **Computer Science**

Examining Committee:

Mark Schmidt, Professor, Department of Computer Science, UBC
Supervisor

Danica J. Sutherland, Assistant Professor, Department of Computer Science, UBC
Supervisory Committee Member

Abstract

As the impact of machine learning on all aspects of daily life grows, the importance of understanding why and how it works follows. Despite the success of recent machine learning based systems, several elements of the general machine learning pipeline remain poorly understand. This lack of understanding is no longer a question strictly for academics. Companies now spend millions of dollars training models, incurring massive energy and carbon costs. Without a solid understanding of the training process, predicting downstream model performance as a function of the choices made before training remains impossible. Many tricks have been discovered to address these challenges, but the tricks themselves remain poorly understood.

In particular, training machine learning models is framed as an optimization problem, but optimization theory lacks sufficient tools to analyze modern problems. Optimization theory often fails to explain why problems are hard or why algorithms are effective. A pointed example of this is the Adam optimization algorithm. Adam is widely successful and considered to be the default optimization algorithm in machine learning, but theory predicts it to be no better than classical algorithms like gradient descent. Adam stems from a line of "adaptive optimizers" that in some way adapt to the problem, but what they are adapting to is not clearly defined either. This thesis aims to highlight characteristics of optimization problems that Adam is addressing, and highlight how classical theoretical assumptions fail to explain Adam.

We isolate heavy-tailed class imbalance in language modelling as a characteristic that makes gradient descent fail, while Adam is unaffected. Further analysis shows that this characteristic leads to correlations between the gradients and Hessians of the model, a quality theorized to help Adam. Following this, we find that imbalanced features, as seen in a setting using graph neural networks, additionally cause gradient descent to fail while Adam remains effective. Finally, we further challenge existing theory, showing the performance of Adam can be both improved and destroyed by the choice of basis to run the optimization problem in. The majority of existing theory is invariant to the basis being used, and therefore fails to capture Adam's advantage.

Lay Summary

Machine learning based systems "learn" from data by iteratively trying to be "less wrong." These models contain, often millions or even billions of, "settings" known as parameters that are slightly adjusted throughout the training process. Concretely, each parameter is a number, and the goal is to tune those numbers long enough to get desirable behaviour of the model. The process of finding a good set of parameters is referred to as training or optimization. Many algorithms exist for optimization, but the most popular one in practice, Adam, remains poorly understood. We study this algorithm from multiple directions with the hope of isolating why it works so well in practice. By examining structure in data being used for learning, differences between Adam and better understood algorithms, and properties of the optimization problem itself, we hope to gain insights on what makes optimization problems hard and what makes Adam so effective.

Preface

The work presented in this thesis stems from the following published papers.

- Chapter 2 was published at NeurIPS 2024

  Frederik Kunstner, Alan Milligan, Robin Yadav, Mark Schmidt, and Alberto Bietti (2024). "Heavy-Tailed Class Imbalance and Why Adam Outperforms Gradient Descent on Language Models". *Neural Information Processing Systems*

  The initial idea of studying class imbalance came from discussions between Frederik and Alberto, and Robin and Frederik implemented initial experiments to support this claim. I substantially upgraded the codebase to handle larger experiments and designed new datasets and models to provide stronger evidence for our hypothesis. Frederik wrote the initial draft of the paper and he and I brought it through the publication process.

- Chapter 3 was published at the NeurIPS 2024 Workshop on Optimization

  Alan Milligan, Frederik Kunstner, Hamed Shirzad, Mark Schmidt, and Danica J. Sutherland (2024). "Normalization Matters for Optimization Performance on Graph Neural Networks". *NeurIPS Workshop on Optimization for Machine Learning*

  Hamed brought the idea to Frederik and me after Mark gave a talk including the content of Chapter 2. He introduced us to graph neural networks and typical problems involving them. I designed, implemented, and ran all experiments and found that the normalization method was having a substantial impact. I wrote the initial draft of the paper and Frederik and Danica assisted in editing the final form of the paper.

- Chapter 4 was published at NeurIPS 2025

  Tianyue H. Zhang, Lucas Maes, Alan Milligan Alexia Jolicoeur-Martineau, Ioannis Mitliagkas, Damien Scieur, Simon Lacoste-Julien, and Charles Guille-Escuret (2025). "Understanding Adam Requires Better Rotation Dependent Assumptions". *Neural Information Processing Systems*

  Charles provided the initial idea of studying how Adam performs under different rotations, and Tianyue and Lucas lead the project and did a majority of the experimentation and writing. I joined later on, and contributed all the experimentation and writing regarding using update orthogonality as a new metric, along with connections

between SVD rotations and Muon. Tianyue, Charles, and I were primarily involved in the publication process of the paper.

Generative AI tools were not used in the preparation of this thesis.

Contents

List of Tables

List of Figures

Acknowledgments

This thesis and the rest of my degree would not have been possible without the help of many people. To them I am grateful and indebted.

To my advisor Dr. Mark Schmidt, thank you for taking a chance on a lost undergraduate unsure what they wanted to do with their life. Your advice, both academic and otherwise, has always helped me figure out how to get through the many challenges that come with grad school and life. I'm fortunate to have had an advisor who is not only a very capable academic but also a good person.

To the faculty at UBC Computer Science, thank you for the years of teaching. In particular, I would like to thank my pseudo-cosupervisor Dr. Danica J. Sutherland. Thank you for taking extra time out of your already very busy schedule to help me when needed. Thank you for all your advice, suggestions, and help throughout my degree, I hope to one day know as much as you. I would also like to thank Dr. Margo Seltzer for her excellent job leading the department, thank you for your support of students.

To my previous advisors throughout my Bachelor's, thank you Dr. Dongwook Yoon and Dr. Kevin Leyton-Brown. Dongwook, thank you for introducing me to how computer science research works. Kevin, thank you for introducing me to how machine learning research works.

To my collaborators throughout my Bachelor's and Master's, thank you Anna Offenwanger, Chris Cameron, Jason Hartford, Robin Yadav, Hamed Shirzad, and Charles Guille-Escuret. Some of you first introduced me to what research was all about, some of you taught me about fields I was struggling to understand, all of you helped me become a more capable student.

To the people of ml-in-568 (past and present), thank you Frederik Kunstner, Tianyue (Helen) Zhang, Dylan Green, Betty Shea, Victor Sanches Portella, Wilder Lavington, Cathy Meng, Aaron Mishkin, Greg d'Éon, Sophie Greenwood, Rebecca Lin, Nick Ioannidis and many others for fostering an environment that felt like home. You've all had an impact on me one way or another. I've enjoyed watching each of you move forward with the next chapters of your lives, I suppose now it's my turn.

To the few remaining people outside of my academic life, thank you Kathryn Milligan, Emily Nold, Emily Ni, and any other friends and family for putting up with my lack of availability and even larger lack of interesting things to talk about.

In particular I would like to thank Frederik Kunstner and Helen Zhang.

To Fred, thank you for being a better research mentor and friend than I ever could have asked for. It is not an exaggeration to say that the majority of this thesis would not exist if not for you. More than that, you taught me how to be an effective grad student. Thank you for making me a better programmer, writer, thinker, and person. From the deadline pushes to the poster sessions, you've set the example for me of what a successful PhD looks like.

To Helen, thank you for staying by side through these years. Thank you for putting up with me, getting me out of the house (lab), and reminding me to have a life. Thank you for your optimism in the face of uncertainty, when I struggle to find mine. Thank you for helping me grow as a person. Thank you for occasionally eating white people food. Now we'll finally get to suffer through the Montreal winter together.

Chapter 1

Introduction

Over the past decade, machine learning based systems have come to have a substantial impact on daily life. Both at work and at home, machine learning underlies the "artificial intelligence" that helps people solve problems. Machine learning is used to recommend products to customers, control self-driving cars, propose new chemical compounds, and be useful chat assistants among countless other applications. In recent years, an unprecedented amount of time, energy, and capital has been invested in the development and operation of even more capable systems.

While many of the resulting innovations are impressive, the desire for downstream progress has taken priority over understanding why new innovations are actually effective. Classically, algorithms were developed with rigor and theoretical guarantees. That is, if a certain set of assumptions held, a meaningful statement could be made about how an algorithm will behave. Conversely, deep learning based systems typically offer no guarantees and are notoriously uninterpretable. Even early on in the success of deep learning, classically trained academics refereed to many new methods as alchemy. As that success has continued, the theory and practice of machine learning have continued to diverge. Successful methods are typically developed with a combination of vague intuitions and trial and error, to a point where "grad student descent" (a pun on gradient descent) is used to describe the development of new methods. This process is not without consequences, when methods fail or models produce harmful output, we do not know why. Given the prevalence machine learning has in modern society, these failures can lead to negative social impact, significant financial cost, and increasingly large carbon emissions.

Despite there being a significant gap between the theory and practice of modern machine learning, there remains significant effort in developing an understanding of why things work. This thesis contains a detailed look at one gap in particular, through three different lenses. At its core, the "learning" in machine learning is solving a mathematical optimization problem. In the past, the optimization problems being solved were well understood. Algorithms could provably make progress solving the problem and their solutions could be well characterized. However, the increasing scale and complexity of data, architectures, and tasks has lead to poorly understood optimization problems that are often solved with brittle, hand-tuned, heuristic algorithms. One of the most successful examples of these heuristics is the Adam optimization algorithm (Kingma and Ba, 2015). The work presented in this thesis aims to help explain this algorithm's (very) widespread dominance. Over the past decade it has

become the default choice for nearly all machine learning related optimization problems despite fundamental gaps in the understanding of why it works. Further understanding of algorithms like Adam guide the optimization community in developing more robust and principled algorithms.

1.1 Optimization Problems

Machine learning methods typically rely on solving an optimization problem, colloquially known as minimizing the loss. Intuitively, a measure of "how incorrect" a model is is defined (the *loss function*), and that "incorrectness" is minimized. We define n input-output pairs $(\mathbf{x}_i, y_i)$ as a dataset $\mathcal{D}$, and desire that our model m predict y_i as a function of $\mathbf{x}_i$ and the model's parameters $\mathbf{w}$. That is, for $i \in \{1, \ldots, n\}$, we desire $m(\mathbf{x}_i, \mathbf{w}) \approx y_i$. We refer to each $\mathbf{x}_i$ as an *example* or *sample*, and each y_i as a *label* or *target*. A loss function ℓ_i is a measure of how well the model is predicting the desired output for given example. Common examples are the squared loss for regression problems and the cross-entropy loss for classification problems. We write

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \ell_i(m(\mathbf{x}_i, \mathbf{w}), y_i)$$

to denote the loss function for the entire dataset, and aim to minimize it as a function of $\mathbf{w}$. The resulting set of weights is denoted as $\mathbf{w}^*$, the "optimal" set of weights for the model. As an optimization problem, this is written as

$$\mathbf{w}^* \in \arg\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}).$$

This framework includes a wide set of possible machine learning tasks for different choices of dataset, model, and loss function.

Example 1 (Linear Regression)**.** *Suppose our dataset consists of vectors $\mathbf{x}_i \in \mathbb{R}^d$ of real valued features and regression targets $y_i \in \mathbb{R}$. A linear regression model with weights $\mathbf{w} \in \mathbb{R}^d$ is written as $m(\mathbf{x}, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}$ with predicted regression targets $\hat{y}_i = \mathbf{w}^\top \mathbf{x}_i$. Using the squared loss $\ell(\hat{y}, y) = (\hat{y} - y)^2$, we end up with the optimization problem*

$$\arg\min_{\mathbf{w}} \sum_{i=1}^{n} (\mathbf{w}^\top \mathbf{x}_i - y_i)^2 = \arg\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

where $\mathbf{X}$ is a matrix with rows $\mathbf{x}_i^\top$ and $\mathbf{y}$ is a vector of each y_i.

Linear regression is a classic example of machine learning and its characteristics are well understood. It is an example of a smooth and convex optimization problem, properties which enable theory to predict the behavior of various optimization methods. However, this does not reflect the complex nature of modern optimization problems. Adding modest additional complexity departs from this well understood regime.

Example 2 (ReLU Classifier)**.** *Suppose our dataset consists of vectors* $\mathbf{x}_i \in \mathbb{R}^d$ *of real valued features and class labels* $\mathbf{y}_i \in \mathbb{R}^2$*. A neural network classifier with a Rectified Linear Unit (ReLU) activation and parameters* $\mathbf{W}_1 \in \mathbb{R}^{d\times h}$ *and* $\mathbf{W}_2 \in \mathbb{R}^{h\times 2}$ *can be written as* $m(\mathbf{x}, [\mathbf{W}_1,\ \mathbf{W}_2]) = \mathbf{W}_1\sigma(\mathbf{W}_2\mathbf{x})$ *where* $\sigma(\mathbf{z}) = \max\{0, \mathbf{z}\}$ *is taken element-wise. Typically, the output of this model* $m(\mathbf{x}, [\mathbf{W}_1,\ \mathbf{W}_2]) = \hat{\mathbf{z}}$ *is treated as an unnormalized log probability distribution over the classes. We exponentiate and normalize this distribution to get a probability over the classes* $\hat{\mathbf{y}} = \exp(\hat{\mathbf{z}})/\sum_{c=1}^{2}\exp(\hat{\mathbf{z}}_c)$ *and use the cross-entropy loss* $\ell_i(\hat{\mathbf{y}}, \mathbf{y}) = -\mathbf{y}_{i1}\log(\hat{\mathbf{y}}_{i1}) - \mathbf{y}_{i2}\log(\hat{\mathbf{y}}_{i2})$*. Including the normalization step, this leaves us with the optimization problem*

$$\arg\min_{[\mathbf{W}_1,\ \mathbf{W}_2]} -\sum_{i=1}^{n}\left(\mathbf{y}_{i1}\log\left(\frac{\exp(\hat{\mathbf{z}}_{i1})}{\sum_{c=1}^{2}\exp(\hat{\mathbf{z}}_{ic})}\right) + \mathbf{y}_{i2}\log\left(\frac{\exp(\hat{\mathbf{z}}_{i2})}{\sum_{c=1}^{2}\exp(\hat{\mathbf{z}}_{ic})}\right)\right).$$

Even in a relatively simple two-layer neural network, previously enjoyed properties such as smoothness and convexity do not hold. Studying the behavior of optimization algorithms in this context is incredibly difficult, and so trial and error and vague intuition often guide the development of new algorithms. Further, simple neural networks are only where the complexity begins; far more complex architectures and data distributions are common in practice.

1.2 Building towards Adam

While the contributions of this thesis are primarily interested in the Adam algorithm, it is a relatively recent development. Optimization problems like Example 1 have been of interest for far longer, and several methods have been developed to solve them. Nearly all modern methods are based on gradient descent (Algorithm 1), a classic, well understood, first-order optimization algorithm. This algorithm iteratively adjusts $\mathbf{w}$ by taking steps in the direction of the negative gradient, which is the direction of steepest descent at any given point. In the case of a smooth, convex problem like Example 1, this algorithm will provably minimize the gap in loss between the initial and optimal loss at a rate of at least $O(t^{-1})$ where t is the number of steps taken.

While gradient descent (GD) is less computationally complex than second order optimization methods, it can still be prohibitively expensive with a large dataset. If we have n training examples each with d features, one gradient computation will cost at least $O(nd)$ operations. Modern datasets often have millions or even billions of examples, meaning any dependence on n will be prohibitively expensive. To mitigate this, an approximation known as stochastic gradient descent (SGD) is used. Rather than computing the full gradient $\mathbf{g}_t$, a subset of the dataset will be randomly selected at each step (referred to as a batch or mini-batch) and used to approximate the full gradient. The batch gradient will be equal to the true gradient in expectation, but is a noisy approximation in practice. This approximate gradient using

Algorithm 1 Gradient Descent

Require: α: stepsize
Require: $\mathcal{L}(\mathbf{w})$: objective function with parameters $\mathbf{w}$
Initialize $\mathbf{w}_0$, $t \leftarrow 0$
while $\mathbf{w}_t$ not converged **do**
    $t \leftarrow t + 1$
    $\mathbf{g}_t \leftarrow \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}_{t-1})$
    $\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} - \alpha \cdot \mathbf{g}_t$
end while
return $\mathbf{w}_t$

batch $\mathcal{B}_t$ is written as

$$\tilde{\mathbf{g}}_t = \frac{1}{|\mathcal{B}_t|} \sum_{i \in \mathcal{B}_t} \nabla_{\mathbf{w}} \ell(m(\mathbf{x}_i, \mathbf{w}_t), y_i).$$

Another common addition to SGD is momentum. While multiple momentum like methods exist (Nesterov acceleration for example (Nesterov, 1983)), the most common is the so called "heavy-ball momentum" due to Polyak (Polyak, 1964). In addition to the standard gradient update, a momentum term that tracks how the updates have been changing is added. This update can be written as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}_t) + \beta(\mathbf{w}_t - \mathbf{w}_{t-1}),$$

however a different parameterization is typically used in practice. Rather than saving the previous iterate $\mathbf{w}_{t-1}$, a "momentum buffer" $\mathbf{m}_t$ of the gradients is kept and used for the update. This is mathematically equivalent and leads to an update of the form

$$\mathbf{m}_{t+1} = \beta \mathbf{m}_t + \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}_t)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \mathbf{m}_t.$$

Momentum only provably accelerates gradient descent on quadratic functions like Example 1, but in practice is crucial for optimization of deep networks. This addition also introduces the idea of state in optimization algorithms, where the update is dependent on the entire sequence of iterates. Stateless optimizers have the advantage of requiring less memory and computational complexity, however their effectiveness is often limited.

In all optimization algorithms, it is typically necessary to tune the learning rate α. While other hyperparameters exist (such as the momentum hyperparameter β), the learning rate often has the largest impact and can drastically change the speed at which the loss is minimized. Given this, a common question is can the learning rate be tuned or chosen automatically? This question gives rise to a class of optimizers refereed to as "adaptive." The common theme of these algorithms is automatically choosing a step size, often for each coordinate, based on the properties of already seen gradients and iterates. However, it is an

open question as to what the algorithms are actually "adapting" to. Nevertheless, adaptive optimizers have dominated in recent years, and are the default choice for most problems.

An early example of an adaptive method is Rprop (for "resilient backpropagation") (Riedmiller and Braun, 1993), which disregarded the magnitude of entries of the gradient and only looked at their sign. Roughly, if the sign of a given gradient entry agreed between steps, Rprop increases the learning rate for that coordinate, and decreased it if the sign changed. The Rprop update can be written as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \boldsymbol{\alpha}_t \circ \operatorname{sign}(\mathbf{g}_t)$$

where $\boldsymbol{\alpha}_t$ is a vector of each entry's adjusted learning rate. While Rprop saw some early success when using the deterministic full batch gradient, stochasticity caused Rprop to become unstable and lead to very large weight updates. This issue lead to the development of RMSProp (Tieleman and Hinton, 2012). Instead of using the exact sign of the gradient, RMSProp element-wise divides the gradient with the magnitude of a moving average of the mini-batch gradients. If this operation were done only with the current gradient rather than a moving average, it would equivalent to taking the sign as in Rprop. The moving average forces gradients of "adjacent" batches to be divided by a similar value, alleviating stability issues in Rprop. To accomplish this, RMSProp stores a moving average of the entry-wise squared gradient and divides the gradient entries by the entry-wise square root (plus a small constant ϵ for numerical stability). Mathematically, the RMSProp update is

$$\begin{aligned}\mathbf{v}_{t+1} &= \beta \mathbf{v}_t + (\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}_t))^2 \\ \mathbf{w}_{t+1} &= \mathbf{w}_t - \alpha \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}_t)) / \sqrt{\mathbf{v}_{t+1} + \epsilon}.\end{aligned}$$

While RMSProp was developed as an extension of Rprop, the idea of dividing by the square root of the gradient squared predates it. The AdaGrad (Duchi et al., 2011) algorithm was proposed slightly earlier in the context of online learning problems, a related subfield of optimization. In that context, AdaGrad provably minimizes regret, but the proof techniques and problems in online learning do not perfectly carry over to general machine learning problems. RMSProp additionally has a few distinctions from AdaGrad, notably that AdaGrad keeps a sum of squared gradients rather than a moving average and relies on a specific decreasing learning rate scheme to ensure the theory holds.

The final piece leading up to Adam is the combination of RMSProp and the previously discussed heavy-ball momentum. The developers of RMSProp had attempted to add momentum, however were unable to find a momentum scheme that improved performance. Most attempts involved keeping a momentum buffer of the updates, after they had been computed based on the gradient and squared gradient buffer. Adam retains the moving average of squared gradient entries (as in RMSProp) and adds a second moving average of just the gradient (as with heavy-ball momentum). These two moving averages are then used

to compute the update. A slightly simplified form of the Adam update is

$$\begin{aligned}\mathbf{m}_{t+1} &= \beta_1 \mathbf{m}_t + (1-\beta_1)\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}_t)\\ \mathbf{v}_{t+1} &= \beta_2 \mathbf{v}_t + (1-\beta_2)(\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}_t))^2\\ \mathbf{w}_{t+1} &= \mathbf{w}_t - \alpha \mathbf{m}_{t+1}/\sqrt{\mathbf{v}_{t+1}+\epsilon}.\end{aligned}$$

Despite the increased memory cost of having two momentum buffers, Adam has shown unprecedented empirical success. Extensive research has been conducted to develop new algorithms with better performance than Adam, but very few examples exist. Only a single followup algorithm has gained significant adoption, the AdamW variant. It is often helpful to add a penalty to a model's complexity to discourage overfitting, which in practice typically involves encouraging the model's weights to have a small norm. This is done by adding the norm of the parameters to the loss during optimization, leading to an optimization problem of the form

$$\mathbf{w}^* \in \arg\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

where λ is a "strength" hyperparameter. In deep learning this is often referred to as weight decay, or more traditionally as (ℓ_2-) regularization. Loshchilov and Hutter (2019) found that using only the Adam update on the loss function $\mathcal{L}(\mathbf{w})$ while using the standard gradient update on $\frac{\lambda}{2}\|\mathbf{w}\|^2$ was beneficial, and labeled this as "weight decay decoupling." The full AdamW algorithm is shown in Algorithm 5, where $\alpha \cdot \lambda \cdot \mathbf{w}_{t-1}$ is the update from the weight decay decoupling. Additionally, lines 7 and 8 are referred to as the "bias correction" step in Adam. Given that $\mathbf{m}$ and $\mathbf{v}$ are initialized to zero, this "biases" the moving average towards zero, and so this additional step will remove that "bias."

Algorithm 2 AdamW Optimization Algorithm

Require: α: stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: exponential decay rates for moment estimates
Require: λ: weight decay coefficient
Require: ϵ: small constant for numerical stability
Require: $\mathcal{L}(\mathbf{w})$: objective function with parameters $\mathbf{w}$
1: Initialize $\mathbf{w}_0$, $\mathbf{m}_0 \leftarrow \mathbf{0}$, $\mathbf{v}_0 \leftarrow \mathbf{0}$, $t \leftarrow 0$
2: **while** $\mathbf{w}_t$ not converged **do**
3: $t \leftarrow t + 1$
4: $\mathbf{g}_t \leftarrow \nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}_{t-1})$
5: $\mathbf{m}_t \leftarrow \beta_1 \cdot \mathbf{m}_{t-1} + (1-\beta_1) \cdot \mathbf{g}_t$
6: $\mathbf{v}_t \leftarrow \beta_2 \cdot \mathbf{v}_{t-1} + (1-\beta_2) \cdot \mathbf{g}_t^2$
7: $\hat{\mathbf{m}}_t \leftarrow \mathbf{m}_t/(1-\beta_1^t)$
8: $\hat{\mathbf{v}}_t \leftarrow \mathbf{v}_t/(1-\beta_2^t)$
9: $\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} - \alpha \cdot \hat{\mathbf{m}}_t/(\sqrt{\hat{\mathbf{v}}_t}+\epsilon) - \alpha \cdot \lambda \cdot \mathbf{w}_{t-1}$
10: **end while**
11: **return** $\mathbf{w}_t$

Choices of hyperparameters β_1 and β_2 can recover other algorithms, for example setting β_1 to zero will recover RMSProp. Many attempts to analyze Adam also do so through a simpler algorithm known as Sign Descent, where only the sign of the gradient is considered. This special case corresponds to setting β_1, β_2, and ϵ to zero. Comparing the properties of algorithms such as Sign Descent and gradient descent has been a fruitful method for attempting to understand the properties of Adam.

1.3 Contributions

This thesis contains a collection of works attempting to understand why the Adam optimization algorithm works so well in practice. These works aim to identify a particular property of an optimization problem that is causing a performance difference between the well understood gradient descent and Adam. Once a property is identified, we study what is making gradient descent worse, Adam better, or both. This is primarily done through experimentation on simpler more understandable problems, and then scaled up to show the identified phenomenon is likely also the cause in practice. While none of these works claim to identify the single reason Adam is effective, extensive experimental results do show a strong correlation with performance.

Chapter 2: Heavy-tailed Class Imbalance

In Chapter 2, we identify a key characteristic that we argue significantly contributes to Adam's dominance over SGD when training language models. In language modelling tasks, Adam often shows far superior performance to SGD. In contrast, in other domains such as training vision models Adam and SGD have a less significant performance gap. Language modelling is typically done through "next word prediction," where a model is given a prefix of words and returns a distribution of what it thinks the next word is. We identify a characteristic of language data that we argue causes this gap. In many tasks, the distribution of labels $\mathbf{y}_i$ is balanced. For example, in the ImageNet-1K dataset (Deng et al., 2009), most of the 1000 classes have around 1300 samples. However, language data has a naturally imbalanced structure. Some words are very frequent, but there are many words that are infrequent. This gives rise to the distribution of the classes being predicted to be proportional to $1/k$ where k is the index of the classes as ranked by frequency. This can be called a power law or a Zipfian distribution, and it is known the distribution of words in natural language follows this law (Piantadosi, 2014). We show through extensive experimentation and theory on simpler models that this structure produces a gap between Adam and SGD. SGD struggles to make progress on lower frequency classes, where as Adam is able to minimize the loss for all classes at the same rate. We additionally find that this imbalance leads to a correlation between gradients and Hessians in the last layer of language models, which Adam exploits with its update scheme. The eigenvalues of the Hessian additionally span several orders of magnitude, creating an ill conditioning problem SGD will struggle with. To further support these claims, we create vision and synthetic datasets where typically

Adam and SGD would typically have similar performance, and create a performance gap by adding a heavy tail to the class distribution. We find that this phenomenon is not unique to deep learning as well, and create linear models where Adam outperforms SGD despite little theory existing to predict this gap.

Chapter 3: Feature Based Ill Conditioning

In Chapter 3, we build on the results of Chapter 2 by finding another setting where Adam outperforms GD. Graph neural networks (GNNs) are a class of models used with graph based data including applications in network analysis, product recommender systems, and drug design. It is well known amongst practitioners that GD is often very ineffective for training, and Adam has become the standard choice. In contrast to work presented in Chapter 2, the common baseline datasets used in GNN research have (often perfectly) balanced classes. We find that it is systematically the case that Adam well outperforms GD on common datasets and GNN architectures, including a linear model where theory would not predict a gap. In this instance, it is properties of the features in these datasets that is creating a performance gap between Adam and GD. The features in graph datasets are often based on bag-of-words representations of text. Examples include nodes being bag-of-words encoded product reviews and edges products commonly bought together or nodes based on academic publication abstracts and edges representing publications that cite each other. Given these features are based on natural language, we find that a similar imbalance occurs in feature frequency. As in Chapter 2, this imbalance in feature frequency leads to a poorly conditioned optimization problem, and GD struggles. Furthermore, we find that a common preprocessing step used to "normalize" features in these datasets is actually widening this performance gap between Adam and GD, and in fact making both worse. We display that under a more typical normalization method, both algorithms minimize the loss dramatically faster.

Chapter 4: Basis Sensitivity

In Chapter 4, we examine Adam's sensitivity to the basis being used in the optimization problem. More classical algorithms like SGD are "rotation equivariant," meaning that if the function is rotated (and not stretched or squished), the trajectory of the iterates will be rotated as well, preserving the behavior of the algorithm. Many assumptions commonly used in optimization theory are rotation invariant. For example, a common assumption placed on functions is $L-$smoothness with respect to the ℓ_2 norm. The constant L describes how "smooth" the function is, and does not change if the function is rotated. Adam on the other hand is not a rotation equivariant algorithm. Its trajectory will change under rotation, meaning that common assumptions used to study Adam theoretically cannot fully describe its behaviour. We show that Adam's performance does in fact depend on the basis being used. In most cases, random rotations of the basis degrades performance. We argue that this basis dependence requires that theoreticians studying Adam use assumptions that are not rotationally invariant. Further, we identify some non-rotationally invariant assumptions that exist in the literature, and find that they are ineffective at predicting Adam's performance.

That is, improving constants in their definition does not in fact improve Adam's eventual performance. However, we find that there do exist bases where Adam's performance is improved. Inspired by recent advances in the literature, we rotate gradient matrices by their left and right singular vectors, leading to a faster decrease in the loss. Finally, we draw connections between this variant of Adam and the recently popularized Muon algorithm (Jordan et al., 2024), and show that update "orthogonality" is a quantity that does in fact correlate with Adam's performance.

Chapter 2

Heavy-tailed Class Imbalance

Adam has been shown to outperform gradient descent on large language models by a larger margin than on other tasks, but it is unclear why. We show that a key factor in this performance gap is the heavy-tailed class imbalance found in language tasks. When trained with gradient descent, the loss of infrequent words decreases more slowly than the loss of frequent ones. This leads to a slow decrease on the average loss as most samples come from infrequent words. On the other hand, Adam and sign-based methods are less sensitive to this problem. To establish that this behavior is caused by class imbalance, we show empirically that it can be reproduced across architectures and data types, on language transformers, vision CNNs, and linear models. On a linear model with cross-entropy loss, we show that class imbalance leads to imbalanced, correlated gradients and Hessians that have been hypothesized to benefit Adam. We also prove that, in continuous time, gradient descent converges slowly on low-frequency classes while sign descent does not.

2.1 Introduction

The recent success of large language models such as GPT-3 (Brown et al., 2020) and its successors has relied on costly training procedures at unprecedented scale. A key ingredient in their training is the Adam optimizer (Kingma and Ba, 2015), which outperforms stochastic gradient descent (SGD) on language problems by a large margin. Despite this large performance gap, we have a poor understanding of why Adam works better and it has been difficult to find new optimizers that consistently improve over Adam (Schmidt et al., 2021). Not only is it computationally difficult to validate new optimizers on large models, but we also lack theoretical guidance; we do not know what "problem" Adam solves to outperform SGD.

The success of Adam on language transformers has been well documented. Multiple works have found metrics or statistics that correlate with the improved performance of Adam, showing that it yields uniform updates across parameters despite imbalanced gradients (Liu et al., 2020b), gives a better descent direction than the gradient (Pan and Li, 2023), and takes a path over which a robust variant of the condition number is smaller (Jiang et al., 2023). But these observations do not provide a mechanism explaining what property of the problem leads to the improved performance of Adam.

Plausible mechanisms have been put forward, but they do not provide a complete explanation. Zhang et al. (2020b) show that Adam-like methods are more resilient to heavy-tailed noise, which seems more prominent in language than in vision tasks. But noise is not the primary cause of the gap, as it already appears in deterministic training (Kunstner et al., 2023). An alternative hypothesis is that the magnitude of the gradient and Hessian are correlated, which justifies clipping (Zhang et al., 2020a). But to justify methods that normalize element-wise, like Adam and sign-like methods, we additionally need the gradient and Hessian to be correlated across parameters (Crawshaw et al., 2022). While there is empirical evidence for this behavior in neural networks, we do not have a good understanding of why this occurs, nor why this would be more pronounced on language rather than vision tasks.

2.1.1 Contributions

Our goal is to answer the following question: *what is the "problem" that makes SGD slow on language tasks, that Adam "fixes" to perform better?*

We argue the problem is what we call heavy-tailed class imbalance, where rare classes account for a large fraction of the data. Language data is imbalanced as some words are much more frequent than others, typically following a power-law. A common modeling assumption is Zipf's law, where the kth most frequent word has frequency $\propto \frac{1}{k}$ (Piantadosi, 2014). For language tasks framed as next-token prediction, this property is reflected in the tokens and leads to heavy-tailed class imbalance. This contrasts with typical vision datasets such as MNIST, CIFAR, and ImageNet, which are curated to have uniform classes, but also with imbalanced problems with a small number of classes. For example, in binary

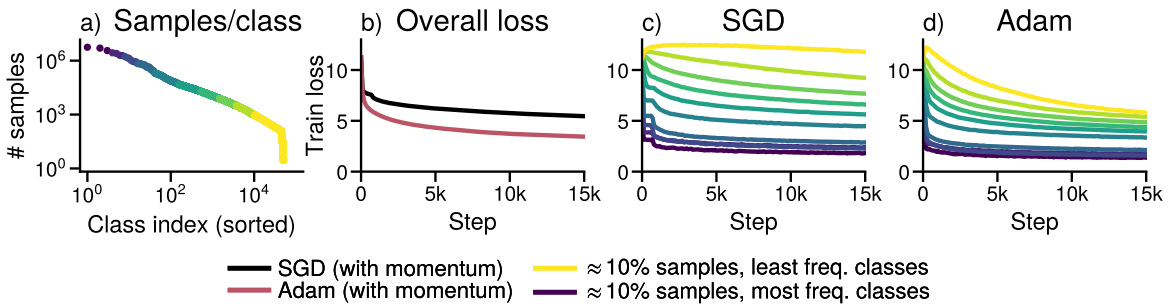


Figure 2.1: **Gradient descent does not make progress on low-frequency classes, while Adam does.**
Training GPT2-Small on WikiText-103. **(a)** Distribution of the classes sorted by class frequency, split into groups corresponding to ≈10% of the data. **(b)** Overall training loss. **(c, d)** Training loss for each group using SGD and Adam. SGD makes little to no progress on low-frequency classes while Adam makes progress on all groups. **(b)** is the average of **(c, d)** for the respective optimizer.

classification, extreme imbalance implies the minority class has a limited impact on the loss; with an imbalance of 99:1, only 1% of the data comes from the minority class.

The performance gap arises because SGD makes slow progress on rare classes, see Figure 2.1. On a binary problem, slow performance on 1% of the data need not have a large impact on the average loss if we make fast progress on the remaining 99% of the samples. In contrast, the heavy-tailed class imbalance found in language tasks makes it possible for low-frequency classes to account for most of the data and significantly contribute to the loss, leading to slow performance overall.

We show that heavy-tailed class imbalance makes SGD slow across tasks in Section 2.2. We show that modifying vision datasets to exhibit heavy-tailed imbalance leads to slow progress with SGD on architectures where the performance gap with Adam is typically smaller. The impact of heavy-tailed imbalance can even be seen on linear models. Additionally, the performance of SGD improves with techniques that address imbalance such as upweighting rare classes.

Our findings provide a simple model where Adam outperforms SGD, a softmax linear model under heavy-tailed class imbalance, which we analyze in Section 2.3. We show empirically that a correlation between the magnitude of the gradient and Hessian across coordinates, used to justify the benefits of Adam, appears naturally even on a linear model with class imbalance. We provide intuition as to how this pattern emerges through an assignment mechanism that leads to a correlation between class frequencies and the magnitude of the gradient and Hessian across parameters. We additionally prove that, on a simple dataset and in continuous time, GD is slow on low-frequency classes while sign descent is insensitive to the class frequencies.

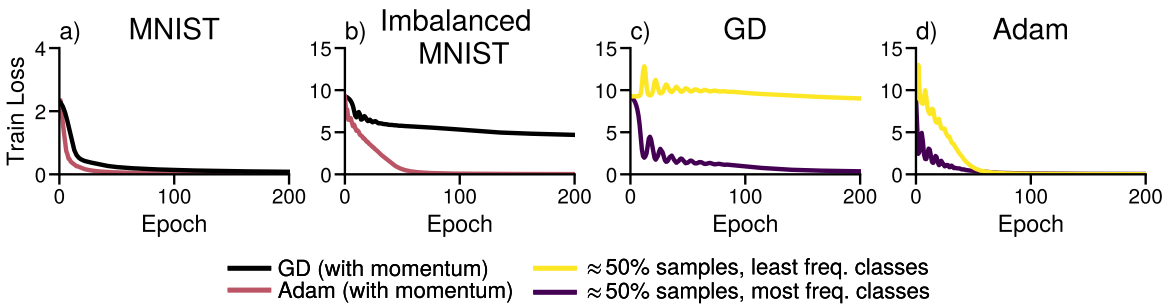


Figure 2.2: **Adam outperforms GD for training a CNN under heavy-tailed class labels. (a)** Performance on the MNIST dataset. **(b)** Performance on a modified MNIST with two groups of classes. The first group consists of the 10 original classes with ≈ 5k samples each, while the second consists of $\approx$10k added classes with 5 examples each. **(c, d)** Performance of GD and Adam on the two groups. The initial loss is higher for imbalanced MNIST as there are $\approx 10^4$ classes instead of 10, leading to a loss of $-\log(1/10^4) \approx 9.2$ for a uniform prediction instead of $-\log(1/10) \approx 2.3$.

We do not claim that class imbalance is the only reason Adam outperforms SGD, as other properties of the data or architectures likely also contribute to this gap. Instead, we show that Adam consistently outperforms SGD under heavy-tailed class imbalance. The difficulty of minimizing the loss of minority classes has been explored for binary problems or problems few classes (Anand et al., 1993; Francazi et al., 2023), but the recent scaling of large language models to predictions over more than 100000 classes puts the problem on a new scale. Our findings indicate that heavy-tailed class imbalance has a significant impact on training performance and should be a consideration for future optimizers to perform well on language and other tasks exhibiting heavy-tailed class imbalance.

2.2 Experimental results and ablation studies

Figure 2.1 suggests a correlation between class frequencies and optimization performance that impacts SGD more than Adam. The goal of this section is to verify that (i) class imbalance is a root cause for the performance gap between SGD and Adam, and (ii) whether this gap can be reproduced with simpler algorithms, such as deterministic optimizers, or using sign descent as a proxy for Adam.

To test these hypotheses, we perform experiments focusing on the training loss as our objective is to understand what makes optimization difficult. We use a simple training procedure, with a constant step-size tuned by grid search. For visualization, we split the data into groups of classes with similar frequencies, as in Figure 2.1. For instance, for 10 groups, the first group corresponds to $\approx$10% of the samples from the most frequent classes. This grouping is only used for visualization and does not affect training. The models, datasets and training procedures are described in Appendix A.1.

In Appendix A.2, we give additional information and additional ablation experiments on language models. We show that the heavy-tailed class distribution appears across datasets and tokenizers, and that the separation across class frequencies observed on the training loss in Figure 2.1 also affects the validation loss. We show that similar dynamics appear on smaller language models, including when training only the last layer while keeping the embedding and attention modules frozen at initialization. Finally, we show that stochasticity is not necessary to reproduce the impact of heavy-tailed class imbalance, and that it also appears when using deterministic updates (i.e., GD instead of SGD). As a result, we use deterministic updates whenever possible, denoted by GD in the figures.

2.2.1 Reproducing the frequency gap with vision models

Language transformers are often contrasted with vision CNNs, where we do not see a large performance gap between SGD and Adam. Our hypothesis is that a key differentiation between the two settings is the heavy-tailed class imbalance present in language data. In this section, we show that making heavy-tailed vision datasets leads to slower performance with SGD and a larger performance gap with Adam. These experiments show that heavy-tailed imbalance has a significant impact on performance and can make an otherwise "easy" problem into a "hard" one for SGD.

CNN. We first use a CNN on a variant of MNIST with heavy-tailed class imbalance. We augment the dataset to have two equally-sized groups of classes with a relative frequency difference of 1000. The first group consists of the original 10 classes with ≈5k samples/class. For the second, we create ≈10k new classes with 5 samples/class. We create new classes by copying existing images and adding a "barcode" in a corner of the image, see Appendix A.1. The performance of GD and Adam is shown in Figure 2.2. On the original MNIST dataset, both optimizers drive the loss to 0, and Adam still makes progress on both groups in the imbalanced case. But on the imbalanced variant, GD makes almost no progress on half of the data corresponding to the low-frequency classes and progress stalls. However, it eventually converge if run for much longer (see Appendix A.4.2), indicating that the problem is one of slow optimization rather than getting stuck in a local minima.

ResNet. We replicate this effect with a ResNet18 on an imbalanced variant of ImageNet. We subsample classes with frequencies $\pi_k \propto 1/k$ and compare against a uniform subset with a similar number of samples. In Figure 2.3, we see that SGD and Adam perform similarly on uniform data but a performance gap appears across class frequencies on the heavy-tailed imbalanced dataset. As in Figures 2.1 and 2.2, SGD is slower on imbalanced data, especially on low-frequency classes.

Vision Transformers. This performance gap also appears with vision transformers (ViTs). In Appendix A.3, we see that SGD and Adam both perform well on ImageNet, but exhibit a similar performance gap as in Figure 2.1 on the imbalanced variant. While ViTs may require more raw data, data augmentations, or regularization to generalize as well as ResNets (Steiner et al., 2022), there does not seem to be a large gap between SGD and Adam without

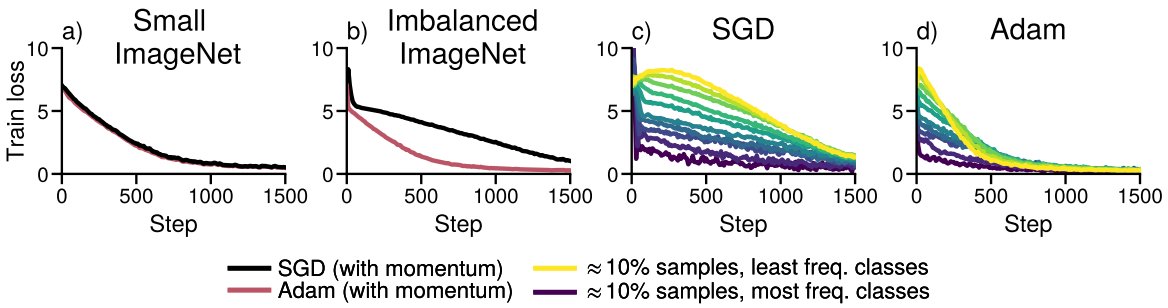


Figure 2.3: **Adam outperforms SGD for training a ResNet under heavy-tailed class labels. (a)** Performance on a subset of ImageNet and **(b)** an imbalanced subset of ImageNet with class frequencies $\pi_k \propto 1/k$. **(c, d)** Performance of GD and Adam on groups corresponding to ≈10% of the data.

class imbalance.

2.2.2 Reproducing the frequency gap with a linear model on uniform data

To highlight that heavy-tailed imbalance alone can lead to the observed difficulties, we reproduce this behavior in a simple setting: a softmax linear model with cross-entropy loss. We create a dataset where the class frequencies approximate $\pi_k \propto 1/k$ and draw n samples uniformly from $[0, 1]$ in d dimensions, independently of the label. While there is no relationship to learn, the optimization problem is still well posed and a linear model can separate the data if $n \ll d$. As on the transformer of Figure 2.1, GD makes less progress on low-frequency classes than Adam, as shown in Figure 2.4.

This example illustrates that a problem that might look innocuous at first is hard to optimize with GD due to heavy-tailed imbalance, while the performance of Adam is less negatively impacted. Nonetheless, imbalance alone is not sufficient to make GD slow. It is possible to generate pathological datasets with heavy-tailed imbalance where GD fits all classes fast, by making all the samples (close to) orthogonal. In this case, each sample is learned independently of the others, and there is no difference across classes. However, perfectly orthogonal data is unlikely, especially as we expect samples from similar classes to be assigned a similar (correlated) representation. We discuss this issue and give additional examples on the linear model in Appendix A.4.

2.2.3 Interactions between optimizer and imbalance

We have shown that heavy-tailed class imbalance can lead to different performance across class frequencies, but it is not clear which component of the training process has the highest impact on this behavior. We next experiment with simple algorithms to answer the following questions. (i) Is the impact of class imbalance due to stochasticity, or does it happen with

deterministic training? (ii) Which component of Adam leads to an improved performance? and (iii) If imbalance is the problem, can we improve the performance of SGD by reweighting the losses?

Class imbalance already impacts deterministic optimization. A natural hypothesis to explain the impact of class imbalance is that it may be due to small batch sizes in SGD; rare classes could be sampled less often, and thus learned more slowly. On the other hand, stochasticity has been found to have little impact on the gap between SGD and Adam (Kunstner et al., 2023). Our experiments in Figures 2.2, 2.4 and 2.5 and further examples in Appendix A.2.3 reproduce the dynamics of Figure 2.1 with full batch GD and Adam, indicating the problem already arises in the deterministic setting.

Adam and sign descent both perform well under imbalance. Following Kunstner et al. (2023), we check whether the benefit of Adam is due to a change in the magnitude of the update or its direction. Changing the magnitude as in normalized GD is known to perform better on separable problems (Nacson et al., 2019), while the benefits of Adam have been attributed to the change of direction close to sign descent (Tieleman and Hinton, 2012; Balles and Hennig, 2018). We compare the performance of GD, Adam, normalized GD and sign descent, with and without momentum, for training the last layer of a small transformer in Figure 2.5 and on additional problems in Appendix A.5. Normalization and momentum helps across problems, but they have less impact on the performance gap across class frequencies than changing the update direction. Sign descent and Adam have a similar performance.

Upweighting low-frequency classes can help. Given our hypothesis that the performance gap between (S)GD and Adam is due to class imbalance, we expect interventions directly targeting imbalance to improve performance. In Appendix A.5.1, we show that upweighting the loss of low-frequency classes can improve the performance of SGD. While reweighting is not complete solution as it changes the objective function, this experiment supports the hypothesis that the optimization problem is due to heavy-tailed class imbalance.

2.3 An investigation on linear models

Heavy-tailed imbalance already leads to slow performance on the linear softmax model of Figure 2.4, but we do not have a good understanding of why GD becomes slow while Adam is less affected. In this section, we explore the effect of heavy-tailed class imbalance on the special case of softmax linear models, showing that it leads to correlated, imbalanced gradients and Hessians. In Section 2.3.1, we give an example on a quadratic where imbalanced Hessians lead to a performance gap between GD and Adam. In Section 2.3.2, we show that class imbalance leads to imbalanced gradients and Hessians that are correlated with class frequencies through an *assignment mechanism*, showing that this pattern emerges naturally. Finally, we prove that on a simple imbalanced problem and in continuous time, GD is slow on low-frequency classes while sign descent is fast on all classes in Section 2.3.3.

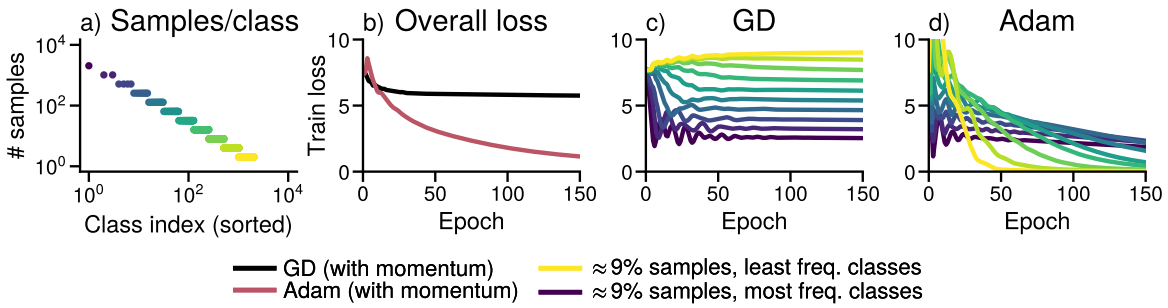


Figure 2.4: **The impact of heavy-tailed class imbalance is reproducible with linear models.** Softmax regression on synthetic data. The inputs are drawn from a uniform distribution on $[0, 1]^d$. The target classes are heavy-tailed **(a)** and independent of the inputs, but the model can still fit the data as it is overparameterized. **(b, c, d)** Overall training loss and performance of GD and Adam on each subset.

2.3.1 Intuition on a weighted quadratic problem

Consider the following toy problem which is purposefully oversimplified to provide a high-level intuition about the optimization dynamics. Suppose we have c functions $f_1, ..., f_c$, corresponding to the losses for each class, that are on the same scale in the sense that gradient descent with step-size α makes fast progress on any f_i. For concreteness, take $f_i(w) = \frac{1}{2}\|w\|^2$, where GD with a step-size of 1 converges in one step. Instead of running GD on each function independently, suppose we run GD on the weighted average $f(w_1, ..., w_c) = \sum_{i=1}^{c} \pi_i f_i(w_i)$ with positive weights $\pi_1 \geq ... \geq \pi_c$, $\sum_i \pi_i = 1$, corresponding to the class frequencies. If these weights span multiple orders of magnitude, we expect a similar behavior as in Figures 2.1 to 2.5, as illustrated in Figure 2.6. GD makes slow progress on functions with low weights as the gradient w.r.t. w_k is scaled by π_k,

$$w_k^{(t)} = w_k^{(t-1)} - \alpha \pi_k f_k'(w_k^{(t-1)}) = (1 - \alpha\pi_k)^t w_k^{(0)}.$$

This slow convergence on functions with low weights cannot be fixed by increasing the step-size, as increasing it beyond $1/\pi_1$ would cause instabilities on the highest-frequency "class" f_1. The problem is that we use the same step size for all functions, which have different scales. Adam and sign descent are less sensitive to this problem as their updates are independent of π_k,

$$w_k^{(t)} = w_k^{(t-1)} - \alpha \frac{\pi_k f_k'(w_k^{(t-1)})}{\left|\pi_k f_k'(w_k^{(t-1)})\right|} = w_k^{(t-1)} - \alpha \operatorname{sign}(f_k'(w_k^{(t-1)})).$$

While sign descent or Adam with a fixed step-size need not converge and can oscillate around the minimum, they perform much better in early iterations, independently of π_k.

Another perspective is that the imbalance in the weights $\pi_1, ..., \pi_c$ makes the problem ill-conditioned. The weights not only affect the gradient of f but also its Hessian, which

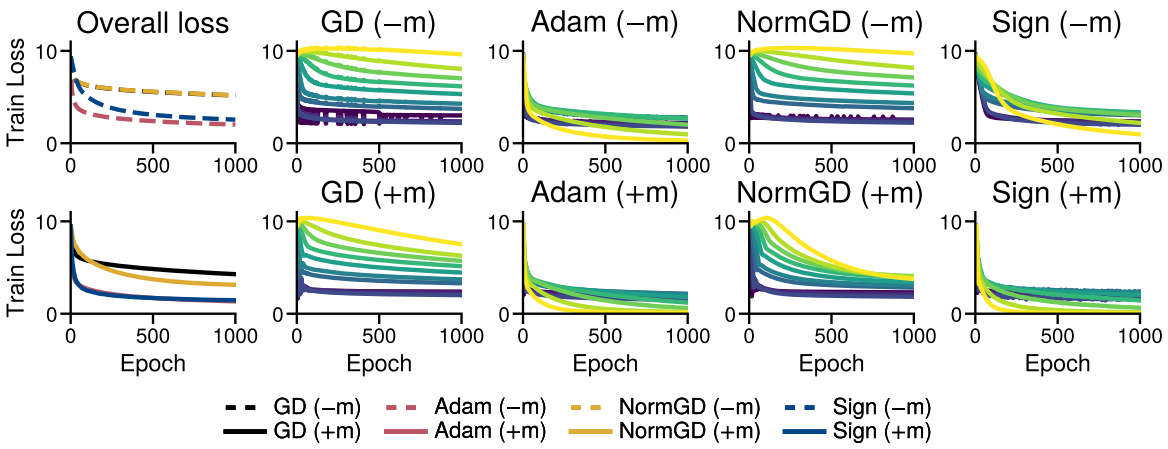


Figure 2.5: **Sign descent, as a simplified form of Adam, performs well on low-frequency classes.** Training the last layer of a simplified one-layer transformer with GD, Adam, normalized GD, and sign descent, with and without momentum (±m). Momentum and normalizing the magnitude help but have smaller effects than using sign descent, which recovers similar dynamics to Adam.

is $\text{Diag}([\pi_1, ..., \pi_c])$. A common intuition for Adam is that using the magnitude of the coordinates of the gradient as a preconditioner is a good proxy for the Hessian diagonal (Duchi et al., 2011; Kingma and Ba, 2015), which would also lead to larger step-sizes for coordinates with small π_k. While this does not hold in general (Kunstner et al., 2019), the gradient can be a reasonable approximation to the Hessian on this problem. The gradient is $[\pi_1 w_1, ..., \pi_c w_c]$. If the weights $\pi_1, ..., \pi_c$ vary by orders of magnitude more than the parameters $|w_1|, ..., |w_c|$, the gradient and Hessian will be correlated, and preconditioning by the gradient magnitude or Hessian diagonal will yield similar directions.

2.3.2 Correlations between the magnitude of the gradient and Hessian across coordinates

What is lacking to explain Adam's improved performance is an understanding of how a correlation between the gradient and Hessian arises in realistic problems. This feature has been observed on neural networks, but we do not yet know why it appears, even on the softmax linear problem. The caricature of the diagonal quadratic problem of the previous section provides some intuition, but does not directly apply to the softmax linear model of Figure 2.4 as that problem is neither quadratic nor separable. Nonetheless, a similar pattern emerges in the rows $\mathbf{w}_1, ..., \mathbf{w}_c$ of its parameter matrix $\mathbf{W} \in \mathbb{R}^{c \times d}$; the magnitude of the gradient and Hessian across rows and the class frequencies can become correlated during training due to class imbalance. In this section, we establish this observation empirically and provide a mechanism for how it emerges.

In Figure 2.7, we show the gradient norm against the Hessian trace with respect to each

row $\mathbf{w}_k$ throughout the trajectory of Adam on the softmax linear model of Figure 2.4. While there is no correlation at initialization, the gradient and Hessian blocks become correlated with class frequencies during training and become imbalanced. This imbalance in the diagonal blocks is the main feature of the Hessian as the than off-diagonal blocks are orders of magnitude smaller, as shown in Figure 2.9. Similar dynamics occur with GD, although only on high-frequency classes as GD makes little progress on low-frequency classes, see Appendix A.6. This correlation also appears in the last layer of large models such as GPT2-Small used in Figure 2.1, as shown in Figure 2.8.

To explain this behavior, we show that the impact of samples on the Hessian follows an *assignment mechanism*: if the model assigns samples to their correct class, the Hessian with respect to $\mathbf{w}_k$ is primarily influenced by samples from class k, leading to a correlation between the magnitude of the gradient, Hessian, and class frequencies. To capture this effect, we introduce some notation and a simplifying assumption. Suppose we have n samples with inputs $\mathbf{x}_i \in \mathbb{R}^d$ and labels $y_i \in [c]$, where class k has frequency $\pi_k = {}^{n_k}/_{n}$. The parameters of the linear model are $\mathbf{W} \in \mathbb{R}^{c\times d}$. We write $\mathbf{p}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x})$ for the predicted probabilities where σ is the softmax, and summarize the data as

$$\bar{\mathbf{x}} = \tfrac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i, \quad \bar{\mathbf{x}}^k = \tfrac{1}{n_k}\sum_{i:y_i=k} \mathbf{x}_i, \quad \bar{\mathbf{H}} = \tfrac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i\mathbf{x}_i^\top, \quad \bar{\mathbf{H}}^k = \tfrac{1}{n_k}\sum_{i:y_i=k} \mathbf{x}_i\mathbf{x}_i^\top.$$

Assumption 1 (correct assignment)**.** The model correctly assigns samples to class k if it predicts k with non-negligible probability p on samples from that class ($\mathbf{p}(\mathbf{x}_i)_k = p = \omega({}^1/_c)$ for $\mathbf{x}_i$ from class $y_i = k$), and predicts k with near-random chance otherwise ($\mathbf{p}(\mathbf{x}_i)_k = O({}^1/_c)$ for $\mathbf{x}_i$ where $y_i \neq k$).

Proposition 2. *If initialized at $\mathbf{W}_0 = 0$, the gradient and Hessian of the loss $\mathcal{L}$ w.r.t. $\mathbf{w}_k$ are*

$$\nabla_{\mathbf{w}_k}\mathcal{L}(\mathbf{W}_0) = \pi_k\bar{\mathbf{x}}^k - \tfrac{1}{c}\bar{\mathbf{x}}, \qquad \nabla^2_{\mathbf{w}_k}\mathcal{L}(\mathbf{W}_0) = \tfrac{1}{c}\big(1-\tfrac{1}{c}\big)\bar{\mathbf{H}}, \tag{2.1}$$

During training, if the model correctly assigns samples to class k with probability p (Assumption 1),

$$\begin{aligned}\nabla_{\mathbf{w}_k}\mathcal{L} &= \ (1-p)\pi_k\,\bar{\mathbf{x}}^k \ + O\big(\tfrac{1}{c}\big),\\ \nabla^2_{\mathbf{w}_k}\mathcal{L} &= p(1-p)\pi_k\,\bar{\mathbf{H}}^k + O\big(\tfrac{1}{c}\big),\end{aligned} \quad \textit{and} \quad \|\nabla_{\mathbf{w}_k}\mathcal{L}\| \sim \left(\frac{1}{p}\frac{\|\bar{\mathbf{x}}^k\|}{\operatorname{Tr}(\bar{\mathbf{H}}^k)}\right)\operatorname{Tr}(\nabla^2_{\mathbf{w}_k}\mathcal{L}) \ \textit{as } c\to\infty, \tag{2.2}$$

for classes where the frequency does not vanish too quickly, $\pi_k = \omega({}^1/_c)$.

The assumption that $c \to \infty$ is used to obtain a simple and interpretable equation in the correlation. In practice, $c > 10^3$ appears sufficient to make the dependence on π_k appear, as in Figures 2.7 and 2.8.

At initialization, Equation (2.1) shows that the Hessian blocks are uniform across classes while the gradients depend on π_k. If the data is uniform across classes ($\|\bar{\mathbf{x}}^k\| \approx \|\bar{\mathbf{x}}^{k'}\|$) while the frequencies differ by orders of magnitude, the the gradient blocks will mirror the class

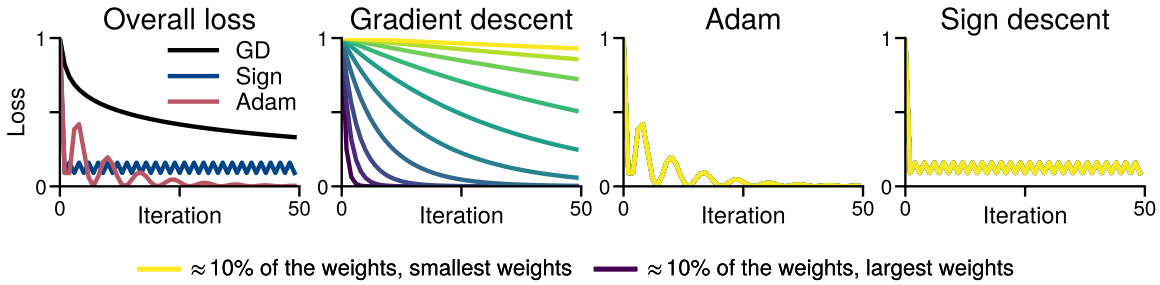


Figure 2.6: **Class-separation on the quadratic problem of Section 2.3.1 with weights $\pi_k \propto 1/k$.** GD fits functions with low weights more slowly, while Adam and sign descent have the same dynamics across all functions and all the lines overlap as every parameter w_i is initialized at $w_i = 1$.

frequencies for high-frequency classes where $\pi_k \gg 1/c$. This confirms the pattern observed at initialization in Figures 2.7 and 2.8. During training, Equation (2.2) indicates a correlation between gradient norm and Hessian trace if classes have similar values of $\|\bar{\mathbf{x}}^k\|$, $\mathrm{Tr}(\bar{\mathbf{H}}^k)$ and predicted probabilities p, confirming the behavior observed during training in Figures 2.7 and 2.8 for the high frequency classes. As Adam fits low-frequency classes faster in Figure 2.4, they have a value of p closer to 1 (shown in Appendix A.6) and deviate slightly from the trend in Figure 2.7, as expected from Equation (2.2).

We now give the main intuition and defer the derivation of the asymptotics to Appendix A.7. We ignore off-diagonal blocks here, as they are orders of magnitude smaller than diagonal blocks (Figure 2.9), and show in Appendix A.7.1 that they are expected to be small.

Proof idea. Our loss is $\mathcal{L}(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{W}, \mathbf{x}_i, \mathbf{y}_i)$, where ℓ is a softmax linear model,

$$\ell(\mathbf{W}, \mathbf{x}, y) = -\log(\sigma(\mathbf{W}\mathbf{x})_y), \qquad \text{with} \qquad \sigma(\mathbf{z})_k = \frac{\exp(\mathbf{z}_k)}{\sum_j \exp(\mathbf{z}_j)}. \tag{2.3}$$

Writing $\mathbf{p}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x})$ for the vector predicted probabilities, the gradient and Hessian blocks are

$$\nabla_{\mathbf{w}_k} \ell(\mathbf{W}, \mathbf{x}, y) = (y = k - \mathbf{p}(\mathbf{x})_k)\mathbf{x}, \qquad \nabla^2_{\mathbf{w}_k} \ell(\mathbf{W}, \mathbf{x}, y) = \mathbf{p}(\mathbf{x})_k(1 - \mathbf{p}(\mathbf{x})_k)\mathbf{x}\mathbf{x}^\top. \tag{2.4}$$

The contribution of a sample $(\mathbf{x}, y)$ to the gradient w.r.t. $\mathbf{w}_k$ primarily depends on whether the sample belongs to class k through the $y = k$ term, while the contribution to the Hessian block depends on whether the model assigns that sample to class k through $\mathbf{p}(\mathbf{x})_k$. At initialization, $\mathbf{p}(\mathbf{x})_k = 1/c$ for all samples, and averaging the terms in Equation (2.4) yields Equation (2.1). Highlighting this effect during training is more challenging due to the dependency on the predictions. However, if $\mathbf{W}$ start to assign samples to their correct classes (Assumption 1), we can obtain a similar decomposition as Equation (2.1). For a given class k, the probabilities for correct labels are all p while the probabilities for incorrect ones are bounded by $O(1/c)$, which vanishes in the limit of $c \to \infty$. □

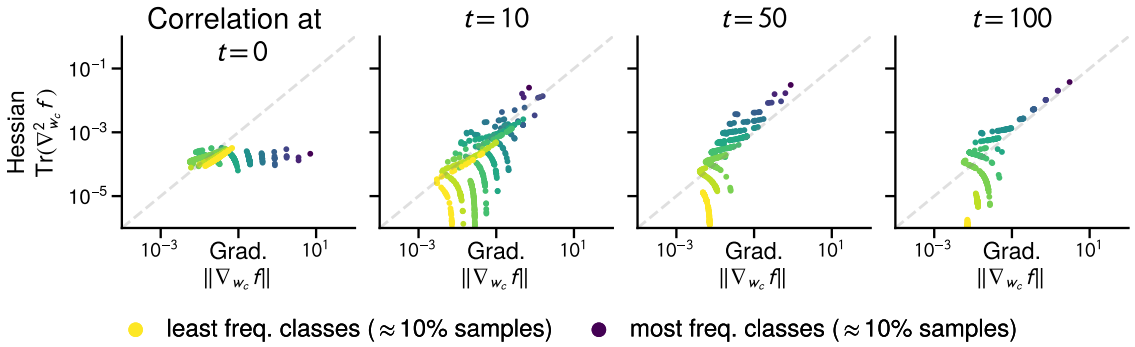


Figure 2.7: **The gradient norm and Hessian trace across blocks become correlated during training,** over the path taken by Adam in training the linear model of Figure 2.4. The blocks correspond to the rows $\mathbf{w}_1, ..., \mathbf{w}_c$ of the parameter matrix $\mathbf{W}$. The color indicates the class frequency, showing that lower (higher) frequency classes have smaller (larger) gradient norm and Hessian trace.

This assignment mechanism explains why the gradient, Hessian, and class probabilities can become correlated on the linear model. While the gradient does not directly approximate the Hessian, the main feature of the imbalance in the Hessian comes from the weighting by the class frequencies $\pi_1, ..., \pi_c$, which is present in both the gradient and the Hessian, as shown in Figures 2.7 and 2.9. This correlation is not a global property of the problem, as there are parameters for which the opposite pattern holds, see Appendix A.6, but it appears during training if the optimization algorithm makes progress. While the per-coordinate normalization of Adam or sign descent was not designed to specifically address class imbalance, they appear to benefit from this property to make faster progress.

Our results complement prior work on optimization with class imbalance on problems with two or few classes, which argued that the gradient is dominated by the majority class, and as a result is biased towards making progress on the majority class at the expense of the minority class (Anand et al., 1993; Ye et al., 2021; Francazi et al., 2023). While this explains why GD might not make fast progress on rare classes, it was not clear why this would lead to slow performance on average, especially under heavy-tailed imbalance where there is no "majority". Our results show that, in addition to imbalance in the gradients, class imbalance leads to optimization difficulties through imbalanced Hessians.

2.3.3 Improvement of sign-based approaches over gradient descent

While the above arguments provide a high-level intuition as to why the gradient might be a reasonable proxy for the Hessian, it remains difficult to formally describe this effect and prove the benefits of Adam over GD without strong assumptions. Doing so would require a fine-grained analysis of the dynamics, as the correlation only appears during training. To obtain a provable a guarantee highlighting the benefit of sign-based methods, we consider a

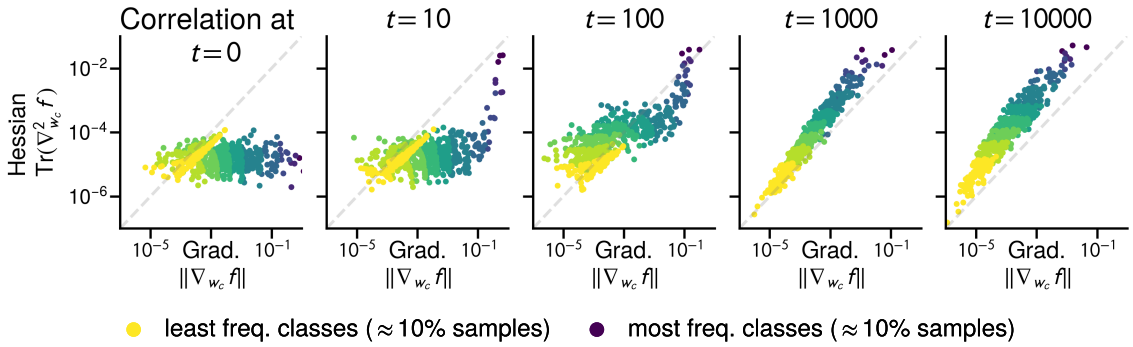


Figure 2.8: **The gradient-Hessian blocks also become correlated in the last layer of large models.** Reproducing Figure 2.7 on the transformer of Figure 2.1. Evolution of the gradient norm and Hessian trace for each row $\mathbf{w}_c$ of the last layer through optimization with Adam. Colors indicates class frequency. Lower (higher) frequency classes have smaller (larger) gradient norm and Hessian trace.

stripped-down problem where the only difficulty lies in the class imbalance:

This setting is trivial as a possible solution is $\mathbf{W} = \alpha\mathbf{I}$ with $\alpha \to \infty$, or taking one step of gradient descent with an arbitrarily large step-size. However, we will see that the dynamics with small step-sizes already exhibit the separation by class frequencies observed experimentally. In this simplified setting, we show that the continuous time variant of gradient descent, gradient flow, and sign descent as a proxy for Adam, obtain qualitatively different convergence rates (proof in Appendix A.8).

Theorem 3. *On the* simple imbalanced setting*, gradient flow and continuous time sign descent initialized at* $\mathbf{W} = 0$ *minimize the loss of class* k*,* $\ell_k(t) = -\log(\sigma(\mathbf{W}(t)\mathbf{e}_k)_k)$*, at the rate*

$$\textit{Gradient flow:} \quad \ell_k(t) = \Theta(1/\pi_k t), \quad \textit{Continuous time sign descent:} \quad \ell_k(t) = \Theta\big(e^{-ct}\big).$$

The difference between the sublinear rate of gradient flow $(1/t)$ and linear rate of sign descent (e^{-t}) is similar to existing results for separable logistic regression, where normalized updates converge faster as they keep increasing the margin despite small gradients (Nacson et al., 2019). While the setting studied here is separable, we still observe the separation across class frequencies on problems that are not separable, either because the problem has examples with different output for the same inputs, as in Figure 2.1, or when adding regularization, as in Appendix A.4.3. The novel element is that the convergence of gradient flow strongly depends on the class frequencies π, while the convergence of sign descent is independent of the class frequencies.

This setting is admittedly oversimplified and does not capture some of the features observed in our experiments. For example, in Theorem 3, the loss is monotonically decreasing for all

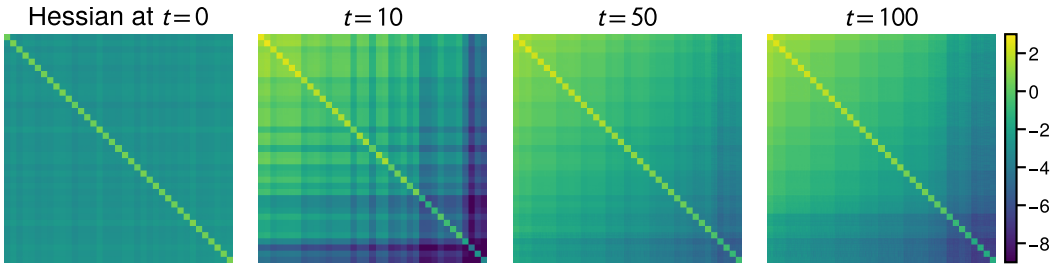


Figure 2.9: **The diagonal Hessian blocks are orders of magnitude larger than off-diagonal blocks.** Showing the magnitude of a subset of the Hessian blocks $(\log_{10}(\left|\mathrm{Tr}(\nabla^2_{ij}\mathcal{L})\right|))$ for a $[160 \times 160]$ subset of the Hessian, sampling 40 classes log-uniformly and 40 input dimensions uniformly.

classes. This no longer holds once we introduce a bias term and the loss from low-frequency classes will instead first increase, as can be seen for example in Figure 2.4. This setting is also biased towards sign descent, as the inputs are aligned with the basis vectors. Finally, the problem is inadequate to study large step-sizes, as it can be solved in one large step. On data with non-orthogonal classes, large step-sizes would lead to training instabilities and oscillations in the loss of frequent classes, as can be seen in Figures 2.2 to 2.5. Nevertheless, this result formally establishes the benefit of sign-based updates and we believe it captures the key difficulty encountered by GD under heavy-tailed class imbalance.

2.4 Discussion and limitations

Interaction with stochasticity. Our experiments include both stochastic and deterministic training regimes and show that stochasticity is not the cause of the slow performance of SGD on low-frequency classes, as it already appears between full batch GD and Adam. This observation is consistent with prior work showing that the performance gap between SGD and Adam on language transformers already appears with deterministic training (Kunstner et al., 2023). However, we do not attempt to quantify the interaction between stochasticity and class imbalance and leave it for future work.

Training performance vs. generalization. Our main focus is on optimization performance. Our observations need not generalize to the validation loss, especially in settings prone to overfitting, as good training performance may lead to overfitting on classes with few samples (Sagawa et al., 2020). However, some form of memorization might be needed in long-tailed settings (Feldman, 2020), and if SGD cannot even fit the training data, generalization cannot be good. On the transformer of Figure 2.1, we observe similar dynamics across frequencies on the validation loss, shown Appendix A.2.2. Training dynamics on the empirical and population loss are also often similar, particularly early in training (see, e.g., Nakkiran et al., 2021; Ghosh et al., 2022), and the one-pass training regime commonly used in large language models might mitigate those issues by blurring the line between train and test loss.

Additional difficulties due to text data. We study the effect of the distribution of the classes, the *next* token to be predicted, but other optimization difficulties might arise from the heavy-tailedness of text data. For example, the sequence of tokens used as inputs to the embedding layer are also heavy-tailed. This imbalance might lead to slow progress for rare tokens with GD, giving another potential cause for a performance gap. With stochastic training, this imbalance leads to sparse updates to the embedding layer, close to the setting that initially motivated AdaGrad (Duchi et al., 2011) and follow-up works attempting to correct this frequency imbalance in the input tokens (Défossez and Bach, 2017; Li et al., 2022). Beyond the inputs, full sentences (Williams et al., 2015) and latent rules or mechanisms required to understand a paragraph (Michaud et al., 2023) may also display heavy tails, and Adam could be beneficial if those are captured by intermediate layers (e.g., Meng et al., 2022; Wang et al., 2023b; Bietti et al., 2023). The choice of tokenization has also been shown to impact downstream performance, which has been attributed to the lack of samples on rare tokens (Gowda and May, 2020) and the improved efficiency of more uniform tokenizers (Zouhar et al., 2023). Our results indicate that tokenization also has a large impact on optimization performance.

Difficulties due to architectures. Beyond the class distribution, additional optimization difficulties may arise from the architectures, due to depth, signal propagation (Noci et al., 2022; He et al., 2023), vanishing gradients and higher order derivatives (Liu et al., 2020b; Orvieto et al., 2022). The simplified transformer of Ahn et al. (2023) also exhibits many of the difficulties observed in the literature on regression instead of a classification problem. However, a phenomenon similar to the assignment mechanism could still explain the benefit of Adam. The oscillations in the loss observed at the feature level by Rosenfeld and Risteski (2023) suggests a link between subsets of the samples and subsets of the parameters. For example, if a convolution filter detects a specific background color and captures a specific feature of the data, the magnitude of the gradient and Hessian at intermediate layers could be influenced by the relative frequency of the feature in the data, leading to another form of imbalance.

Recent ablations on the benefit of Adam for language transformer. Parallel to our work, recent investigations have looked into the benefits of Adam on language transformers. Zhang et al. (2024a) argue that the Hessian has a block-diagonal structure, with similar magnitude within blocks but very different magnitudes across blocks, and that Adam may improve performance by using a different step-size for different blocks. This hypothesis is supported by recent ablations studies. Zhang et al. (2024b) show that the element-wise preconditioning in Adam is not necessary and can be replaced by a single parameter across such blocks while maintaining performance, which they coin Adam-mini. Similarly, Zhao et al. (2024b) show that the performance of Adam can be recovered by training most of the network with (S)GD, except for the last layer and LayerNorm parameters. Both approaches still need to treat the last layer separately, either using a step-size for each row $\mathbf{w}_c$ of the last layer in the case of Zhang et al. (2024b) or by using Adam to train the last layer in the case of Zhao et al. (2024b). These observations complement our approach, which focuses on

the impact of heavy-tailed class imbalance on the last layer, and are consistent with our conclusion that one of the main benefit of Adam is to counteract the slow progress on rare classes by preconditioning the last layer.

2.5 Conclusion

We have shown that heavy-tailed class imbalance leads to a performance gap between (S)GD and Adam. This effect is reproducible across architectures and data types, but is most salient on language tasks which naturally exhibit heavy-tailed imbalance. As vision tasks are typically more uniform, imbalance is a key differentiating feature of the training difficulties in language tasks. The correlation between entries of the gradient and Hessian that occurs due to class imbalance provides justification for the intuition that Adam-like algorithms "adapt to curvature". We provide an explanation for how this correlation arises during training through the assignment mechanism and prove on a simplified setting that gradient descent performs poorly on low-frequency classes while sign descent does not.

Chapter 3

Feature Based Ill Conditioning

We show that feature normalization has a drastic impact on the performance of optimization algorithms in the context of graph neural networks. The standard normalization scheme used throughout the graph neural network literature is not motivated from an optimization perspective, and leads (S)GD to frequently fail. Adam does not fail, but is also negatively impacted by standard normalization methods. We show across multiple datasets and models that better motivated feature normalization closes the gap between Adam and (S)GD, and speeds up optimization for both.

3.1 Introduction

Data normalization is a standard step in data processing pipelines. Normalizing the features to have mean zero and unit variance is taught in introductory machine learning (e.g. Murphy, 2022, Ch. 10) and widely used in deep learning (e.g. LeCun et al., 2012)). But the increasing complexity of model architectures, data processing schemes, and optimization algorithms often takes the spotlight off these classical tools. Each subcommunity ends up developing its own set of standard tricks and practice, tuned to the specific problems encountered in vision, language, or graph data. While it makes sense that different tools and practices may be needed for different data types, it can also lead to situations where practices that are considered standard or common knowledge in one subcommunity are unknown or ignored in another. This paper highlights one such case. While much of the machine learning community is aware of the benefits of feature normalization, it appears to have been overlooked in standard benchmarks used to evaluate graph neural networks (GNNs). We show that paying attention to normalization has a substantial impact on training performance.

Our main contributions are:

- We highlight that fitting GNNs with gradient descent (GD) can be difficult, even for small problems included in standard GNN benchmarks, while Adam (Kingma and Ba, 2015) performs better.
- We show that this performance gap is due to (a lack of) data normalization. The gap is more pronounced on datasets with bag-of-words features, which lead to data matrices with heavy-tailed singular values. Normalizing *per-feature*, rather than the *per-row* that appears to be the default in GNNs, gives a simple fix that drastically speeds up optimization, regardless of the algorithm used.

Our contributions are summarized in Figure 3.1, which shows that GD fails to fit the model while other algorithms such as Adam work when using the *per-row* normalization. Normalizing *per-feature* leads to a dramatic performance improvement for both algorithms.

3.2 Effects of column- and row-normalization

While it is "common knowledge" in some circles that GD can struggle on GNNs, this problem is not well documented in the literature. There are a few statements that GD is considered slow or requires more tuning (Morris et al., 2024), and some empirical comparisons show poor performance for GD on some problems (e.g. You et al. 2020 or Izadi et al. 2020, Fig. 1). To our knowledge, there has not been a systematic investigation of the performance of optimization algorithms on GNNs. We first show that even well-studied problems common in GNN benchmarks can be hard to fit with GD.

We run full-batch gradient descent with momentum $\beta = 0.9$, and tune the step size with a grid search to minimize the training loss at the end of the given budget. We repeat this process with Adam, tuning the step size but leaving the momentum parameters to the

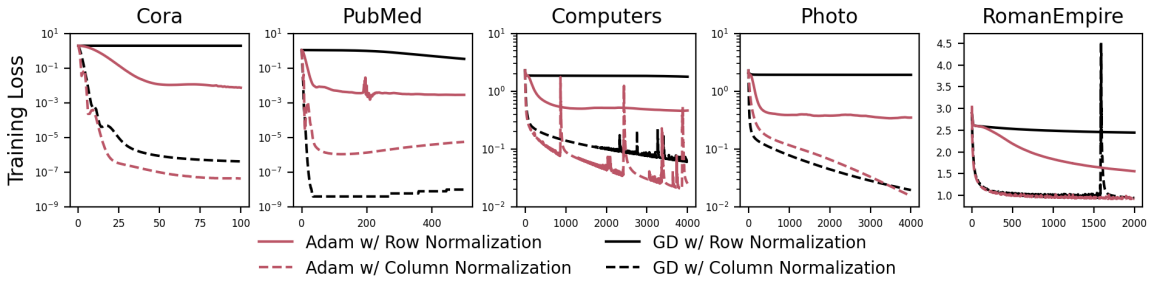


Figure 3.1: **GD struggles on GNNs with standard row normalization, but optimization becomes much easier with column normalization.** Training a Graph Convolutional Network (GCN) using **SGD** and **Adam**, using row normalization (solid) and column normalization (dashed). Similar results hold across models and datasets, as shown in Appendix B.2.

default $\beta_1 = 0.9$, $\beta_2 = 0.999$.

We show results using a Graph Convolutional Network (GCN) (Kipf and Welling, 2016) in Figure 3.1, using the row normalization used in most GNN data preprocessing (solid lines). Strikingly, GD makes almost no progress, across all datasets. Adam, on the other hand, does roughly work. However, when we change to *per-feature* normalization (dashed), both GD *and Adam* train drastically faster, mostly eliminating the performance gap. This effect is due to the features and can be replicated across architectures. We show similar effects with linear models in Figure 3.2, and Graph Attention (GAT) networks and GraphSAGE in Appendix B.2.

The per-feature normalization step used above common standardization method to achieve zero mean and unit variance. With `numpy`-like broadcasting rules, assuming $\mathbf{X}$ is an $n \times d$ matrix of n samples with d features each, the operation computes the normalized data matrix $\tilde{\mathbf{X}}$ as

$$\begin{aligned}
\tilde{\mathbf{X}} &\leftarrow (\mathbf{X} - \text{mean}(\mathbf{X}, \text{dim} = 0)) && \# \text{ mean } \textit{returns a vector of } d \textit{ means} \\
\tilde{\mathbf{X}} &\leftarrow \tilde{\mathbf{X}}/\text{std}(\tilde{\mathbf{X}}, \text{dim} = 0) && \# \text{ std } \textit{returns a vector of } d \textit{ standard deviations.}
\end{aligned}$$

We call this *column normalization*, as it normalizes each column of features independently.

The preprocessing step found in many GNN problems, *row normalization*, differs in two ways. Rather than ensure the data has mean zero and unit standard deviation, it normalizes the data to be non-negative and sum to 1. More importantly, it operates on each row (the features for one node in the graph) independently. That is,

$$\begin{aligned}
\tilde{\mathbf{X}} &\leftarrow (\mathbf{X} - \min(\mathbf{X}, \text{dim} = 1)) && \# \min \textit{ returns a vector of } n \textit{ minima} \\
\tilde{\mathbf{X}} &\leftarrow \tilde{\mathbf{X}}/\text{sum}(\tilde{\mathbf{X}}, \text{dim} = 1) && \# \text{ sum } \textit{returns a vector of } n \textit{ sums.}
\end{aligned}$$

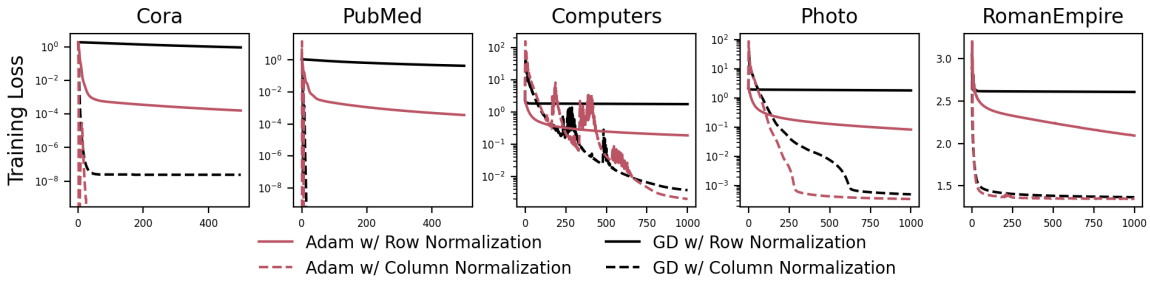


Figure 3.2: **On a linear model, both Adam and GD are improved by by switching from row normalization to column normalization.** Training a Linear GCN using **SGD** and **Adam**, using row normalization (solid) and column normalization (dashed).

This row normalization operation is implemented in PyTorch Geometric (Fey and Lenssen, 2019) as `NormalizeFeatures`, DGL (Wang et al., 2019, `RowFeatNormalizer`) and Spektral (Grattarola and Alippi, 2021, `NormalizeOne`). This row-normalization may have been inherited from Kipf and Welling (2016) who used it on datasets where node are represented by bag-of-word features on a textual representation of the node.[1] Those features take value in $\{0, 1\}$ to indicate whether a word is present. On this data, subtracting the minimum does nothing, while the sum-to-one normalization changes the representation of a sample from the sum of the word embbedings to their average.

Both normalization procedures address orthogonal issues (samples of different scales vs. features of different scales). They can, and depending on the application possibly should, be combined. The issue is that row-normalization alone appears to have become a standard in GNN benchmarks. It appeared in problems used to establish foundational GNN methodology (Kipf and Welling, 2016; Veličković et al., 2018) and continues to be used in modern evaluations (Luo et al., 2024). Others have also called attention to data normalization in GNNs, as Tönshoff et al. (2024) criticized the benchmark Dwivedi et al. (2022) for not normalizing vision data following best practices (i.e., normalizing the channels). But far more attention is devoted to normalization procedure within the network (Ioffe and Szegedy, 2015; Ulyanov et al., 2016; Ba et al., 2016; Zhao and Akoglu, 2020; Cai et al., 2021; Dwivedi et al., 2022) than the normalization of the input data. By ignoring the inputs, we are routinely solving much harder optimization problems than we need to.

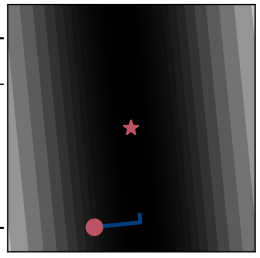
[1] Kipf and Welling (2016) motivate this normalization by saying that the use the parameter initialization scheme of Glorot and Bengio (2010) and "accordingly (row-)normalize input feature vectors" (and indeed they use $n \times d$ matrices). We can find no reference to row normalization by Glorot and Bengio; rather, their scheme is explicitly motivated by an assumption that "input features variances are the same," i.e. *column* normalization.

3.3 Effect of normalization on imbalanced features

To highlight the benefits of column normalization and the potential pitfall of using only row-normalization, consider the used-car dataset below. It has three features, mileage, sale price and year of construction, which span orders of magnitude. Fitting a linear regression on this input data would give give a poorly conditioned quadratic optimization problem, leading to slow optimization.

Raw data X

Mileage (km)	Sale price ($)	Year
100 000	16 999	2008
130 000	14 999	2013
150 000	11 999	2012
200 000	17 999	2014

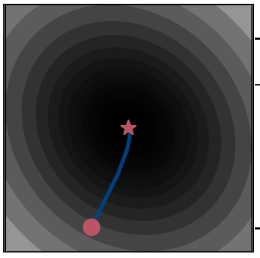


Normalized loss Landscape. 2d-slice of the loss landscape on the first two coordinates for a quadratic $\mathcal{L}(\mathbf{w}) = \frac{1}{2}\|\mathbf{X}\mathbf{w}\|^2$. In blue, 100 iterations of GD starting at ●. Minimum at ⋆.

The quadratic optimization problem resulting from a linear regression with this input data has a condition number of $\approx 10^5$. The reason for this large condition number is features of different scales. Column normalization fixes this problem and yields a condition number of ≈ 10. As gradient descent requires $O(\kappa \log(\epsilon))$ iterations to achieve an error of ϵ (in the worst-case) (Karimi et al., 2016), column-normalization gives a speed up in optimization performance of 10^4, see below. Row-normalization does not fixes this imbalance, as shown below, and using it on generic data introduces other problems.

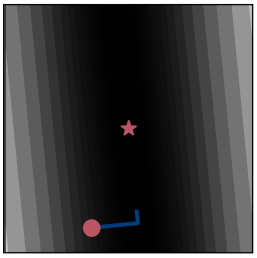
Column-normalized

Mileage	Price	Year
-1.24	0.66	-1.64
-0.41	-0.22	0.54
0.14	-1.53	0.11
1.51	1.09	0.98



Row-normalized

Mileage	Price	Year
0.87	0.13	0
0.91	0.09	0
0.94	0.06	0
0.93	0.08	0



Problem 1: Row-normalization does not normalizing the features. Despite transforming the data (and being called `NormalizeFeatures` in some libraries), row-normalization does not adress the problem of having very different scales across features. The optimization problem is not made any easier, and can be made more difficult. In Appendix B.3, we show that **row normalization can lead to worse performance than not preprocessing the data at all** across models and datasets.

Problem 2: Destroying information. Column normalization can be implemented as a linear operation, $\tilde{\mathbf{X}} = \mathbf{X}\mathbf{R} + \mathbf{b}$, where $\mathbf{R} : \mathbb{R}^{d \times d}$ and $\mathbf{b} : \mathbb{R}^d$. Thus, linear models (with biases) on the raw data and the normalized data have equivalent expressivity.[2] Row normalization,

[2]Generalization, however, may differ due to regularization. This can be explicit, by changing the impact of l2-regularization, or implicit, by changing which minimum-norm solution the algorithm converges to.

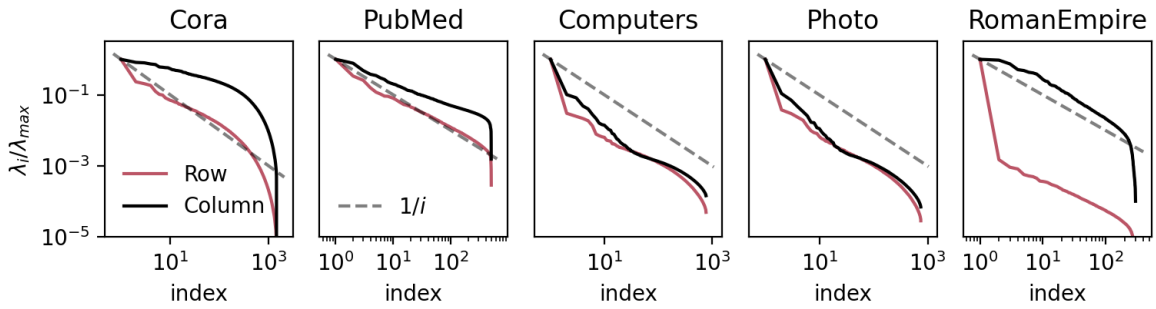


Figure 3.3: **Column Normalization improves conditioning on a linear model with GNN datasets.** Normalized eigenvalues ($\lambda_i/\lambda_{\max}$) of the Hessian of a linear model with row- and column-normalization. Dashed line includes $1/i$ for scale. The relative eigenvalues are larger when normalizing by column, indicating faster convergence with GD.

however, can destroy information. On the example above, the *year* feature becomes exactly 0 and the dataset is now only 2-dimensional. This is not an issue when each samples does not have the same minimum feature, for example if all features are binary as the subtraction of the minimum is a no-op. But on arbitrary dataset, as above, row-normalization can make it harder to fit the data.

3.4 Features with different scales in GNN benchmark problems

Typical graph benchmark datasets exhibit an imbalance in the features. However, this imbalance does not appear through features of different scales, as in Section 3.3, but through features of different frequencies. Many datasets use bag-of-words features, binary vectors encoding whether a word is present. Those words have different frequencies (typically roughly a power law), which leads to data matrices $\mathbf{X}^\top\mathbf{X}$ that have a very imbalanced diagonal and imbalanced eigenvalues (if $\mathbf{x}_j$ is binary, $\frac{1}{n}(\mathbf{X}^\top\mathbf{X})_{jj}$ is the frequency of $\mathbf{x}_j = 1$). Row normalization does not address this frequency-based imbalance while column normalization drastically improves performance.

To illustrate this effect, we give in Figure 3.3 a visualization of the spectrum of the eigenvalues of a linear (one layer, no activation function) GNN using row- and column-normalization. We show the normalized $\lambda_i/\lambda_{\max}$, which are indicative of the convergence speed of gradient descent along the ith eigenvector, as the error contracts by a factor of $(1 - \lambda_i/\lambda_{\max})$ at each step if the step-size if $\alpha = 1/\lambda_{\max}$. Column normalization produces values of $\lambda_i/\lambda_{\max}$ that uniformly higher than row-normalization. While the eigenvalues only provide a direct argument for why normalization helps on linear models (such as in Figure 3.2), it helps us understand where the difficulty of the problem arises and suggests why the optimization performance improves on the deep models of Figure 3.1.

3.5 Discussion, Related Work, and Future Work

Limitations and Future Work. We focused solely on the training dynamics of GNNs with classical models and datasets. We plan to extend this study to consider generalization and more modern models. Our focus is on deterministic optimization, which is common on graph networks as it is difficult to obtain estimates of the stochastic gradient. However, it is possible that column-normalization might increase the variance during stochastic optimization, as rare features are divided by a small standard deviation, increasing their magnitude. We also do not claim that poorly conditioned inputs are the *only* reason Adam consistently outperforms GD on GNNs; there are likely other properties of model architectures and datasets that influence the optimization dynamics of both algorithms. We hope to explore these further in future work.

GNNs and Normalization. Graph Neural Networks such as GCNs (Kipf and Welling, 2016) and GATs (Veličković et al., 2018) have been used in domains with graph structure such as proteins, co-purchase networks, citation networks, and traffic analysis (Hu et al., 2021; Takac and Zabovsky, 2012; Derrow-Pinion et al., 2021). You et al. (2020) found that Adam usually performs better than GD, but indicated this might be a problem with hyperparameter tuning. The difficulties due to data normalization might have lead to the more widespread adoption of Graph Transformers (e.g. Rampášek et al., 2022; Shirzad et al., 2023; Shirzad et al., 2024; Deng et al., 2024), which use batch normalization (Ioffe and Szegedy, 2015) rather than layer normalization (Ba et al., 2016). Batch norm is very similar to column normalization while layer norm is closer to row normalization, and might mitigate some of the issues highlighted here.

Adam versus (S)GD. Duchi et al. (2011) claimed AdaGrad (a predecessor to Adam) has better performance over GD on sparse data, using bag-of-words features as an example. This is consistent with our results, but we highlight that it is likely the imbalanced features caused an ill-conditioning problem, rather than sparsity. Column normalization densifies features, but helps improve conditioning. Other analysis has shown that heavy tail language based *labels* cause (S)GD to fail due to a poorly conditioned Hessian, while Adam is unaffected (Chapter 2). Adam has also been shown to follow a trajectory over which the robust condition number (see Figure 3.3) is smaller than (S)GD (Jiang et al., 2023).

Chapter 4

Basis Sensitivity

Despite its widespread adoption, Adam's advantage over stochastic gradient descent (SGD) lacks a comprehensive theoretical explanation. This paper investigates Adam's sensitivity to rotations of the parameter space. We observe that Adam's performance in training transformers degrades under random rotations of the parameter space, indicating a crucial sensitivity to the choice of basis in practice. This reveals that conventional rotation-invariant assumptions are insufficient to capture Adam's advantages theoretically. To better understand the rotation-dependent properties that benefit Adam, we also identify structured rotations that preserve or even enhance its empirical performance. We then examine the rotation-dependent assumptions in the literature and find that they fall short in explaining Adam's behavior across various rotation types. In contrast, we verify the orthogonality of the update as a promising indicator of Adam's basis sensitivity, suggesting it may be the key quantity for developing rotation-dependent theoretical frameworks that better explain its empirical success.

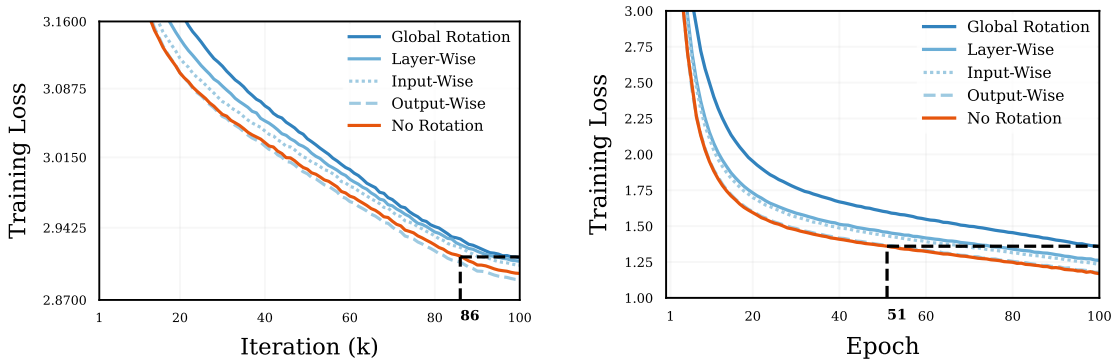
4.1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities as their scale grows (Brown et al., 2020; Kaplan et al., 2020). However, this unprecedented growth in model scale has led to a proportional increase in the economic (Dong and Xie, 2024; Sharir et al., 2020; Varoquaux et al., 2024) and environmental (Luccioni et al., 2023; Luccioni et al., 2019) costs associated with their training. Despite this clear motivation, Adaptive Moment Estimation (Adam) (Kingma and Ba, 2015) has persisted as the go-to optimizer especially for language models, with only minor modifications such as AdamW (Loshchilov and Hutter, 2019) becoming widely adopted since Adam's inception. This success has prompted extensive research to provide theoretical justification for Adam's performance. While the original convergence proof for Adam was later found to be flawed (Rubio, 2017), recent studies have proposed rigorous convergence proofs under plausible assumptions (Li et al., 2023b; Chen et al., 2019; Défossez et al., 2022).

However, these proofs do not elucidate Adam's advantages over SGD when training transformer models (Vaswani et al., 2017). Numerous works attempted to explain Adam's superiority, employing diverse assumptions and analytical frameworks (Zhou et al., 2024; Pan and Li, 2023; Zhang et al., 2024a; Kunstner et al., 2024). The heterogeneity of these approaches has led to a lack of consensus on which theoretical explanations most accurately capture the fundamental mechanisms underlying Adam's improved performance. For instance, Zhang et al. (2020b) suggests it stems from enhanced robustness to heavy-tailed noise, while Kunstner et al. (2023) argues it plays no role.

This study focuses on a fundamental distinction between Adam and SGD: Adam's dependency on the coordinate system. SGD is rotation-equivariant, meaning if the loss landscape is rotated, the resulting optimization trajectories from SGD will be the same up to that rotation. In contrast, Adam produces substantially different trajectories. Although the sensitivity of adaptive methods to rotations of the parameter space is well established (Duchi et al., 2011), it remains unclear what properties of a rotation benefit or hinder performance, particularly in practical neural network training. Our experimental investigation reveals that Adam's performance when training transformers empirically degrades when the objective function undergoes random rotations (Figure 4.1). This result demonstrates that Adam's effectiveness crucially depends on the canonical basis, challenging the adequacy of many existing theoretical frameworks used to analyze Adam's performance. Assumptions employed in the literature (Appendix C.6) are typically rotation-invariant, and thus the resulting frameworks are agnostic to the basis, preventing them from fully capturing or justifying Adam's empirical advantages.

Beyond theoretical motivations, recent studies have shown that applying rotations to the canonical basis can significantly enhance the performance of Adam and other optimizers (Vyas et al., 2025; Gupta et al., 2018; Jordan et al., 2024). However, these rotations are often designed based on intuition and heuristics rather than systematically understanding their impact. By identifying the key properties that make a basis advantageous for Adam, we can



(a) GPT2 (124M)

(b) ViT/S (22M)

Figure 4.1: **Adam's performance degrades under certain random rotations of the parameter space, demonstrating its dependence on the standard basis.** (a) For GPT2, global rotations lead to a 16% slowdown in training. (b) ViT experiences a more dramatic 96% slowdown under global rotations. Performance is preserved under output-wise rotations but progressively worsens with input-wise, layer-wise, and global rotations, revealing Adam's increasing sensitivity to coordinate changes of broader scopes. Experimental details are provided in Section 4.3.1.

develop more principled approaches to constructing optimal rotations that may outperform the existing rotation-based methods.

To understand the relationship between basis orientation and Adam's performance, we address two key questions:

Q1. How do various types of rotations influence Adam's performance in practice?

We investigate **Q1** by conducting experiments in Section 4.3, examining Adam's convergence when rotating specific regions of the parameter space. We also identify some rotations that preserve or enhance Adam's performance. These findings provide a more nuanced picture of Adam's adaptive behavior and its dependency on the basis.

Finally, as rotation-invariant theoretical framework cannot fully capture Adam's advantage, we turn to rotation-dependent assumptions existing in the literature, and seek to answer:

Q2. What rotation-dependent assumptions adequately capture Adam's behavior under rotations?

Section 4.4 examines three rotation-dependent assumptions used in literature in this context: L_∞ bounded gradients, Hessian block-diagonality, and L_∞-smoothness. Our analysis reveals that none of these conditions fully capture Adam's behavior under rotations. Recently,

Muon (Jordan et al., 2024) achieved strong performance by approximating orthogonalized gradients for each layer. Inspired by this, we measure the orthogonality of Adam's weight matrix updates for each layer (up to scalar multiplication) and find it closely aligns with performance across various rotations.

We summarize our key contributions:

- **Analysis of Adam's sensitivity to various scopes of parameter space rotations in neural network training**: We conduct in Section 4.3.1 an empirical study demonstrating Adam's sensitivity to random parameter space rotations in practical training. We found a clear correlation between rotation scope (e.g., global, layer-wise) and performance, where broader scope leads to greater degradation. In Section 4.3.2 we also employ a structured, SVD-based rotation inspired by Zhao et al. (2024a) that improves Adam's convergence.
- **Challenging existing rotation-dependent assumptions**: We assess the applicability of rotation-dependent properties in the literature by examining them jointly with our experimental study, and find that existing theoretical frameworks are not properly equipped to understand the beneficial properties of Adam.
- **Verifying a better rotation-dependent quantity:** Given that SVD-based rotations improve performance, we analyze the singular values of the layer updates and find that their orthogonality is a strong predictor of Adam's performance across different bases. We also draw a connection to the Muon optimizer (Jordan et al., 2024), which approximates orthogonalized updates, and provide additional empirical support for its underlying intuition.

4.2 Preliminaries

Let $f : \mathbb{R}^d \to \mathbb{R}$ be the loss of a neural network with d trainable parameters. Stochastic optimization algorithms approximate $\arg\min_{w\in\mathbb{R}^d} f(w)$ by only accessing independent stochastic functions f_B that depend on a stochastic minibatch B following some data distribution $\mathcal{D}$ such that $\forall w \in \mathbb{R}^d, \mathbb{E}_{B\sim\mathcal{D}}[f_B(w)] = f(w)$. Our study examines the optimization process under **rotations of the parameter space**. More formally, let $SO(d)$ be the set of rotation matrices,

$$SO(d) = \left\{\mathbf{R}\in\mathbb{R}^{d\times d}\colon \mathbf{R}^\top\mathbf{R} = \mathbf{R}\mathbf{R}^\top = \mathbf{I}, \det(\mathbf{R}) = 1\right\}. \tag{4.1}$$

Instead of directly optimizing f, we consider its rotated counterpart $f^{(\mathbf{R})} : w \to f(\mathbf{R}^\top w),\ \mathbf{R} \in SO(d)$. This transformation rotates the coordinate system while preserving the geometry of the optimization landscape.

We say that an optimizer is **rotation equivariant** if, after a rotation of the parameter space, its trajectories are equally rotated.

Definition 1 (Rotational equivariance)**.** *Consider an optimization algorithm $\mathcal{A}$ applied to the function f, generating iterates $w_{t+1} = \mathcal{A}(\{w_i\}_{i=0\ldots t}, f, t)$. We say that the optimization algorithm is* ***rotation equivariant*** *if it satisfies, $\forall \mathbf{R} \in SO(d)$, $\mathbf{R}w_{t+1} = \mathcal{A}(\{\mathbf{R}w_i\}_{i=0\ldots t}, f^{(\mathbf{R})}, t)$, where $f^{(\mathbf{R})} : w \to f(\mathbf{R}^\top w)$ is the rotation of f.*

Proposition 1. *SGD with momentum is rotation-equivariant.*

4.2.1 Rotational Equivariance of SGD

The rotation equivariance of SGD(M) is a straightforward result as the gradient operator is rotation equivariant; we provide the proof in Appendix C.4 for clarity. In contrast , Adam is not rotation equivariant, due to its element-wise division (Figure 4.2).

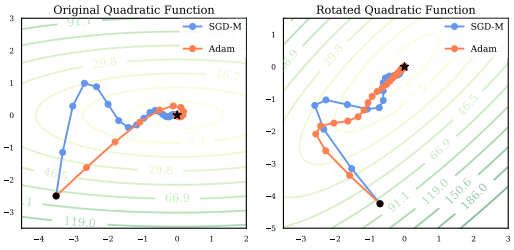


Figure 4.2: Trajectories of SGD-M and Adam on a quadratic under two different rotations. **SGD-M maintains the same trajectory up to rotation; Adam does not.**

4.2.2 Training Neural Networks in Rotated Parameter Spaces

A crucial aspect of our study is the empirical evaluation of Adam's performance under parameter space rotations. Our approach (Figure 4.3) maintains the weights w_t in the standard basis and perform Adam's optimization steps in the rotated space. This allows us to leverage existing neural network frameworks for forward and backward propagation while examining Adam's behavior under rotation.

Rotations in high dimension. It is computationally intractable to operate with full $d \times d$ rotation matrices due to the size of modern neural networks. We employ a composite approach that combines block-diagonal rotations with strategic permutations to circumvent this limitation while preserving the essential characteristics of uniformly sampled rotations, effectively emulates the statistical properties of full-scale rotations. A detailed description and ablation studies are provided in Appendix C.1.

Numerical considerations. Neural network training is sensitive to numerical precision (Li et al., 2018; Wang et al., 2018; Sun et al., 2022), and it is crucial to ensure that rounding errors from rotations do not significantly confound the impact of the rotation. In particular, we apply rotations in single precision, and we refrain from using FlashAttention (Dao et al., 2022), which was found to increase numeric deviations (Golden et al., 2024). We validate our methodology with ablations on SGD rotation equivalence, various rotation dimensions, and the use of FlashAttention in Appendix C.1.

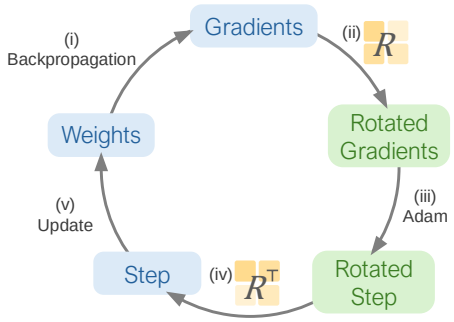


Figure 4.3: Methodology to train neural networks under parameter space rotations. (i) Forward and backward passes in the standard space to retrieve the gradients. (ii) The gradients are rotated using $\mathbf{R}$. (iii) Adam receives the rotated gradients and produces an update $\Delta w^{(\mathbf{R})}$ in the rotated space. (iv) $\Delta w^{(\mathbf{R})}$ is rotated back to the original space using $\mathbf{R}^\top$. (v) The parameters are updated with $\mathbf{R}^\top \Delta w^{(\mathbf{R})}$.

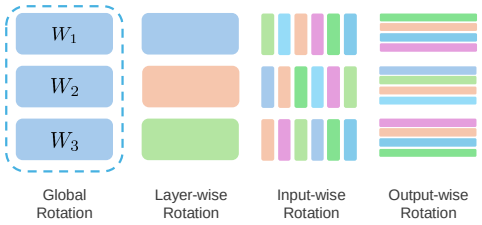


Figure 4.4: Illustration of different rotation scopes for a model with weights $\mathcal{W} \triangleq \{\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3\}$. Global rotation rotates the entire parameter space at once, layer-wise only performs rotations within each layer subspace, and input-wise (resp. output-wise) rotates within the weights originating from a same input neuron (resp. leading to a same output neuron).

4.3 Influence of Rotation on Adam's Efficiency

This section examines the effect of random rotations from neuron-wise to global (Section 4.3.1), showing that broader rotations degrade performance, while smaller-scale rotations have little to no impact. We then demonstrate that specific structured rotations can enhance Adam's performance.

4.3.1 Random Rotations

We first study the effect of four types of random rotations on Adam's performance (Figure 4.4): **Global** (entire parameter space), **Layer-wise** (per-layer subspaces; in transformers, keys/queries/values are treated separately), **Output-wise** (rotate weights where connections terminate at the same neuron in the subsequent layer), and **Input-wise** (rotate weights originating from the same neuron).

Experimental setting. We conduct experiments across three distinct settings spanning both transformer and non-transformer architectures, and language and vision tasks. Technical details and hyperparameters are provided in Appendix C.2.

- **Language modeling (GPT-2, Fig. 4.1(a)):** 124M-parameter decoder-only Transformer (Radford et al., 2019) trained on OpenWebText (Gokaslan and Cohen, 2019).
- **Image classification (ViT, Fig. 4.1(b)):** 22M-parameter Vision Transformer (ViT/S) (Dosovitskiy et al., 2021) evaluated on ImageNet-1K (Deng et al., 2009).

- **Image classification (ResNet, Fig. C.2):** ResNet-50 (He et al., 2016) on ImageNet-1K, where SGD often outperforms Adam (Keskar and Socher, 2017; Wilson et al., 2017).

Results. We make several key observations.

- Adam's performance degrades under global rotations across all settings, confirming that the standard basis possesses advantageous properties.
- The performance further degrades with broader rotation scopes. Layer-wise rotations, which preserve some basis structure, consistently outperform global rotations, highlighting the importance of local coordinate alignment.
- ResNets exhibits minimal performance degradation under rotations. This reduced sensitivity suggests Adam obtains limited benefit from the standard basis structure in ResNets, explaining its historically smaller marginal gain in performance in training these networks.
- Output-wise rotations show no degradation across all settings, with GPT2 even slightly improving. This suggests that Adam's adaptivity within output neurons is minimal, supporting recent approaches to reduce redundancy in Adam's second moments (Zhang et al., 2024b).

Previous works (Zhang et al., 2024a; Zhang et al., 2024b) have highlighted the heterogeneity across different parameter block types in Transformer architectures. In Appendix C.3, we restrict rotations to specific parameter types and study their individual impact on Adam's rotation sensitivity.

4.3.2 Investigating the Performance-Improving Rotations

Inspired by GaLore (Zhao et al., 2024a), which uses low-rank Singular Value Decomposition (SVD) to compress optimizer states, we extend this concept to full-rank SVD to rotate the parameter space. Our approach decomposes the gradient matrix $\mathbf{G}$ of each layer into $\mathbf{G} = \mathbf{USV}^\top$. This decomposition yields a natural rotation of the parameter space through the transformation $\mathbf{G} \to \mathbf{U}^\top\mathbf{GV}$, which corresponds to an output-wise rotation (via $\mathbf{U}^\top$) and an input-wise rotation (via $\mathbf{V}$). We train a GPT2 model under the same conditions as in Section 4.3.1, but in this SVD-rotated space. We update the SVD decompositions every 250 steps. The full AdamW with SVD Rotations is described in Appendix C.4.

Fig. 4.5 show that Adam's performance improves under SVD-based rotations, with a low computational overhead. These results highlight the potential of rotation-based approaches and motivate a more principled understanding of how basis orientation affects optimizer behavior. Instead of relying on intuition or heuristics, we advocate for theory-driven design grounded in rotation sensitivity.

To further understand this behavior, Figure 4.6 shows second moment distributions after training under various rotations. Global random rotations yield more concentrated second

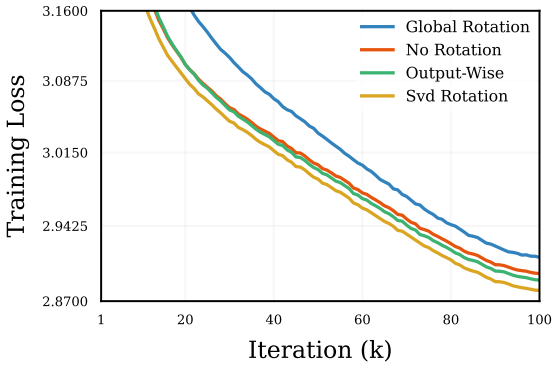


Figure 4.5: Performance of GPT2 trained with Adam in SVD-rotated space, without rotations, with random output-wise rotation and with random global rotation. **The rotations computed with SVD lead to sizeable improvement.**

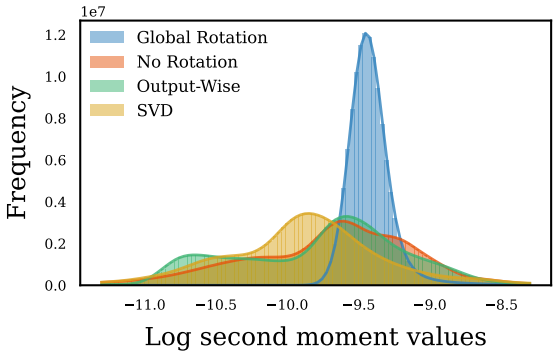


Figure 4.6: Distribution of second-moment values for the final checkpoint of a GPT2 model trained with Adam in various rotated spaces. **Second moments are more concentrated under random rotations, indicating reduced adaptivity.**

moments, implying less variation in effective learning rates and reduced adaptivity, which explains Adam's degraded performance. However, the benefits of SVD rotations are not apparent from second moments alone, suggesting a more subtle relationship between the parameter space and Adam's adaptive behavior.

4.4 Examining Rotation Dependent Assumptions

We have established that Adam's performance depends heavily on the choice of basis. Rotation-invariant analyses yield identical guarantees for all bases, failing to capture performance gaps between rotations. A rotation-dependent assumption is necessary to explain Adam's practical advantage. In this section, we examine rotation-dependent assumption adequacy jointly with our GPT-2 experiments.

4.4.1 Adequacy of existing assumptions in theoritical frameworks

While rotation-invariant assumptions dominate optimization literature, some frameworks incorporate rotation-dependent properties. This subsection examines three existing assumptions and whether they adequately capture Adam's rotation dependency. In particular, we focuses on two aspects: (i) **Practical Feasibility.** The assumption must be realistic in practical settings. (ii) **Alignment with Adam's Performance.** An adequate property should have favorable constants under rotations that improve performance and break down (or have unfavorable constants) under rotations that hinder performance. An ideal theoretical convergence analysis should be based on realistic assumptions that relate the problem's characteristics to the optimizer's performance, where faster theoretical rates correspond to

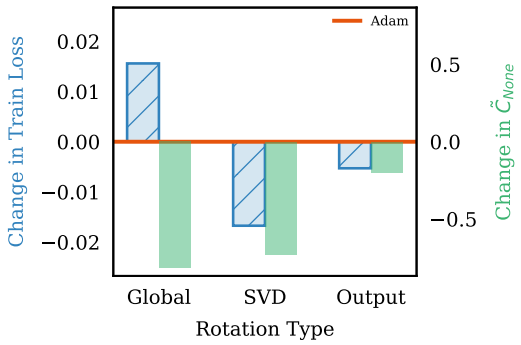


Figure 4.7: Empirical L_∞ gradient bound $\tilde{C}$ over 1000 stochastic gradients at the last checkpoint for Global, SVD, and output-wise rotations, presented as differences from the non-rotated baseline. **The trend disagrees with Adam's performance, especially under global rotation.**

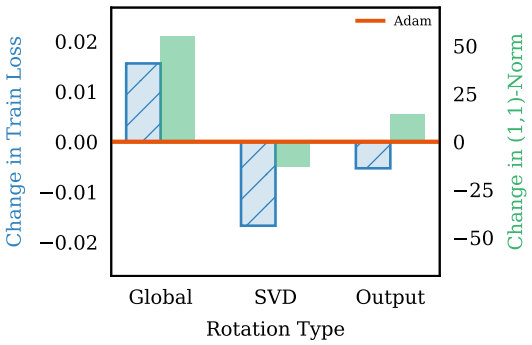


Figure 4.8: Estimated (1,1)-norm of the Hessian and final accuracy for Global, SVD, and output-wise rotations, presented as differences from the non-rotated baseline. **The $(1,1)$ norm correlates with Adam's performance on global and SVD rotations, but not on output-wise.**

better practical performance.

L_∞ bounded gradients Reddi et al. (2018) and Kingma and Ba (2015) assume a bound on the L_∞ norm of stochastic gradients,

$$\forall w \in \mathbb{R}^d,\ \|\nabla f_B(w)\|_\infty \leq C \quad \text{almost surely.} \tag{4.2}$$

The constant C depends on the basis as the L_∞ norm is not preserved under rotations. To evaluate this assumption, we compute the empirical bound on the rotated gradients,

$$\tilde{C_{\mathbf{R}'}}(\mathbf{R}) := \max_{B_i} \|\nabla f_{B_i}^{(\mathbf{R})}(w_{\mathbf{R}'})\|_\infty,$$

where $w_{\mathbf{R}'}$ denotes the last checkpoint obtained by running Adam under rotation $\mathbf{R}'$. The maximum is over 1000 stochastic minibatches B_i, across different rotations $\mathbf{R}$ (see Section 4.3). Figure 4.7 reveals that $\tilde{C}$ significantly decreases under random global rotations, predicting better performance, but we observe degradation in Adam's convergence. This discrepancy shows that the L_∞ gradient bound fails to capture the beneficial properties of the basis for Adam.

L_∞-smoothness and $(1,1)$-norm L_∞-smoothness was recently shown to guarantee the convergence of Adam, and presented as a potential key property of the basis (Xie et al., 2025; Balles et al., 2020). We first remember its definition.

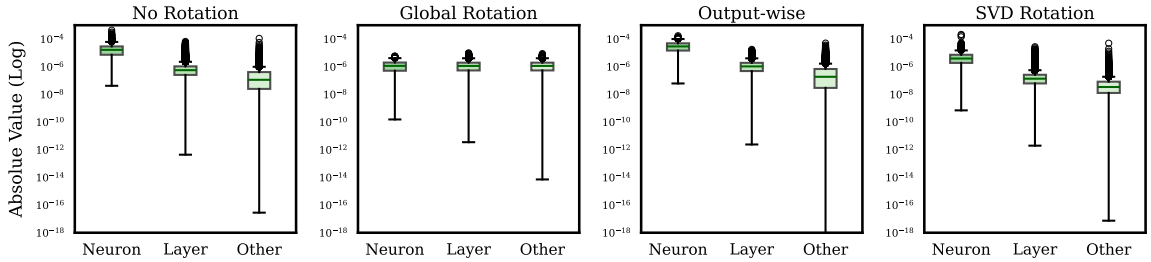


Figure 4.9: Distribution of Hessian values within output neuron, layer, and non-layer in the second Transformer block attention projection layer. **In no rotation, values within neuron are of magnitude higher than others, presuming a possible block-diagonal structure.** The structure is preserved in SVD and output-wise rotations, and lost in global rotation.

Definition 2. *A function f is C-smooth wrt. $\|\cdot\|_\infty$ if $\|\nabla f(x) - \nabla f(y)\|_1 \leq C\|x-y\|_\infty \; \forall \; x, y \in \mathbb{R}^d$.*

Given the challenges in directly estimating the L_∞-smoothness constant, Xie et al. (2025) proposed using the (1,1)-norm of the Hessian as a surrogate, defined as:

$$\|H\|_{(1,1)} := \sum_{m=1}^{M} \sum_{n=1}^{N} |H_{mn}|,$$

where H_{mn} represents the element at the m-th row and n-th column of the Hessian matrix. Notably, they observed a degradation in their estimate of $\|H\|_{(1,1)}$ under global random rotations. However, it remains unclear whether this degradation is a universal phenomenon for all rotations of the parameter space or if it specifically correlates with Adam's performance. To investigate this, we estimate the $(1,1)$-norm by averaging the L_1 norm of Hessian rows, sampled using the methodology described in Section 4.4.1. Figure 4.8 illustrates the change in $\|H\|_{(1,1)}$ under global, SVD, and output-wise rotations.

Under global rotations, we confirm the $(1,1)$-norm degradation reported in (Xie et al., 2025), while SVD rotations improve it in line with Adam's performance gains, suggesting a link between this geometric property and optimizer efficiency. Output-wise rotations yield slight performance gains but reduced $(1,1)$-norm, indicating that the metric does not capture all relevant factors. Overall, the $(1,1)$-norm shows promise as a rotation-sensitive indicator, especially under global and SVD rotations, but its limitations motivate the development of refined or complementary measures.

Block-diagonality of the Hessian A common hypothesis for understanding Adam's behavior is that the Hessian is well-approximated by a block-diagonal matrix (Zhang et al., 2024a). Then, random rotations likely disrupt block-diagonality and hinder convergence, while rotations within diagonal blocks preserve the structure, explaining the stable performance under output-wise rotations.

To examine this assumption's validity, we sample rows of the Hessian of $f^{(\mathbf{R})}$ at a checkpoint $w_{\mathbf{R}'}$:

$$\mathbf{r_i}(\mathbf{R}, \mathbf{R}') = \frac{1}{k}\sum_{j=1}^{k} \nabla^2 f_{B_j}^{(\mathbf{R})}(w_{\mathbf{R}'})_{[i,:]} = \frac{1}{k}\sum_{j=1}^{k} \mathbf{R}\nabla^2 f_{B_j}(w_{\mathbf{R}'}) \cdot \mathbf{R}^\top \mathbf{e_i},$$

where $\mathbf{e_i}$ denotes the i^{th} canonical basis vector. The vector $\mathbf{r_i}$ represents the average of the i-th row of the stochastic Hessian over k minibatches. As k increases, $\mathbf{r_i}$ converges to the true Hessian row. We set $k = 5000$ in the experiments, and use efficient Hessian-vector products (Dagréou et al., 2024).

We partition the indices of the Hessian row $\mathbf{r_i}$ corresponding to weight w_i into three disjoint subsets:

$$\mathbf{r_i} = \mathbf{r}_{\mathbf{i}[I_N]} + \mathbf{r}_{\mathbf{i}[I_L]} + \mathbf{r}_{\mathbf{i}[I_L]}, \quad \text{where}$$

- I_N are indices of weights leading to the same output neuron as w_i,
- I_L are indices of other weights from the same layer,
- I_L are indices of weights of other layers,

and $\mathbf{r}_{\mathbf{i}[I_N]} = r_i$ (resp. I_L and I_L) in the indices in I_N (resp. I_L and I_L) and zero elsewhere.

δw direction	Random	Update
$\mathbf{r}_{\mathbf{i}[I_N]} \cdot \delta w$ (Neuron)	2.86e-05	-4.60e-10
$\mathbf{r}_{\mathbf{i}[I_L]} \cdot \delta w$ (Layer)	-8.71e-06	1.30e-08
$\mathbf{r}_{\mathbf{i}[I_L]} \cdot \delta w$ (Other)	**1.48e-04**	**2.02e-07**

Table 4.1: Contribution $\mathbf{r}_{\mathbf{i}[I]} \cdot \delta w_{[I]}$ of Hessian values in block I to the variation of the i-th gradient component in direction δw. Averaged over multiple δw, **off-diagonal blocks contribute significantly** in both random and update directions.

Figure 4.9 presents the distribution of absolute values for each subset. Our findings show that entries in I_N and, to a lesser extent, I_L, are significantly larger than those from I_L, supporting an approximate block-diagonal Hessian structure.

Given this approximately block-diagonal Hessian structure, previous work (Zhang et al., 2024a) proposes a strict block-diagonal approximation, assuming that the off-diagonal elements are negligible. We further investigate whether this simplification can accurately reflects the local geometry by assessing the practical implications of the block diagonal Hessian structure. We evaluate how each block contributes to gradient variations for a given small direction δw via:

$$[\nabla f(w + \delta w) - \nabla f(w)]_i \approx e_i^T \nabla^2 f(w) \delta w = \mathbf{r}_{\mathbf{i}[I_N]} \cdot \delta w + \mathbf{r}_{\mathbf{i}[I_L]} \cdot \delta w + \mathbf{r}_{\mathbf{i}[I_L]} \cdot \delta w.$$

Table 4.1 quantifies these contributions in a random direction or update direction. Surprisingly, our results reveal that Hessian values outside the block are the primary drivers of gradient evolution, despite their smaller magnitude. This finding challenges the strict block-diagonal Hessian assumption in theoretical analyses. While the diagonal blocks contain larger values, their limited size compared to the full parameter space means that off-diagonal elements collectively play a crucial role in shaping the loss landscape's geometry. Neglecting off-diagonal

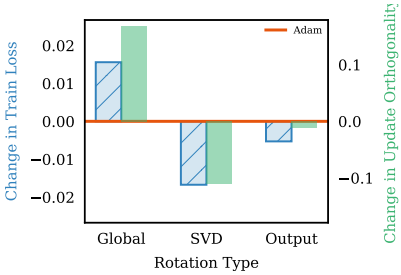


Figure 4.10: Loss and update orthogonality

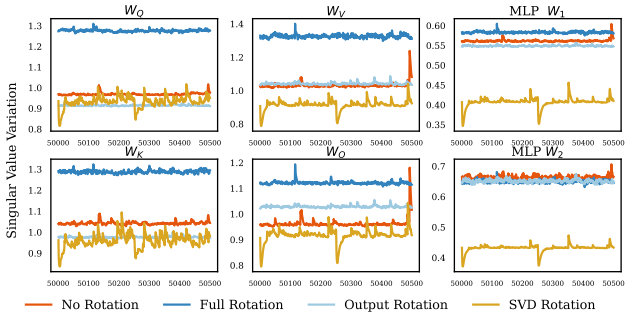


Figure 4.11: Update orthogonality during training

Figure 4.12: (Left) Each panel shows the difference of loss and average CV of singular values relative to the non-rotated baseline. **The orthogonality of layer updates closely aligns with performance under rotation.** (Right) CV of singular values of layer updates over 500 steps, averaged over depth. SVD rotation consistently yields lower CV and more orthogonal layer updates, whereas full rotation shows the opposite. Downward spikes in the SVD Rotation occur when the rotations are recomputed.

elements is an oversimplification, making the approximation inadequate and potentially misleading downstream results.

4.4.2 Orthogonality of layer updates up to scalar factor

In Section 4.3.2 we find that SVD-based rotations improve Adam's performance. We also discuss in Appendix C.5 connections with the recently proposed Muon optimizer (Jordan et al., 2024), which achieves strong performance by performing updates in the orthogonalized first moment direction.

Scaled (semi-)orthogonality. Since orthogonality is defined only for square matrices, we adopt a relaxed notion for rectangular matrices. Specifically, we say that a rectangular matrix is orthogonal if all its singular values are either 1 or 0 (this notion is commonly referred to as *semi-orthogonality* (Abadir and Magnus, 2005)). Moreover, we say that a matrix is a scaled orthogonal matrix if its eigenvalue are either α or 0, where α is the scaling parameter. To measure the scaled orthogonality of a matrix A, we will use the coefficient of variation of its singular values s_i,

$$\mathrm{CV}(s) = \frac{\sigma_s}{\mu_s} = \min_\alpha \frac{1}{\mu_s}\sqrt{\frac{1}{n}\sum_i (s_i - \alpha)^2},$$

where μ_s and σ_s are the mean and standard deviation of the s_i's, respectively. We discuss in Appendix C.3.4 other measures of scaled orthogonality of a matrix

Scaled orthogonality of the layer update. We resume training from a checkpoint for each rotation type for the next 500 steps to measure the orthogonality of the update

$W_{t+1}^{(l)} - W_t^{(l)}$, where $W_t^{(l)}$ represents the weights of layer l at time t. For simplicity, we omit the layer index (l). To separate the effect of step size and weight decay, we measure the update as $A = R^T M_t^{(R)} / (\sqrt{V_t^{(R)}} + \epsilon)$.

Observations. Overall, we find that SVD consistently yields lower CV and more orthogonal updates under the coefficient of variation measure, and output-wise rotation behaves similarly to the no-rotation baseline. Full rotation consistently results in the least orthogonal updates. This ranking aligns clearly with the observed performance of Adam under rotations (Figure 4.10), making it a promising quantity to understand Adam's behavior. In Figure 4.11, we show the CV across time for each layer type, averaged over the depth. See Appendix C.3.4 for full results per layer and over time. Notably, CV drops right after the SVD rotation is recomputed at every 250 steps. This offers insight into the frequency tradeoff of SVD rotations, where more frequent updates improve performance but introduce computational overhead. Overall, our analysis suggests that update orthogonality strongly correlates with optimizer performance, supporting the approaches of Muon (Jordan et al., 2024) and SOAP (Vyas et al., 2025) and opening new avenues for rotation aware theoretical frameworks.

4.5 Related Work

Optimization under rotations. Shampoo (Gupta et al., 2018) first demonstrated the benefits of optimizing under rotations by running AdaGrad under regularly updated rotations, identifying the singular values of the gradient matrices. More recently, SOAP (Vyas et al., 2025) improved Shampoo by applying it to Adam and appropriately updating the moments at each change of basis. Muon (Jordan et al., 2024) concurrently explores a similar approach, but removing the need to explicitly store rotation matrices and instead orthogonalizes gradient matrices with a fast matrix iteration. While these methods show promising empirical improvements via heuristics, our work highlights the importance of developing better theoretical tools to understand their success.

On the theoretical side, we consider (Xie et al., 2025) to be the closest related study, showing that Adam converges more slowly with a randomly rotated loss landscape. They provide convergence analysis based on L_∞ geometry, demonstrating that this yields a better empirical smoothness constant for GPT-2 models. While their work offers valuable theoretical insights, our study takes a more experimental stance. We aim to paint a comprehensive picture of Adam's behavior under a spectrum of rotations, from random to structured transformations, and evaluate how existing rotation-invariant assumptions correlate with Adam's performance. Notably, Balles et al. (2020) also provides relevant insights through the lens of sign gradient descent.

Understanding Adam. Our work casts light on the critical interactions between Adam and the coordinate system, contributing to a growing body of research on Adam's behavior and convergence. Recent works have attributed Adam's success to the heterogeneous block-

diagonal structure of its Hessian (Zhang et al., 2024a), though we find this assumption to be unrealistic. Others have improved convergence guarantees: Défossez et al. (2022) and Guo et al. (2022) offered simplified and novel derivations, Zhang et al. (2022) argued that vanilla Adam converges without modification, Zhou et al. (2024) provided a general convergence analysis for adaptive methods in non-convex settings, and Li et al. (2023b) proposed a convergence proof for Adam without relying on globally bounded gradients. Li et al. (2023a) developed a convergence analysis based on generalized smoothness conditions, and Hübler et al. (2023) proposed parameter-agnostic convergence results under these relaxed conditions. Finally, lower bounds for non-convex optimization were established by Arjevani et al. (2019), with Wang et al. (2023a) addressing the gap between upper and lower bounds for Adam's iteration complexity.

Adam's advantages over SGD. Prior works have attempted to justify Adam's advantages over SGD. Zhang et al. (2020b) and Zhou et al. (2024) suggest SGD suffers more from heavy-tailed noise, with Adam converging faster when gradients are sparse. However, Kunstner et al. (2023) found that noise reduction through larger batch sizes benefits Adam but not SGD. Additionally, Chapter 2 ties Adam's advantage in language models to ill-conditioning caused by heavy-tailed class imbalance, and Pan and Li (2023) to directional sharpness.

4.6 Conclusion

In this work, we have conducted a comprehensive investigation into Adam's sensitivity to rotations of the parameter space, revealing key insights into its optimization dynamics. We demonstrated that some rotations possess advantageous properties, opening new avenues for algorithmic contributions to adaptive algorithms. Our study demonstrates that Adam's performance is intricately tied to the choice of basis, a relationship that existing theoretical frameworks struggle to capture adequately. This investigation highlights the limitations of current rotation-invariant assumptions in explaining Adam's behavior, and identifies update orthogonality as a promising theoretical tool. As the field evolves, we hope these findings will spark new avenues of research, potentially leading to more robust optimization algorithms and deepening our understanding of the fundamental principles underlying successful deep learning optimization.

Chapter 5

Open Problems

This thesis has discussed three separate studies into what the Adam optimizer is doing to be so effective. In Chapter 2, we investigated how the distribution of the class labels in text data are creating a uniquely difficult problem for gradient descent, while Adam remains unaffected and even leverages this property. Following this work, Chapter 3 examines a similar phenomenon occurring in the distribution of the features in some datasets, where ill conditioning again occurs causing gradient descent to struggle. Finally, Chapter 4 takes a different approach, studying how Adam's performance is effected by the basis the optimization problem is being solved in, finding the choice of basis can both degrade and improve performance. Despite the insights brought by these works, there remain many questions as to why Adam and its surrounding optimization pipeline are so effective. In this chapter we conclude with discussion of future questions along these lines.

5.1 Interactions with Learning Rate

Despite Adam's ability to "choose" an adaptive step size for each coordinate, it is often crucial to tune the learning rate α in practice. Large step sizes can cause optimization to diverge while small ones make minimal progress. A robust method of choosing the step size is yet to be developed, despite extensive work on the subject. In practice having a constant learning rate is often not enough either. Practitioners have observed that a very small learning rate is necessary at initialization, but can be increased early on in training, a practice known as learning rate warmup (Kalra and Barkeshli, 2024). Conversely, as training progresses it is common practice to decrease the learning rate in some fashion. This is motivated both by theory (where SGD is known to converge with a decreasing step size) and empirical observations (where it is often the case local curvature increases throughout optimization).

Several schemes exist, but the most common is to decay the learning rate using the half period of a cosine wave (Loshchilov and Hutter, 2017). While these practices are common regardless of the optimization algorithm being used, it is likely the case that Adam has unique interactions with the learning rate and scheduling strategies. Liu et al. (2020a) argue Adam's dependence on moving averages motivates the need for warmup, hypothesizing the momentum buffers will have higher variance near initialization and therefore require a smaller step size to not diverge. Methods designed specifically for Adam(W) have also shown success in practice, such as "Schedule-Free" AdamW (Defazio et al., 2024). Characterizing interactions between Adam (and related algorithms) with the step size remains an interesting direction with several unanswered questions.

5.2 Batch Size and Stochasticity

Early theories of Adam's superior performance proposed that Adam was more resistant to "heavy-tailed noise" in the gradients, but followup work refuted this claim. Kunstner et al. (2023) showed that when the batch size is increased from standard batch sizes up until full batch deterministic gradient descent, the gap between Adam and SGD actually grows. However, recently a new connection between Adam and stochasticity has been proposed. Both Marek et al. (2025) and Sreckovic et al. (2025) have found that if the batch size is decreased, both Adam and SGD appear to perform better and have relatively similar performance. This is in contrast to the typical view of stochastic optimization in general, where a larger batch size limits the variance in the stochastic gradient estimates, which should lead to faster optimization. While these works present substantial experimental evidence to support their claims, neither provide a comprehensive explanation for why a smaller batch size is actually helping.

While we leave it for future work, we conjecture that the underlying mechanism is very related to the heavy-tailed class imbalance discussed in Chapter 2. Our claims regarding heavy-tailed class imbalance are based on imbalance creating a poorly conditioned optimization problem

where gradient descent will struggle to make progress. This conditioning problem does not rely on stochasticity, we experimentally verify this property in both the deterministic and stochastic setting. However, the distribution of classes is a global property of the dataset. In a randomly sampled small batch, it could be the case that most examples in the batch are of different classes, in which case the imbalance will cause less ill conditioning. Similarly, if a batch is dominated by a majority class, gradient descent will minimize the loss with respect to that class and ignore the minority class. As the batch size increases, its properties will more closely look like the full dataset, or in other words the heavy-tailed class imbalance will become more pronounced in an individual batch. In the previous two cases, the global ill conditioned nature of the dataset will be less prominent at a given step. We leave further investigation into this hypothesis for future work.

5.3 Generalization

This thesis has primarily focused on the characteristics of Adam while minimizing the training loss, but this is not the primary focus on many machine learning applications. Generalization, the ability to have accurate predictions on new, unseen, data is impacted by the optimization procedure, and it is known that SGD and Adam reach different minimizers. Properties of the solutions SGD finds have been extensively studied, such as Wu et al., 2022 finding that SGD will tend to reach "flat" minimizers. Properties like this are thought to be favourable for generalization, and the study of the set of minimizers Adam finds is less studied. In practice, it is often the case that Adam will suffer slightly in generalization when compared to SGD (Zhou et al., 2020; Keskar and Socher, 2017). Recent work such as Fan et al. (2025) explore these questions on simple models, further study is required to fully understand the generalization properties of Adam at scale.

5.4 Recent Optimizers

Despite Adam's dominance in practice, attempts at creating superior domain specific or general optimization algorithms are very frequent. Schmidt et al., 2021 gives a non-exhaustive list of nearly 100 of such attempts, a list which has expanded significantly since its publication. Throughout this thesis, we have used Adam and AdamW interchangeably, but the underlying mechanism that lead to AdamW being more popular is itself not fully understood. Several works have made other minor modifications to Adam, but they are seldom adopted in practice. Examples include adding classical optimization theory inspired variance reduction techniques (Yuan et al., 2024) and adding a third moving average for the second moment (Pagliardini et al., 2025). While these approaches occasionally offer better performance in some cases, they often add additional complexity or more hyperparameters, hindering adoption.

A more recent development in optimization for deep learning has been "matrix valued" optimization algorithms. Classically, optimization theory typically assumes the objects in

question are vectors and ignores the potential structure in matrix valued parameters and gradients. However, there have been some attempts to take advantage of matrix structure (Carlson et al., 2015). Gupta et al. (2018) proposed considering optimization over tensors, along with a dense preconditioning scheme they termed Shampoo. While their method saw some adoption, renewed interest came around the introduction of the Muon optimizer, which introduced a computationally efficient method of "orthogonalizing" gradient matrices. Followup work has shown that under simplification, Muon and Shampoo share the same update. As discussed in Chapter 4, recent work has seen several approaches that see performance gains by treating the gradient matrices as matrices, and using the (potentially approximately) closest (semi-)orthogonal matrix for their update instead. We discuss in Appendix C.5 how this update is connected to Adam in a rotated parameter space. Recent work has also attempted to combine the success of these new algorithms with Adam. Vyas et al. (2025) propose to use eigenvectors of Shampoo's preconditioner to precondition the gradient before applying the Adam update to the parameters. Understanding the convergence behaviors and general characteristics of this new class optimizers remains largely an open question, although some attempts at theoretical analysis has been made (Shen et al., 2025). Further study will be required to address these questions.

Bibliography

K. M. Abadir and J. R. Magnus (2005). *Matrix Algebra*. 1st. See Section 4.23 on Semi-orthogonality. Cambridge: Cambridge University Press (cited on page 44).

Kwangjun Ahn, Xiang Cheng, Minhak Song, Chulhee Yun, Ali Jadbabaie, and Suvrit Sra (2023). "Linear attention is (maybe) all you need (to understand transformer optimization)". *arXiv preprint* arXiv:2310.01082. arXiv: 2310.01082 (cited on page 24).

Rangachari Anand, Kishan G. Mehrotra, Chilukuri K. Mohan, and Sanjay Ranka (1993). "An improved algorithm for neural network classification of imbalanced training sets". *IEEE Transactions on Neural Networks* 4.6, pages 962–969 (cited on pages 13, 21).

Yossi Arjevani, Yair Carmon, John C. Duchi, Dylan J. Foster, Nathan Srebro, and Blake E. Woodworth (2019). "Lower bounds for non-convex stochastic optimization". *Mathematical Programming* 199, pages 165–214 (cited on page 46).

Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). "Layer Normalization". *Neural Information Processing Systems*. arXiv: 1607.06450 [stat.ML] (cited on pages 29, 32, 64, 70).

Lukas Balles and Philipp Hennig (2018). "Dissecting Adam: The Sign, Magnitude and Variance of Stochastic Gradients". *International Conference on Machine Learning*. Volume 80, pages 413–422 (cited on page 16).

Lukas Balles, Fabian Pedregosa, and Nicolas Le Roux (2020). *The Geometry of Sign Gradient Descent*. arXiv: 2002.08056 (cited on pages 41, 45).

Lucas Beyer, Xiaohua Zhai, and Alexander Kolesnikov (2022). "Better plain ViT baselines for ImageNet-1k". *CoRR* abs/2205.01580. arXiv: 2205.01580 (cited on pages 65, 104).

Alberto Bietti, Vivien Cabannes, Diane Bouchacourt, Herve Jegou, and Leon Bottou (2023). "Birth of a Transformer: A Memory Viewpoint". *Neural Information Processing Systems* (cited on page 24).

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei (2020). "Language Models are Few-Shot Learners". *Neural Information Processing Systems* (cited on pages 11, 34).

Vivien Cabannes, Berfin Simsek, and Alberto Bietti (2024). "Learning associative memories with gradient descent". *International Conference on Machine Learning* (cited on page 91).

Tianle Cai, Shengjie Luo, Keyulu Xu, Di He, Tie-Yan Liu, and Liwei Wang (2021). "GraphNorm: A Principled Approach to Accelerating Graph Neural Network Training". *International Conference on Machine Learning* (cited on page 29).

David E. Carlson, Edo Collins, Ya-Ping Hsieh, Lawrence Carin, and Volkan Cevher (2015). "Preconditioned Spectral Descent for Deep Learning". *Neural Information Processing Systems* (cited on page 50).

Xiangyi Chen, Sijia Liu, Ruoyu Sun, and Mingyi Hong (2019). "On the Convergence of A Class of Adam-Type Algorithms for Non-Convex Optimization". *International Conference on Learning Representations* (cited on page 34).

Michael Crawshaw, Mingrui Liu, Francesco Orabona, Wei Zhang, and Zhenxun Zhuang (2022). "Robustness to Unbounded Smoothness of Generalized SignSGD". *Neural Information Processing Systems* (cited on pages 11, 126).

Mathieu Dagréou, Pierre Ablin, Samuel Vaiter, and Thomas Moreau (2024). "How to compute Hessian-vector products?" *ICLR Blogposts* (cited on page 43).

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré (2022). "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness". *Neural Information Processing Systems* (cited on page 37).

Aaron Defazio, Xingyu Yang, Ahmed Khaled, Konstantin Mishchenko, Harsh Mehta, and Ashok Cutkosky (2024). "The Road Less Scheduled". *Neural Information Processing Systems* (cited on page 48).

Alexandre Défossez and Francis R. Bach (2017). *AdaBatch: Efficient Gradient Aggregation Rules for Sequential and Parallel Stochastic Gradient Methods* (cited on page 24).

Alexandre Défossez, Leon Bottou, Francis Bach, and Nicolas Usunier (2022). "A Simple Convergence Proof of Adam and Adagrad". *Transactions on Machine Learning Research* (cited on pages 34, 46, 126).

Chenhui Deng, Zichao Yue, and Zhiru Zhang (2024). "Polynormer: Polynomial-Expressive Graph Transformer in Linear Time". *arXiv preprint arXiv:2403.01232* (cited on page 32).

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei (2009). "ImageNet: A large-scale hierarchical image database". *Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society (cited on pages 7, 38, 63, 64, 104, 105).

Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, et al. (2021). "Eta prediction with graph neural networks in google maps". *Proceedings of the 30th ACM international conference on information & knowledge management*, pages 3767–3776 (cited on page 32).

Haiwei Dong and Shuang Xie (2024). "Large Language Models (LLMs): Deployment, Tokenomics and Sustainability". *arXiv:2405.17147* (cited on page 34).

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby (2021). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". *International Conference on Learning Representations* (cited on page 38).

John C. Duchi, Elad Hazan, and Yoram Singer (2011). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". *Journal of Machine Learning Research* 12, pages 2121–2159 (cited on pages 5, 18, 24, 32, 34).

Vijay Prakash Dwivedi, Ladislav Rampášek, Michael Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini (2022). "Long range graph benchmark". *Neural Information Processing Systems* 35, pages 22326–22340 (cited on page 29).

Chen Fan, Mark Schmidt, and Christos Thrampoulidis (2025). "Implicit Bias of Spectral Descent and Muon on Multiclass Separable Data". *Neural Information Processing Systems* (cited on page 49).

Vitaly Feldman (2020). "Does learning require memorization? a short tale about a long tail". *Symposium on Theory of Computing*, pages 954–959 (cited on pages 23, 73).

Matthias Fey and Jan E. Lenssen (2019). "Fast Graph Representation Learning with PyTorch Geometric". *ICLR Workshop on Representation Learning on Graphs and Manifolds* (cited on pages 29, 97).

Emanuele Francazi, Marco Baity-Jesi, and Aurélien Lucchi (2023). "A Theoretical Analysis of the Learning Dynamics under Class Imbalance". *International Conference on Machine Learning*. Volume 202. PMLR. PMLR, pages 10285–10322 (cited on pages 13, 21).

Philip Gage (1994). "A new algorithm for data compression". *C Users Journal* 12.2, pages 23–38 (cited on page 67).

Nikhil Ghosh, Song Mei, and Bin Yu (2022). "The Three Stages of Learning Dynamics in High-dimensional Kernel Methods". *International Conference on Learning Representations* (cited on page 23).

Xavier Glorot and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". *International Conference on Artificial Intelligence and Statistics* (cited on page 29).

Aaron Gokaslan and Vanya Cohen (2019). *OpenWebText Corpus* (cited on pages 38, 104).

Alicia Golden, Samuel Hsia, Fei Sun, Bilge Acun, Basil Hosmer, Yejin Lee, Zachary DeVito, Jeff Johnson, Gu-Yeon Wei, David Brooks, and Carole-Jean Wu (2024). "Is Flash Attention Stable?" *arXiv:2405.02803* (cited on page 37).

Baptiste Goujaud, Adrien Taylor, and Aymeric Dieuleveut (2022). "Optimal first-order methods for convex functions with a quadratic upper bound". *arXiv:2205.15033* (cited on page 126).

Thamme Gowda and Jonathan May (2020). "Finding the Optimal Vocabulary Size for Neural Machine Translation". *Findings of the Association for Computational Linguistics*. Findings of ACL. Association for Computational Linguistics, pages 3955–3964 (cited on page 24).

Daniele Grattarola and Cesare Alippi (2021). "Graph Neural Networks in TensorFlow and Keras with Spektral [Application Notes]". *IEEE Comput. Intell. Mag.* (cited on page 29).

Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov (2018). "Learning word vectors for 157 languages". *arXiv preprint arXiv:1802.06893* (cited on page 95).

Charles Guille-Escuret, Baptiste Goujaud, Manuela Girotti, and Ioannis Mitliagkas (2020). "A Study of Condition Numbers for First-Order Optimization". *International Conference on Artificial Intelligence and Statistics* (cited on page 126).

Charles Guille-Escuret, Adam Ibrahim, Baptiste Goujaud, and Ioannis Mitliagkas (2022). "Gradient Descent Is Optimal Under Lower Restricted Secant Inequality And Upper Error Bound". *Neural Information Processing Systems* (cited on page 126).

Charles Guille-Escuret, Hiroki Naganuma, Kilian Fatras, and Ioannis Mitliagkas (2024). "No Wrong Turns: The Simple Geometry Of Neural Networks Optimization Paths". *International Conference on Machine Learning* (cited on page 126).

Zhishuai Guo, Yi Xu, Wotao Yin, Rong Jin, and Tianbao Yang (2022). "A Novel Convergence Analysis for Algorithms of the Adam Family and Beyond". *arXiv:2104.14840* (cited on pages 46, 126).

Vineet Gupta, Tomer Koren, and Yoram Singer (2018). "Shampoo: Preconditioned Stochastic Tensor Optimization". *International Conference on Learning Representations* (cited on pages 34, 45, 50, 125).

Will Hamilton, Zhitao Ying, and Jure Leskovec (2017). "Inductive representation learning on large graphs". *Neural Information Processing Systems* 30 (cited on page 97).

Bobby He, James Martens, Guodong Zhang, Aleksandar Botev, Andrew Brock, Samuel L. Smith, and Yee Whye Teh (2023). "Deep Transformers without Shortcuts: Modifying Self-attention for Faithful Signal Propagation". *International Conference on Learning Representations*. OpenReview.net (cited on page 24).

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep Residual Learning for Image Recognition". *Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, pages 770–778 (cited on pages 39, 64, 65, 105).

Abdolhossein Hoorfar and Mehdi Hassani (2008). "Inequalities on the Lambert function and hyperpower function". *Journal of Inequalities in Pure and Applied Mathematics* 9.2 (cited on page 93).

Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec (2021). "OGB-LSC: A Large-Scale Challenge for Machine Learning on Graphs". *CoRR* abs/2103.09430. arXiv: 2103.09430 (cited on page 32).

Florian Hübler, Junchi YANG, Xiang Li, and Niao He (2023). "Parameter-Agnostic Optimization under Relaxed Smoothness". *NeurIPS Workshop on Optimization for Machine Learning* (cited on page 46).

Sergey Ioffe and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". *International Conference on Machine Learning*. Volume 37. JMLR.org, pages 448–456 (cited on pages 29, 32, 70).

Mohammad Rasool Izadi, Yihao Fang, Robert Stevenson, and Lizhen Lin (2020). "Optimization of Graph Neural Networks with Natural Gradient Descent". *IEEE BigData*. IEEE (cited on page 27).

Kaiqi Jiang, Dhruv Malik, and Yuanzhi Li (2023). "How Does Adaptive Optimization Impact Local Neural Network Geometry?" *Neural Information Processing Systems* (cited on pages 11, 32).

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein (2024). *Muon: An optimizer for hidden layers in neural networks* (cited on pages 9, 34, 36, 44, 45, 124).

Dayal Singh Kalra and Maissam Barkeshli (2024). "Why Warmup the Learning Rate? Underlying Mechanisms and Improvements". *Neural Information Processing Systems* (cited on page 48).

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei (2020). "Scaling laws for neural language models". *arXiv:2001.08361* (cited on page 34).

Hamed Karimi, Julie Nutini, and Mark Schmidt (2016). "Linear Convergence of Gradient and Proximal-Gradient Methods Under the Polyak-Łojasiewicz Condition". *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016* (cited on page 30).

Andrej Karpathy (2022). *NanoGPT* (cited on page 104).

Nitish Shirish Keskar and Richard Socher (2017). "Improving Generalization Performance by Switching from Adam to SGD". *arXiv:1712.07628* (cited on pages 39, 49).

Diederik P. Kingma and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". *International Conference on Learning Representations* (cited on pages 1, 11, 18, 27, 34, 41).

Thomas N Kipf and Max Welling (2016). "Semi-supervised classification with graph convolutional networks". *arXiv preprint arXiv:1609.02907* (cited on pages 28, 29, 32, 96).

Taku Kudo (2018). "Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates". *Annual Meeting of the Association for Computational Linguistics*. ACL, pages 66–75 (cited on page 67).

Frederik Kunstner, Jacques Chen, Jonathan Wilder Lavington, and Mark Schmidt (2023). "Noise is not the main factor behind the gap between SGD and Adam on transformers, but sign descent might be". *International Conference on Learning Representations* (cited on pages 11, 16, 23, 34, 46, 48).

Frederik Kunstner, Philipp Hennig, and Lukas Balles (2019). "Limitations of the empirical Fisher approximation for natural gradient descent". *Neural Information Processing Systems*, pages 4158–4169 (cited on page 18).

Frederik Kunstner, Alan Milligan, Robin Yadav, Mark Schmidt, and Alberto Bietti (2024). "Heavy-Tailed Class Imbalance and Why Adam Outperforms Gradient Descent on Language Models". *Neural Information Processing Systems* (cited on pages iv, 34).

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). "Gradient-Based Learning Applied to Document Recognition". *Proceedings of the IEEE*. Volume 86. 11, pages 2278–2324 (cited on page 63).

Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller (2012). "Efficient BackProp". *Neural Networks: Tricks of the Trade - Second Edition*. Springer (cited on page 27).

Haochuan Li, Jian Qian, Yi Tian, Alexander Rakhlin, and Ali Jadbabaie (2023a). "Convex and Non-convex Optimization Under Generalized Smoothness". *Neural Information Processing Systems* (cited on page 46).

Haochuan Li, Alexander Rakhlin, and Ali Jadbabaie (2023b). "Convergence of Adam Under Relaxed Assumptions". *Neural Information Processing Systems* (cited on pages 34, 46, 126).

Huan Li and Zhouchen Lin (2024). "On the $O(\frac{\sqrt{d}}{T^{1/4}})$ Convergence Rate of RMSProp and Its Momentum Extension Measured by ℓ_1 Norm". *arXiv:2402.00389* (cited on page 126).

Yan Li, Dhruv Choudhary, Xiaohan Wei, Baichuan Yuan, Bhargav Bhushanam, Tuo Zhao, and Guanghui Lan (2022). "Frequency-aware SGD for Efficient Embedding Learning with Provable Benefits". *International Conference on Learning Representations, International Conference on Learning Representations*. OpenReview.net (cited on page 24).

Zhaoqi Li, Yu Ma, Catalina Vajiac, and Yunkai Zhang (2018). "Exploration of Numerical Precision in Deep Neural Networks". *arXiv:1805.01078* (cited on page 37).

Jingyuan Liu and Jianlin et al. Su (2025). *Muon is Scalable for LLM Training*. arXiv: 2502.16982 [cs.LG] (cited on pages xiii, 113, 120).

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han (2020a). "On the Variance of the Adaptive Learning Rate and Beyond". *International Conference on Learning Representations*. OpenReview.net (cited on page 48).

Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han (2020b). "Understanding the Difficulty of Training Transformers". *Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 5747–5763 (cited on pages 11, 24).

Ilya Loshchilov and Frank Hutter (2017). "SGDR: Stochastic Gradient Descent with Warm Restarts". *International Conference on Learning Representations*. OpenReview.net (cited on page 48).

Ilya Loshchilov and Frank Hutter (2019). "Decoupled Weight Decay Regularization". *International Conference on Learning Representations* (cited on pages 6, 34).

Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat (2023). "Estimating the Carbon Footprint of BLOOM, a 176B Parameter Language Model". *Journal of Machine Learning Research* 24.253, pages 1–15 (cited on page 34).

Sasha Luccioni, Victor Schmidt, Alexandre Lacoste, and Thomas Dandres (2019). "Quantifying the carbon emissions of machine learning". *NeurIPS Workshop on Tackling Climate Change with Machine Learning* (cited on page 34).

Yuankai Luo, Lei Shi, and Xiao-Ming Wu (2024). "Classic GNNs are Strong Baselines: Reassessing GNNs for Node Classification". *Neural Information Processing Systems Datasets and Benchmarks Track* (cited on page 29).

Zhi-Quan Luo and Paul Tseng (1993). "Error bounds and convergence analysis of feasible descent methods: a general approach". *Annals of Operations Research* 46.1, pages 157–178 (cited on page 126).

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz (1993). "Building a Large Annotated Corpus of English: The Penn Treebank". *Computational Linguistics* 19.2, pages 313–330 (cited on page 63).

Martin Marek, Sanae Lotfi, Aditya Somasundaram, Andrew Gordon Wilson, and Micah Goldblum (2025). "Small Batch Size Training for Language Models: When Vanilla SGD Works, and Why Gradient Accumulation Is Wasteful". *Neural Information Processing Systems* (cited on page 48).

Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel (2015). "Image-based recommendations on styles and substitutes". *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 43–52 (cited on page 95).

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov (2022). "Locating and editing factual associations in GPT". *Neural Information Processing Systems* (cited on page 24).

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher (2017). "Pointer Sentinel Mixture Models". *International Conference on Learning Representations*. OpenReview.net (cited on page 63).

Francesco Mezzadri (2007). "How to generate random matrices from the classical compact groups". *Notices of the American Mathematical Society* 54, pages 592–604 (cited on page 101).

Eric J. Michaud, Ziming Liu, Uzay Girit, and Max Tegmark (2023). "The quantization model of neural scaling". *Neural Information Processing Systems* (cited on page 24).

Alan Milligan, Frederik Kunstner, Hamed Shirzad, Mark Schmidt, and Danica J. Sutherland (2024). "Normalization Matters for Optimization Performance on Graph Neural Networks". *NeurIPS Workshop on Optimization for Machine Learning* (cited on page iv).

Christopher Morris, Fabrizio Frasca, Nadav Dym, Haggai Maron, İsmail İlkan Ceylan, Ron Levie, Derek Lim, Michael M. Bronstein, Martin Grohe, and Stefanie Jegelka (2024). "Position: Future Directions in the Theory of Graph Machine Learning". *International Conference on Machine Learning* (cited on page 27).

Kevin P. Murphy (2022). *Probabilistic Machine Learning: An introduction*. MIT Press (cited on page 27).

Mor Shpigel Nacson, Jason D. Lee, Suriya Gunasekar, Pedro Henrique Pamplona Savarese, Nathan Srebro, and Daniel Soudry (2019). "Convergence of Gradient Descent on Separable Data". *International Conference on Artificial Intelligence and Statistics*. Volume 89. PMLR. PMLR, pages 3420–3428 (cited on pages 16, 22).

Preetum Nakkiran, Behnam Neyshabur, and Hanie Sedghi (2021). "The deep bootstrap framework: Good online learners are good offline generalizers". *International Conference on Learning Representations* (cited on page 23).

Yurii E. Nesterov (1983). "A method of solving a convex programming problem with convergence rate $O(k^2)$". *Soviet Mathematics Doklady*. Volume 27. English translation of Russian, in Doklady Adademii Nauk SSSR 269, 543–547 (1983)., pages 372–376 (cited on pages 4, 121).

Lorenzo Noci, Sotiris Anagnostidis, Luca Biggio, Antonio Orvieto, Sidak Pal Singh, and Aurélien Lucchi (2022). "Signal Propagation in Transformers: Theoretical Perspectives and the Role of Rank Collapse". *Neural Information Processing Systems* (cited on pages 24, 74).

Antonio Orvieto, Jonas Kohler, Dario Pavllo, Thomas Hofmann, and Aurélien Lucchi (2022). "Vanishing Curvature in Randomly Initialized Deep ReLU Networks". *International Conference on Artificial Intelligence and Statistics*. Volume 151. PMLR. PMLR, pages 7942–7975 (cited on page 24).

Maris Ozols (2009). *How to generate a random unitary matrix*. Technical report (cited on page 101).

Matteo Pagliardini, Pierre Ablin, and David Grangier (2025). "The AdEMAMix Optimizer: Better, Faster, Older". *International Conference on Learning Representations*. OpenReview.net (cited on page 49).

Yan Pan and Yuanzhi Li (2023). *Toward Understanding Why Adam Converges Faster Than SGD for Transformers*. NeurIPS Workshop on Optimization for Machine Learning (cited on pages 11, 34, 46).

Vardan Papyan, XY Han, and David L Donoho (2020). "Prevalence of neural collapse during the terminal phase of deep learning training". *Proceedings of the National Academy of Sciences* 117.40, pages 24652–24663 (cited on page 73).

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". *Neural Information Processing Systems*, pages 8024–8035 (cited on pages 64, 66, 67).

Steven T. Piantadosi (2014). "Zipf's word frequency law in natural language: A critical review and future directions". *Psychonomic bulletin & review* 21, pages 1112–1130 (cited on pages 7, 11).

Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova (2023). "A critical look at the evaluation of GNNs under heterophily: Are we really making progress?" *arXiv preprint arXiv:2302.11640* (cited on page 95).

Boris T. Polyak (1963). "Gradient methods for the minimisation of functionals". *USSR Computational Mathematics and Mathematical Physics* 3.4, pages 864–878 (cited on page 126).

Boris T. Polyak (1964). "Some methods of speeding up the convergence of iteration methods". *USSR Computational Mathematics and Mathematical Physics* 4.5, pages 1–17 (cited on pages 4, 121).

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever (2019). *Language Models are Unsupervised Multitask Learners*. Tech. Report (cited on pages 38, 64).

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini (2022). "Recipe for a general, powerful, scalable graph transformer". *Neural Information Processing Systems* 35, pages 14501–14515 (cited on page 32).

Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar (2018). "On the Convergence of Adam and Beyond". *International Conference on Learning Representations* (cited on pages 41, 126).

Martin A. Riedmiller and Heinrich Braun (1993). "A direct adaptive method for faster backpropagation learning: the RPROP algorithm". *Proceedings of International Conference on Neural Networks , San Francisco, CA, USA, March 28 - April 1, 1993*. IEEE, pages 586–591 (cited on page 5).

Elan Rosenfeld and Andrej Risteski (2023). "Outliers with Opposing Signals Have an Outsized Effect on Neural Network Optimization". *arXiv preprint* arXiv/2311.04163. arXiv: `2311.04163` (cited on page 24).

David Martınez Rubio (2017). "Convergence analysis of an adaptive method of gradient descent". *University of Oxford, Oxford, M. Sc. thesis* (cited on page 34).

Shiori Sagawa, Aditi Raghunathan, Pang Wei Koh, and Percy Liang (2020). "An investigation of why overparameterization exacerbates spurious correlations". *International Conference on Machine Learning* (cited on page 23).

Robin M. Schmidt, Frank Schneider, and Philipp Hennig (2021). "Descending through a Crowded Valley - Benchmarking Deep Learning Optimizers". *International Conference on Machine Learning*. Volume 139. Proceedings of Machine Learning Research. PMLR, pages 9367–9376 (cited on pages 11, 49).

Rico Sennrich, Barry Haddow, and Alexandra Birch (2016). "Neural Machine Translation of Rare Words with Subword Units". *Annual Meeting of the Association for Computational Linguistics*. The Association for Computer Linguistics (cited on pages 63, 67).

Or Sharir, Barak Peleg, and Yoav Shoham (2020). "The cost of training nlp models: A concise overview". *arXiv:2004.08900* (cited on page 34).

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann (2018). "Pitfalls of graph neural network evaluation". *arXiv preprint arXiv:1811.05868* (cited on page 95).

Wei Shen, Ruichuan Huang, Minhui Huang, Cong Shen, and Jiawei Zhang (2025). "On the Convergence Analysis of Muon". *CoRR* abs/2505.23737. arXiv: 2505.23737 (cited on page 50).

Hamed Shirzad, Honghao Lin, Balaji Venkatachalam, Ameya Velingker, David Woodruff, and Danica J Sutherland (2024). "Even Sparser Graph Transformers". *Neural Information Processing Systems*. arXiv: 2411.16278 (cited on page 32).

Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J Sutherland, and Ali Kemal Sinop (2023). "Exphormer: Sparse transformers for graphs". *International Conference on Machine Learning* (cited on page 32).

Teodora Sreckovic, Jonas Geiping, and Antonio Orvieto (2025). "Is your batch size the problem? Revisiting the Adam-SGD gap in language modeling". *CoRR* abs/2506.12543. arXiv: 2506.12543 (cited on page 48).

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). "Dropout: a simple way to prevent neural networks from overfitting". *Journal of Machine Learning Research* 15.1, pages 1929–1958 (cited on page 64).

Andreas Steiner, Alexander Kolesnikov, Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer (2022). "How to train your ViT? Data, Augmentation, and Regularization in Vision Transformers". *Transactions of Machine Learning Research* 2022 (cited on page 14).

Yuxin Sun, Dong Lao, Ganesh Sundaramoorthi, and Anthony Yezzi (2022). "Surprising Instabilities in Training Deep Networks and a Theoretical Analysis". *Neural Information Processing Systems* (cited on page 37).

Lubos Takac and Michal Zabovsky (2012). "Data analysis in public social networks". *International scientific conference and international workshop present day trends of innovations*. Volume 1. 6 (cited on page 32).

Christos Thrampoulidis, Ganesh Ramachandra Kini, Vala Vakilian, and Tina Behnia (2022). "Imbalance Trouble: Revisiting Neural-Collapse Geometry". *Neural Information Processing Systems* (cited on page 73).

Tijmen Tieleman and Geoffrey Hinton (2012). *RMSPROP: Divide the gradient by a running average of its recent magnitude*. Lecture notes `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf` (cited on pages 5, 16).

Jan Tönshoff, Martin Ritzert, Eran Rosenbluth, and Martin Grohe (2024). "Where Did the Gap Go? Reassessing the Long-Range Graph Benchmark". *Trans. Mach. Learn. Res.* 2024 (cited on page 29).

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou (2021). "Training data-efficient image transformers & distillation through attention". *International Conference on Machine Learning*. Volume 139. Proceedings of Machine Learning Research. PMLR, pages 10347–10357 (cited on page 65).

Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky (2016). "Instance Normalization: The Missing Ingredient for Fast Stylization". *CoRR* abs/1607.08022. arXiv: `1607.08022` (cited on page 29).

Gaël Varoquaux, Alexandra Sasha Luccioni, and Meredith Whittaker (2024). "Hype, Sustainability, and the Price of the Bigger-is-Better Paradigm in AI". *arXiv:2409.14160* (cited on page 34).

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin (2017). "Attention is All you Need". *Neural Information Processing Systems*, pages 5998–6008 (cited on pages 34, 64, 65).

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio (2018). "Graph attention networks". *International Conference on Learning Representations* (cited on pages 29, 32, 97).

Nikhil Vyas, Depen Morwani, Rosie Zhao, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham M. Kakade (2025). "SOAP: Improving and Stabilizing Shampoo using Adam for Language Modeling". *International Conference on Learning Representations* (cited on pages 34, 45, 50).

Bohan Wang, Jingwen Fu, Huishuai Zhang, Nanning Zheng, and Wei Chen (2023a). "Closing the gap between the upper bound and lower bound of Adam's iteration complexity". *Neural Information Processing Systems* (cited on page 46).

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt (2023b). "Interpretability in the Wild: a Circuit for Indirect Object Identification in GPT-2 Small". *International Conference on Learning Representations*. OpenReview.net (cited on page 24).

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang (2019). "Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks". *arXiv preprint arXiv:1909.01315* (cited on page 29).

Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan (2018). "Training Deep Neural Networks with 8-bit Floating Point Numbers". *Neural Information Processing Systems* (cited on page 37).

Jake Ryland Williams, Paul R. Lessard, Suma Desu, Eric M. Clark, James P. Bagrow, Christopher M. Danforth, and Peter Sheridan Dodds (2015). "Zipf's law holds for phrases, not words". *Scientific reports* 5.1, page 12209 (cited on page 24).

Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht (2017). "The Marginal Value of Adaptive Gradient Methods in Machine Learning". *Neural Information Processing Systems* (cited on page 39).

Lei Wu, Mingze Wang, and Weijie Su (2022). "The alignment property of SGD noise and how it helps select flat minima: A stability analysis". *Neural Information Processing Systems* (cited on page 49).

Shuo Xie, Mohamad Amin Mohamadi, and Zhiyuan Li (2025). "Adam Exploits ℓ_∞-geometry of Loss Landscape via Coordinate-wise Adaptivity". *International Conference on Learning Representations* (cited on pages 41, 42, 45).

Zhilin Yang, William Cohen, and Ruslan Salakhudinov (2016). "Revisiting semi-supervised learning with graph embeddings". *International Conference on Machine Learning*. PMLR, pages 40–48 (cited on page 95).

Han-Jia Ye, De-Chuan Zhan, and Wei-Lun Chao (2021). "Procrustean Training for Imbalanced Deep Learning". *International Conference on Computer Vision*. IEEE, pages 92–102 (cited on page 21).

Jiaxuan You, Zhitao Ying, and Jure Leskovec (2020). "Design space for graph neural networks". *Neural Information Processing Systems* 33, pages 17009–17021 (cited on pages 27, 32).

Huizhuo Yuan, Yifeng Liu, Shuang Wu, Xun Zhou, and Quanquan Gu (2024). "MARS: Unleashing the Power of Variance Reduction for Training Large Models". *CoRR* abs/2411.10438. arXiv: 2411.10438 (cited on page 49).

Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie (2020a). "Why Gradient Clipping Accelerates Training: A Theoretical Justification for Adaptivity". *International Conference on Learning Representations* (cited on page 11).

Jingzhao Zhang, Sai Praneeth Karimireddy, Andreas Veit, Seungyeon Kim, Sashank J. Reddi, Sanjiv Kumar, and Suvrit Sra (2020b). "Why are Adaptive Methods Good for Attention Models?" *Neural Information Processing Systems*, pages 15383–15393 (cited on pages 11, 34, 46).

Tianyue H. Zhang, Lucas Maes, Alan Milligan Alexia Jolicoeur-Martineau, Ioannis Mitliagkas, Damien Scieur, Simon Lacoste-Julien, and Charles Guille-Escuret (2025). "Understanding Adam Requires Better Rotation Dependent Assumptions". *Neural Information Processing Systems* (cited on page iv).

Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhi-Quan Luo (2024a). "Why Transformers Need Adam: A Hessian Perspective". *Neural Information Processing Systems* (cited on pages 24, 34, 39, 42, 43, 46).

Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Yinyu Ye, Zhi-Quan Luo, and Ruoyu Sun (2024b). *Adam-mini: Use Fewer Learning Rates To Gain More*. Preprint. arXiv/2406.16793 (cited on pages 24, 39).

Yushun Zhang, Congliang Chen, Naichen Shi, Ruoyu Sun, and Zhi-Quan Luo (2022). "Adam Can Converge Without Any Modification On Update Rules". *Neural Information Processing Systems* (cited on pages 46, 126).

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian (2024a). "GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection". *ICLR Workshop on practical ML for limited/low resource settings* (cited on pages 36, 39, 104).

Lingxiao Zhao and Leman Akoglu (2020). "PairNorm: Tackling Oversmoothing in GNNs". *International Conference on Learning Representations*. OpenReview.net (cited on page 29).

Rosie Zhao, Depen Morwani, David Brandfonbrener, Nikhil Vyas, and Sham M. Kakade (2024b). *Deconstructing What Makes a Good Optimizer for Language Models* (cited on page 24).

Dongruo Zhou, Jinghui Chen, Yuan Cao, Ziyan Yang, and Quanquan Gu (2024). "On the Convergence of Adaptive Gradient Methods for Nonconvex Optimization". *Transactions on Machine Learning Research* (cited on pages 34, 46, 126).

Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven Chu-Hong Hoi, and Weinan E (2020). "Towards Theoretically Understanding Why Sgd Generalizes Better Than Adam in Deep Learning". *Neural Information Processing Systems* (cited on page 49).

Fangyu Zou, Li Shen, Zequn Jie, Weizhong Zhang, and Wei Liu (2019). "A Sufficient Condition for Convergences of Adam and RMSProp". *Conference on Computer Vision and Pattern Recognition* (cited on page 126).

Vilém Zouhar, Clara Meister, Juan Luis Gastaldi, Li Du, Mrinmaya Sachan, and Ryan Cotterell (2023). "Tokenization and the Noiseless Channel". *Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, pages 5184–5207 (cited on page 24).

Appendix A

Heavy Tailed Class Imbalance

A.1 Experimental details

This section documents the datasets, models, software, and experimental setup. The code is available at `https://github.com/fkunstner/class-imbalance-sgd-adam`.

A.1.1 Datasets

- **WikiText-103** (Merity et al., 2017), using sequences of 1 024 tokens and the BPE tokenizer (Sennrich et al., 2016), with a vocabulary of size 50 608.
- **WikiText-2** (Merity et al., 2017) is used in Appendix A.2.1 to illustrate that other combinations of datasets and tokenizers lead to heavy-tailed distributions.
- **PTB** (Marcus et al., 1993), using sequences of 35 tokens built from a word-based tokenizer (`basic_english` provided by `torchtext`), for a vocabulary of size 9 920. For deterministic runs, we use the validation set as a reduced training set, labeled **TinyPTB**.
- **MNIST** (LeCun et al., 1998).
- **ImageNet** (Deng et al., 2009).

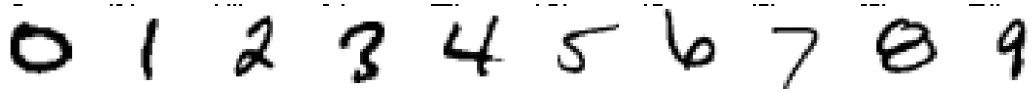
A.1.2 Custom datasets

- **The Random Heavy-Tailed Labels dataset** is a synthetic dataset exhibiting heavy-tailed class imbalance. The number of samples per class and the number of classes are picked to approximate a power-law distribution. We create m "groups" of classes, where each class within a group has the same relative frequency;

  $$\underbrace{1 \text{ class with } 2^m \text{ samples},}_{\text{group } 1} \quad \underbrace{2 \text{ classes with } 2^{m-1} \text{ samples},}_{\text{group } 2} \quad \dots, \quad \underbrace{2^{m-1} \text{ classes with } 2 \text{ samples.}}_{\text{group } m}$$

  The inputs are drawn from a uniform distribution on $[0, 1]$, independently of the class label. The inputs are in $d = (m+1)\,2^m$ dimensions, the number of samples is $n = m\,2^m$ and the number of classes is $c = 2^{m+1} - 1$. We use two variants of the datasets; a large one in Figure 2.4, Appendix A.5 ($m = 11, n = 22\,528, d = 24\,576, c = 4\,095$) and a small one in Appendix A.4 ($m = 8, n = 2\,048, d = 2\,304, c = 511$).

- **The Barcoded MNIST dataset** is a modified variant of MNIST. We start with 50k examples from the original MNIST dataset across 10 classes, and create 51 150 $(5 \times (10 \times 2^{10} - 1))$ new images. The new examples are copies of existing image with an added "barcode", a 10-bit number encoded in a corner of the image, as in the examples below. The class label is a combination of the original class and this barcode.



The **Barcoded-only** dataset contains 10×2^{10} classes with 5 samples each. To obtain an imbalanced dataset, we combine the barcoded images with the original samples from the MNIST dataset to get 101 200 examples spread across 10 250 $(10 \times 2^{10} + 10)$ classes classes; 10 240 with 5 examples per class and 10 classes with $\approx 5k$ examples per class, labeled **MNIST+Barcoded**

- **The Heavy Tailed ImageNet** dataset is a subset of ImageNet (Deng et al., 2009), subsampled to exhibit heavy-tailed class imbalance. We sort the original 1000 classes by frequency and sample $\lceil 1300/k \rceil$ images from the kth class, leading to $n = 10\,217$ samples.
- **The Small ImageNet** dataset is a uniform subset of ImageNet to contrast the with the heavy tailed variant. We sample 10 images per class to get $n = 10\,000$ samples.

A.1.3 Models

- **The 2-layer transformer** used in Appendix A.2.3 is a transformer Vaswani et al. (2017), based on the PyTorch implementation of `TransformerEncoderLayer` (Paszke et al., 2019).

$$\text{Embedding} \to 2\times \text{ [Attention} \to \text{Linear} \to \text{ReLU} \to \text{Linear]} \to \text{Classifier.}$$

The model includes LayerNorm, dropout, and skip connections (He et al., 2016; Ba et al., 2016; Srivastava et al., 2014). The embedding dimension and width of the linear layers is 1000 and the attention modules use 4 heads.

- **The simplified transformer** used in Figure 2.5 and Appendix A.2.3 does not use encoder blocks, and only uses attention:

$$\text{Embedding} \to \text{Attention} \to \text{Classifier.}$$

We remove LayerNorm, dropout, and the block [Linear $\to$ ReLU $\to$ Linear] containing the non-linearity. In Figure 2.5, we freeze the embedding and attention layers at initialization, and only the last classification layer is trained. The model is then a linear model on a fixed feature transformation.

- **The GPT2-Small** model (Radford et al., 2019) is used in Figure 2.1. The blocks includes LayerNorm, residual connections, and dropout on the embedding and dense layers. We

use sinusoid positional encodings as in the transformer architecture (Vaswani et al., 2017). The embedding dimension is 768, the width of the intermediate layers is 3072, and we use 12 encoder blocks with 12 self attention heads.

- **The convolutional network** used in Figure 2.2 and Appendix A.3 is a 2-layer convolution

$$\text{Conv} \to \text{Relu} \to \text{MaxPool} \to \text{Conv} \to \text{Relu} \to \text{MaxPool} \to \text{Linear}$$

- **The linear model** used in Figures 2.4 and 2.7 and Appendix A.5 uses a bias vector.
- **The ResNet18** model (He et al., 2016) is used in Figure 2.3. Additionally, a variant replacing the BatchNorm layers with LayerNorm is used in Appendix A.3.
- **The SimpleViT** model (Beyer et al., 2022) used in Appendix A.3 follows the architecture of a ViT-S/16 (Touvron et al., 2021), based on the `vit-pytorch` implementation (`https://github.com/lucidrains/vit-pytorch` `v1.6.5`).

A.1.4 Training procedures

Our primary focus is on the performance of the optimizers on the training error, using the simplest training procedure possible. We use a constant step-size throughout training, set by grid search. We start with a sparse grid of powers of 10 $[10^{-6}, 10^{-2}, ..., 10^{1}]$ and increase the density to half-powers around the best step-size. The step-size is selected to minimize the maximum over 3 seeds of the training loss at the end of training. For some settings, this selection still produces runs that are unstable; the training loss is the smallest at the end but oscillates a lot during training, reaching loss values that are orders of magnitude worse than at initialization. For those runs, we use the next smaller step-size, which has similar performance at the end but is more stable. We use the following batch sizes with gradient accumulation (computing the gradient through multiple passes)

- The large transformer experiment in Figure 2.1 uses mini-batches of 512 sequences of 1024 tokens.
- The stochastic experiments with a smaller transformer in Appendix A.2.3 uses mini-batches of 512 sequences of 35 tokens.
- Both ResNet18 variants and the Simple Vision Transformer were trained using mini-batches of 1024. The training images were normalized and randomly cropped to 224×224 pixels as is standard for ImageNet training.
- Other experiments use the entire dataset to compute updates

Our experiments ran on a cluster using a mix of A100, P100, V100, and H100 GPUs. The large scale experiment in Figure 2.1 took 3 days on a H100, while all other experiments ran in 2–8 hours. The total amount of compute used for this project is $\approx$3 GPU-years, including preliminary experiments.

A.1.5 Optimization algorithms

Given momentum buffers m_t initialized at $m_0 = 0$ and a (possibly) stochastic gradient $\tilde{g}_t$, we implement the update of GD, normalized GD and sign descent with heavy-ball momentum as

$$\begin{aligned} m_t &= \beta m_{t-1} + d_t, \\ x_{t+1} &= x_t - \alpha m_t, \end{aligned} \qquad \text{with } d_t = \begin{cases} \tilde{g}_t & \text{for gradient descent,} \\ \tilde{g}_t / \|\tilde{g}_t\|_2 & \text{for normalized GD,} \\ \operatorname{sign}(\tilde{g}_t) & \text{for sign descent.} \end{cases}$$

For GD and Adam, we use the standard implementation in PyTorch (Paszke et al., 2019). For all algorithms, we use either momentum with $\beta = 0.9$ ($\beta_1 = 0.9$ for Adam) or no momentum ($\beta = 0$, $\beta_1 = 0$), indicated by solid lines and the legend (+m) for runs with momentum, and dashed lines and the legend (-m) for runs without momentum.

A.1.6 Summary of settings used

Table A.1: Summary of models, datasets and batch-size used

Model	**Dataset**	**Batch size**	**Used in**
GPT2-Small	WT103	512	Figure 2.1 and Figure A.2
2-layer transformer	PTB	512	Figures A.3, A.16 and A.22
1-layer transformer	TinyPTB	Full	Figures A.4 and A.14
1-layer transformer	TinyPTB	Full	Figure 2.5 (last layer only)
CNN	Barcoded MNIST	Full	Figure A.9
CNN	MNIST	Full	Figures 2.2 and A.9
CNN	MNIST+Barcoded	Full	Figures 2.2, A.9, A.15, A.16 and A.20
Linear	Random HT labels, m=11	Full	Figures 2.4, 2.7, A.13, A.16, A.17, A.23 and A.24
Linear	Random HT labels, m=7	Full	Figures A.10 and A.11
Simple ViT	ImageNet	1024	Figure A.7
ResNet18	Small and HT ImageNet	1024	Figures 2.3, A.16 and A.21
ResNet18+LN	Small and HT ImageNet	1024	Figure A.6
Simple ViT	Small and HT ImageNet	1024	Figure A.8

A.2 Language problems

This section provides additional ablations on language models, showing that the impact of class imbalance holds across models of different sizes and using deterministic updates.

A.2.1 shows that the heavy-tailed distribution in text data occurs across datasets and tokenizers.

A.2.2 shows that the imbalanced training speed across frequencies translates to the validation loss.

A.2.3 shows that the imbalance training speed across frequencies and the gap between SGD and Adam can be reproduced with smaller transformers. This effect also appears when training only the last layer, and in the deterministic setting, comparing GD and Adam.

A.2.1 Class distribution for common datasets and tokenizers

Figure A.1 provides additional examples of the heavy-tailed distribution of tokens using the basic english tokenizer in `torchtext` (Paszke et al., 2019), Byte-Pair Encoding (BPE, Sennrich et al., 2016; Gage, 1994) and Unigram (Kudo, 2018) on the PTB and WikiText-2 datasets. The relationship between the relative frequency rank k and and the relative frequency π_k is roughly $\pi_k \propto 1/k$.

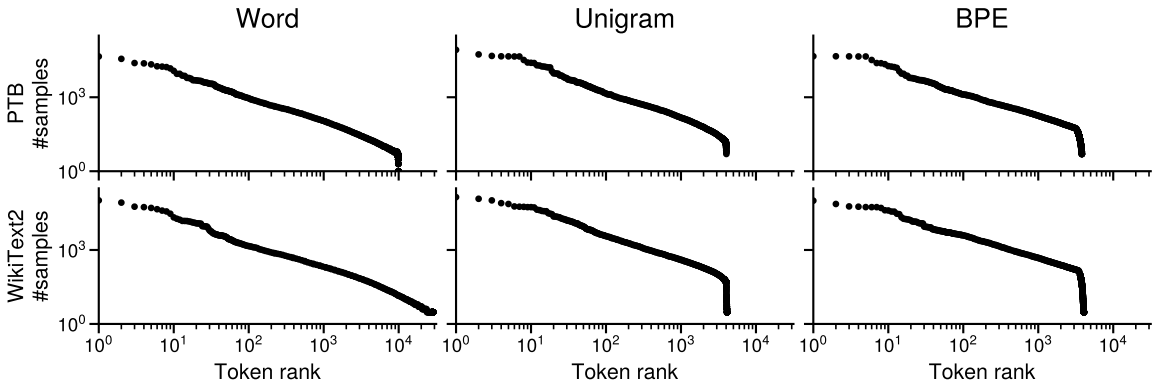


Figure A.1: **Different tokenizers and datasets lead to heavy-tailed token distributions.** Comparison of word and subword tokenization (BPE, Unigram) on the PTB and WikiText2 datasets.

A.2.2 Effect of class imbalance on validation loss

In Figure A.2, we show the validation error on the same problem as Figure 2.1, training GPT2-Small on WikiText-103. The validation loss exhibits the same separation across class frequencies, and the faster progress of Adam on low-frequency classes is also visible. While this trend does not hold for all the settings we investigate, as some settings use smaller

datasets and deterministic training to isolate the source of the training difficulties, the benefit of Adam on low-frequency classes does not immediately lead to overfitting.

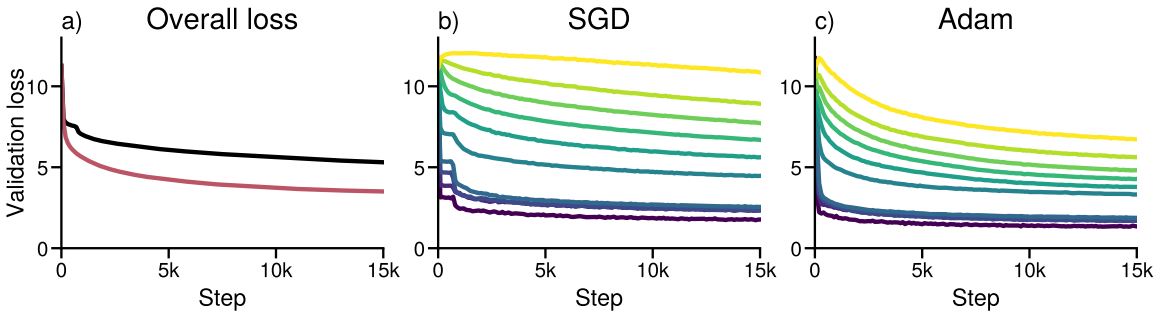


Figure A.2: **The class-separation behavior of Figure 2.1 holds on the validation loss.** Same experiment as Figure 2.1, training GPT2-Small on WikiText-103, but showing the validation loss. (a) Distribution of the classes sorted by class frequency, split into groups corresponding to ≈10% of the data. (b) Overall validation loss. (c, d) Validation loss for each group using SGD and Adam. SGD makes little to no progress on low-frequency classes while Adam makes progress on all groups. (b) is the average of (c, d) for the respective optimizer.

A.2.3 Smaller transformers and deterministic training

In Section 2.2.3, we argued that the qualitatively different behavior on low-frequency classes between SGD and Adam in Figure 2.1 is not due to stochasticity. In this section, we provide additional results showing that this behavior appears across multiple batch sizes on language transformers of different sizes and that it can be reproduced in the deterministic setting.

In Figure A.3, we show that a similar qualitative behavior appears when training a smaller model (2-layer transformer) on a smaller dataset (PTB). In Figure A.4, we repeat the experiment with a 1-layer transformer, trained in full batch on TinyPTB (the validation set of PTB). The separation between GD and Adam on low-frequency classes in the deterministic settings is also visible in Figures 2.2, 2.4, 2.5 and 2.7 in the main paper. These results indicate that stochasticity it is not necessary to reproduce the behavior observed in Figure 2.1. Finally, we repeat the experiment but freeze all the layers except the last, and still observe this behavior in Figure A.5.

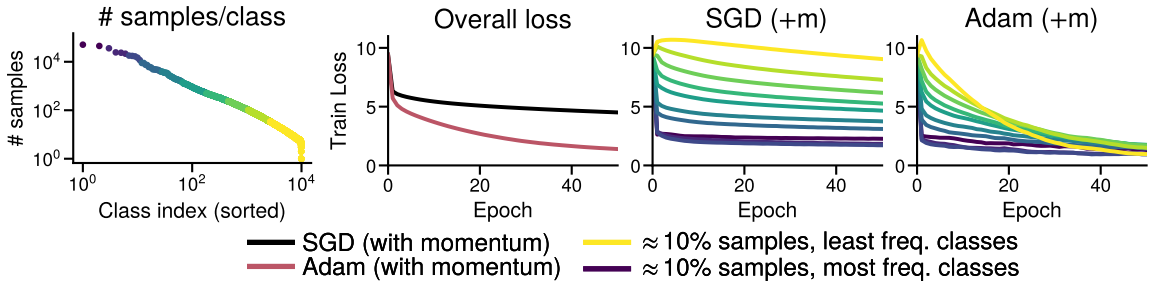


Figure A.3: **Similar behavior as Figure 2.1 on a smaller problem.** Training a 2-layer transformer on PTB with Adam and SGD using larger batch-sizes. As in Figure 2.1, SGD makes little to no progress on low-frequency classes while Adam makes progress on all subsets. Subplots: **(1)** Distribution of the classes and subsets of the data sorted by class frequency, each corresponding to ≈10% of the samples. **(2)** Overall training loss. **(3, 4)** Training loss for each subset for SGD and Adam.

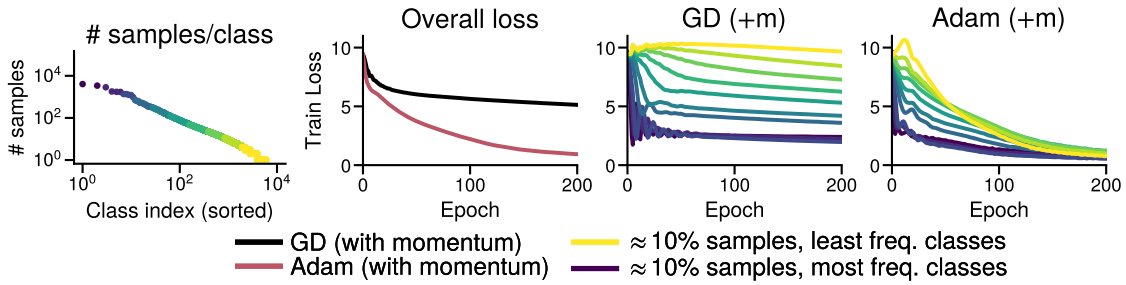


Figure A.4: **Similar behavior as Figure 2.1 on a one-layer transformer with deterministic updates.** Trained on TinyPTB. Subplots: **(1)** Distribution of the classes and subsets of the data sorted by class frequency. **(2)** Overall training loss. **(3, 4)** Training loss for each subset for GD and Adam.

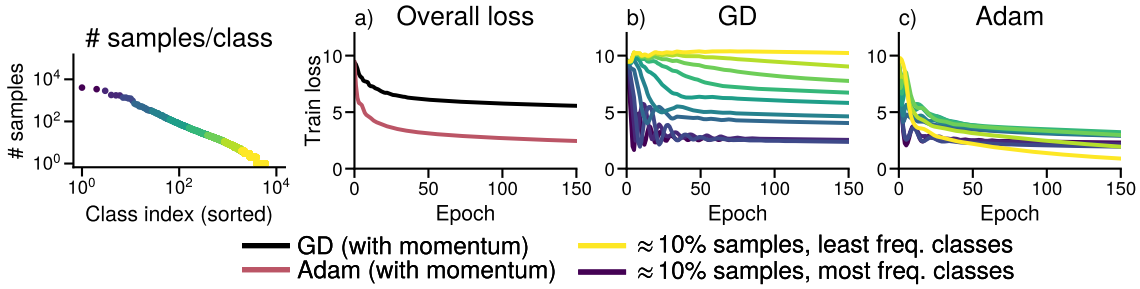


Figure A.5: **Similar behavior as Figure 2.1 when training only the last layer.** Training the last layer of a 1-layer transformer on PTB with Adam and GD with deterministic updates. Subplots: **(1)** Distribution of the classes and subsets of the data sorted by class frequency. **(2)** Overall training loss. **(3, 4)** Training loss for each subset for GD and Adam.

A.3 Vision problems

This section gives additional results on vision tasks to complement Section 2.2.1.

- Figure A.6 shows a similar behavior on a ResNet18 with LayerNorm instead of BatchNorm.
- Figure A.7 shows a similar behavior with a vision transformer.
- Figure A.9 confirms that GD can solve the Barcoded MNIST variant without imbalance.

A.3.1 ResNet18 with LayerNorm

In Figure A.6, we use the same settings Figure 2.3. training a ResNet18 on a uniform and unbalanced subset of ImageNet, but replace the normalization layers with LayerNorm (Ba et al., 2016) instead of BatchNorm (Ioffe and Szegedy, 2015). We observe a similar pattern as in Figure 2.3. Although Adam slightly outperforms SGD on the uniform dataset, the performance gap grows on the imbalanced one.

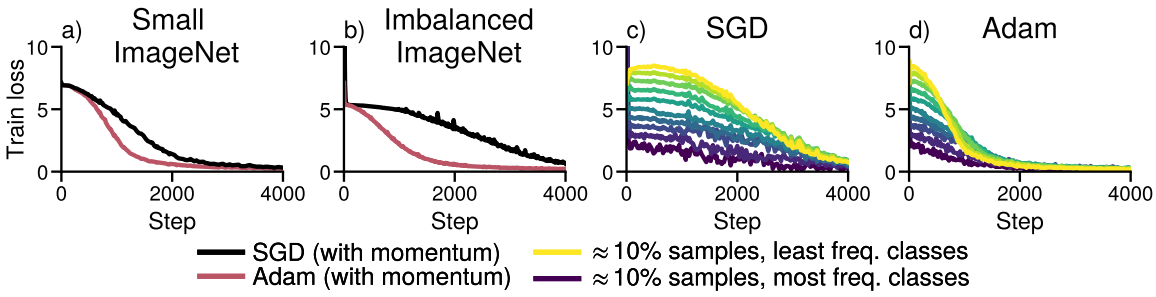


Figure A.6: **Adam outperforms SGD on ResNet with LayerNorm under heavy-tailed imbalance.** (a) Performance on a uniform subset of ImageNet (b) and on an imbalanced subset with class frequencies $\pi_k \propto 1/k$. (c, d) Performance of GD and Adam across frequencies.

A.3.2 Vision Transformers

In Figure A.7, we train a vision transformer on the ImageNet dataset, without subsampling, to confirm that the training behavior is similar. While vision transformers might require more data or regularization than their ResNet counterparts to achieve comparable generalization performance, the optimization problem does not appear to be more difficult for SGD than for Adam.

In Figure A.8, we train the same vision transformer on the uniform and imbalanced subsets of ImageNet. As in prior experiments with vision data, the performance of Adam appears unaffected by the change in class frequencies while the performance of SGD degrades.

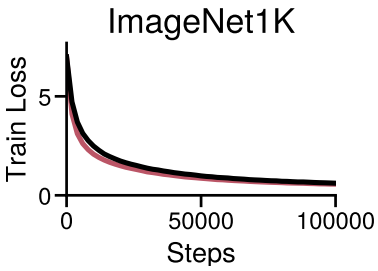


Figure A.7: **Adam and SGD perform similarly training a Vision Transformer with balanced Classes.** Training loss on the full ImageNet dataset (without subsampling). There is little performance in training performance.

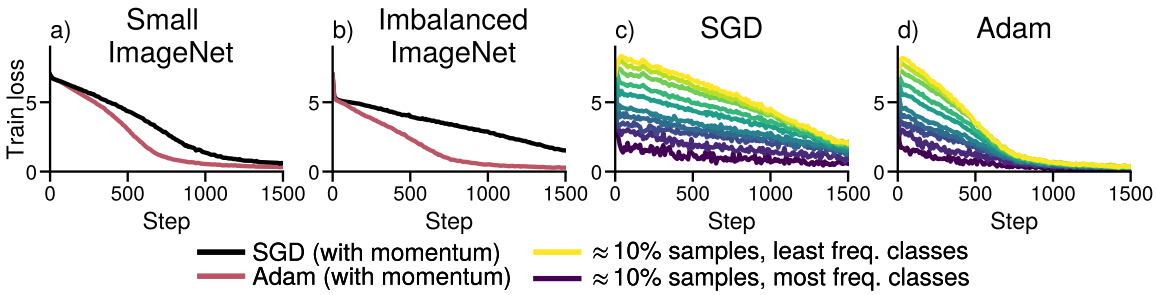


Figure A.8: **Adam outperforms SGD on vision transformer under heavy-tailed imbalance.** (a) Performance on a uniform subset of ImageNet (b) and on an imbalanced subset with class frequencies $\pi_k \propto 1/k$. (c, d) Performance of GD and Adam across frequencies.

A.3.3 Sanity check on Barcoded MNIST

Figure 2.2 in Section 2.2.1 showed that the performance gap between GD and Adam on the imbalanced variant of MNIST with barcoded images is larger than on plain MNIST. In this section, we verify that the training difficulties encountered on the CNN on the imbalanced MNIST dataset of Figure 2.2 are indeed due to class imbalance. As we create new images and new classes by adding a barcode in the corner of existing images, it could be that the dataset becomes harder to fit.

In Figure A.9, we run Adam and GD to train the same network on the MNIST dataset only, the barcoded-only subset of the imbalanced MNIST and the combination of the two, leading to an imbalanced dataset. While Adam is faster GD on the barcoded-only dataset, both algorithms reach negligible error within 200 steps. In contrast, on the combined imbalanced dataset MNIST+Barcoded, GD fails to make progress on the low-frequency classes and stalls.

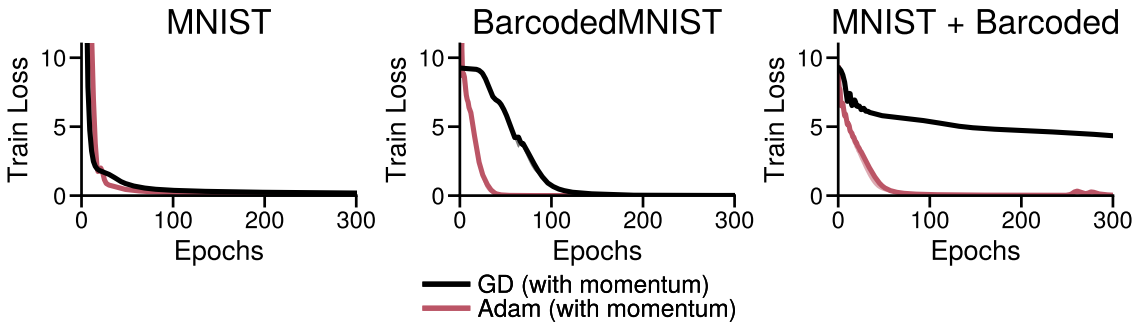


Figure A.9: **GD optimizes on balanced barcoded data.** Training a CNN on only the barcoded portion of the data, which has balanced classes. While Adam is slightly faster, both optimizers reach negligible error within 200 steps. As the level of imbalance is increased, GD performs increasingly worse than Adam.

A.4 Linear models

Section 2.2.2 showed that GD is already slow on linear models. We give additional details here.

A.4.1 discusses the impact of the distribution of the inputs, as it is possible to construct problems exhibiting class imbalance without negatively impacting training.

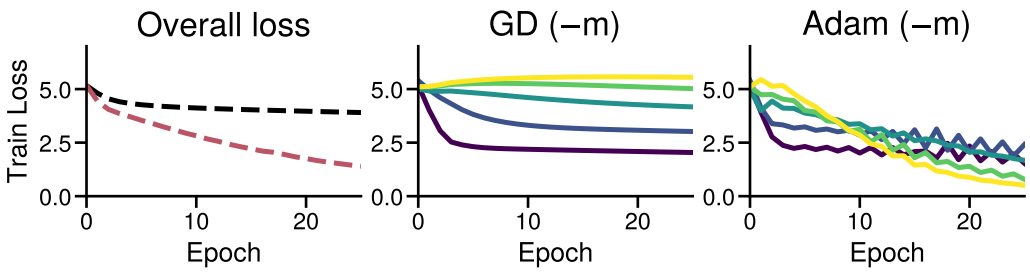
A.4.2 shows that while (S)GD appears stuck in some experiments, it is not due to being stuck in a local minima. It eventually converges, although very slowly, if run long enough.

A.4.3 shows that while some of our datasets are separable, leading to weights going to ∞, class imbalance also impacts optimization when the weights remain small, e.g. when using l2 regularization.

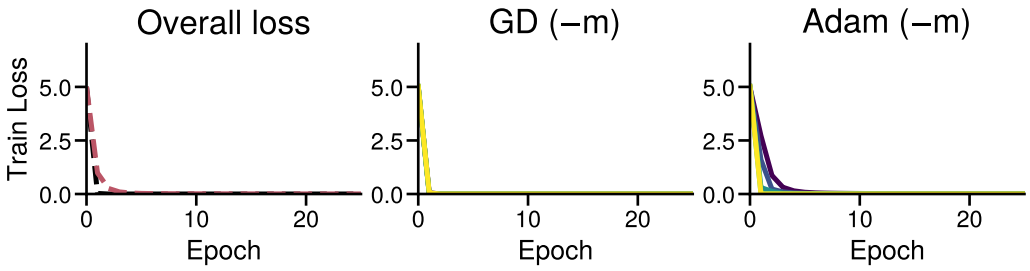
A.4.1 Impact of input distribution

Imbalance alone is not sufficient to induce slow performance of GD on low-frequency classes. It is possible to generate a dataset with heavy-tailed class imbalance where GD fits all classes fast, by making all inputs $\mathbf{x}_i$ (close to) orthogonal, $\langle \mathbf{x}_i, \mathbf{x}_j \rangle \approx 0$ for $i \neq j$. If all samples are orthogonal, $\langle \mathbf{x}_i, \mathbf{x}_j \rangle = 0 \; \forall i \neq j$, a decomposition similar to that used in the proof of Theorem 3 shows that each sample is learned independently of the other, and class frequency has no impact. However, completely orthogonal data is rare. In the last layer of neural networks, we expect samples from the same class to be mapped to similar representation (Papyan et al., 2020), a phenomenon also observed under class imbalance (Thrampoulidis et al., 2022). Using a bias term also increases alignment between samples, as it is equivalent to adding a dimension where each sample has the same value.

In the setting of Theorem 3, class imbalance has an impact because samples from the same class are collinear, even though samples from separate classes are orthogonal. A more realistic mixture model where samples from the same class are aligned ($|\langle \mathbf{x}_i, \mathbf{x}_j \rangle| > \delta$ if $y_i = y_j$) but independent otherwise ($|\langle \mathbf{x}_i, \mathbf{x}_j \rangle| \leq \epsilon$ if $y_i \neq y_j$), as the setting of Feldman (2020) would also exhibit class separation. The class imbalance appears in Figure 2.4 because we draw the inputs from a high-dimensional uniform distribution on $[0, 1]^d$, ensuring that for any two samples $\mathbf{x}_i, \mathbf{x}_j$, $\langle \mathbf{x}_i, \mathbf{x}_j \rangle > 0$. If the data was sampled from $\mathcal{N}(0, 1)^d$ in sufficiently high dimension, the samples would be independent enough to avoid the slowdown due to class imbalance. We illustrate this in Figure A.10, where we use a smaller synthetic data with inputs drawn from $\mathcal{N}(1, 1)$ and $\mathcal{N}(0, 1)$. The zero-mean data is be approximately orthogonal as $d > n$ and does not exhibit a slow progress on low-frequency classes.



(a) **Aligned data – sampled from $\mathcal{N}(1,1)^d$**



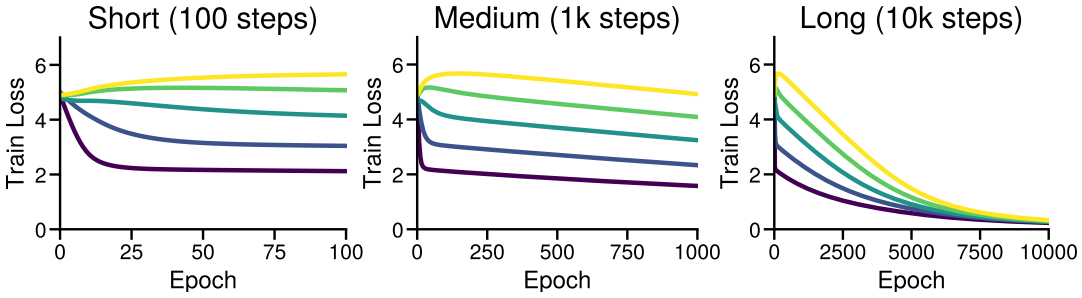
(b) **Independent data – sampled from $\mathcal{N}(0,1)^d$**

Figure A.10: **The distribution of the inputs can have a large impact on optimization.** Linear model on the Random Heavy-Tailed Labels dataset, with Inputs sampled from $\mathcal{N}(1,1)$ (a) and $\mathcal{N}(0,1)$ (b).

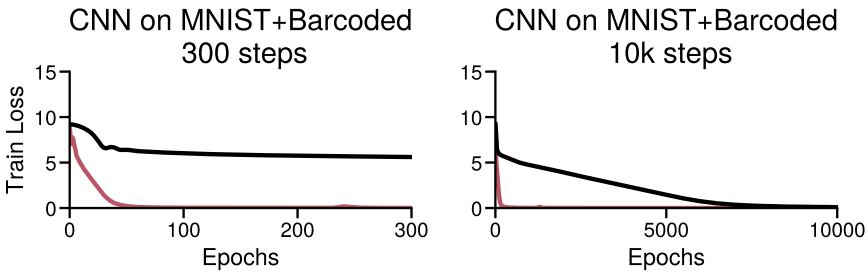
The behavior of GD on aligned data appears to be a better representation of the behavior of GD on language transformers, as we observe a performance separation per class frequency on GD, even when tuning only the last layer of a language transformer in Figure 2.5. Although the embedding weights are initialized to be zero-mean Gaussian noise, the representation of the tokens in a transformer are aligned, and this alignment increases with depth (Noci et al., 2022, e.g.).

A.4.2 An early iteration problem

As GD is slower than Adam at fitting the low-frequency classes, it might seem that GD does not fit the low-frequency classes at all. But when run for longer, GD converges and fits all classes. We show this behavior on the linear model and the CNN on imbalanced MNIST in Figure A.11. This highlight that the difference between the algorithms is primarily a difference at the start of training. However, this "start" can be quite long. In the transformer of Figure 2.1, the average loss on 10% of the data corresponding to the least frequent classes is still higher than at initialization after 15k steps.



(a) **Linear model on synthetic data**



(b) **CNN on MNIST**

Figure A.11: **Training with GD eventually drives the loss down for all classes.** Using the same step-size for different horizons (100, 1k, 10k). GD eventually drives the loss down for all classes, but the loss for the least-frequent classes only decreases below its value at initialization after 1k steps. (a) Linear model on synthetic data, (b) CNN on MNIST.

A.4.3 Impact of regularization

The data used with the linear model of Figure 2.4 is separable, meaning the predicted probabilities for the correct class will converge to 1 while the magnitude of weights go to ∞. This might lead to concerns that the observed behavior is tied to the weights growing without bounds. In Figure A.12, we show that the gap between GD and Adam still appears with regularization limiting the magnitude of the weights. However, as regularization is increased, the L2 penalty makes it difficult to fit low-frequency classes, the problem looks more like $\lambda \frac{1}{2} \|\cdot\|^2$, and the gap between the methods disappears.

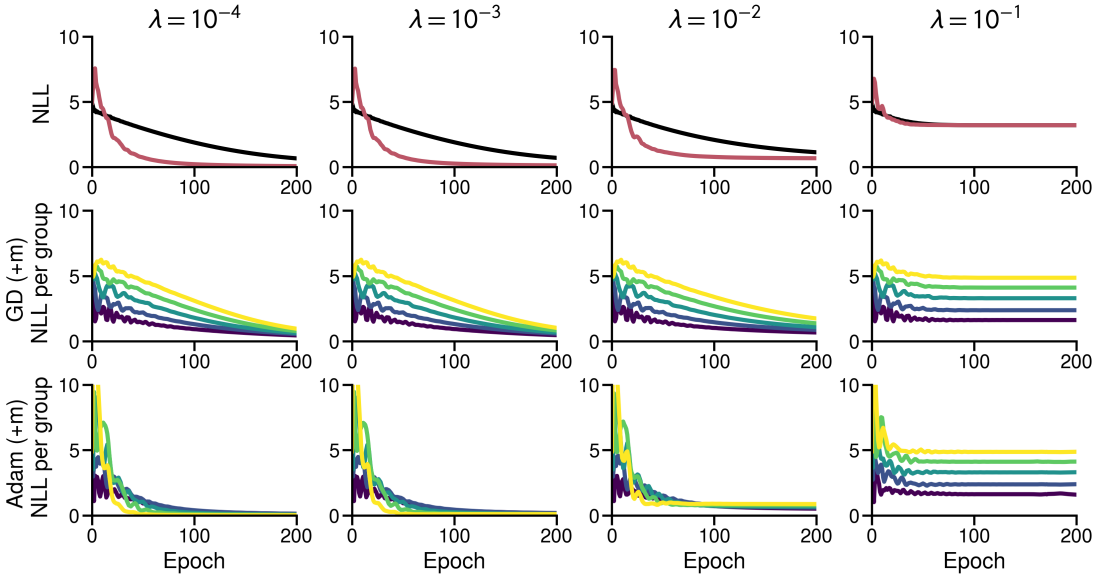


Figure A.12: **The separation between GD and Adam still appears when using L_2 regularization.** Using varying levels of regularization λ on the linear model of Figure 2.4. The plots show the negative log-likelihood and do not include the L_2 penalty.

A.5 Alternative optimizers

Figure 2.5 in Section 2.2.3 we compared GD and Adam to normalized GD and sign descent on the last layer of a one-module transformer on TinyPTB, showing that Adam and sign descent perform similarly. We repeat this experiment on other settings here to confirm that sign descent leads to similar benefits as Adam on low-frequency classes, and that changing the direction, as in sign descent, has more impact than just changing the magnitude, as in normalized GD.

We also observe this behavior on the following problems:

- Figure A.13: A linear model on **Random Heavy-Tailed Labels**, as in Figure 2.4.
- Figure A.14: A one-module transformer on **TinyPTB**, as in Figure A.4, training all layers.
- Figure A.15: A CNN on **MNIST+Barcoded**, as in Figure 2.2.

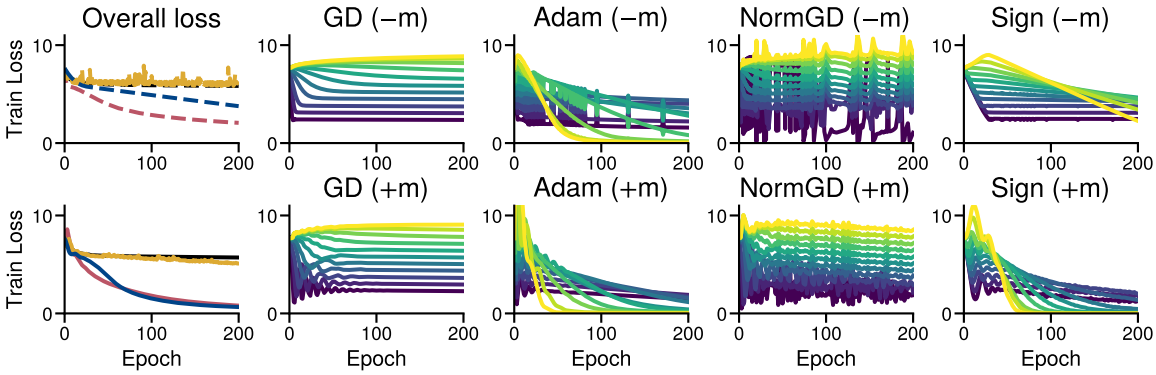


Figure A.13: **All optimizers on the linear model of Figure 2.4.**

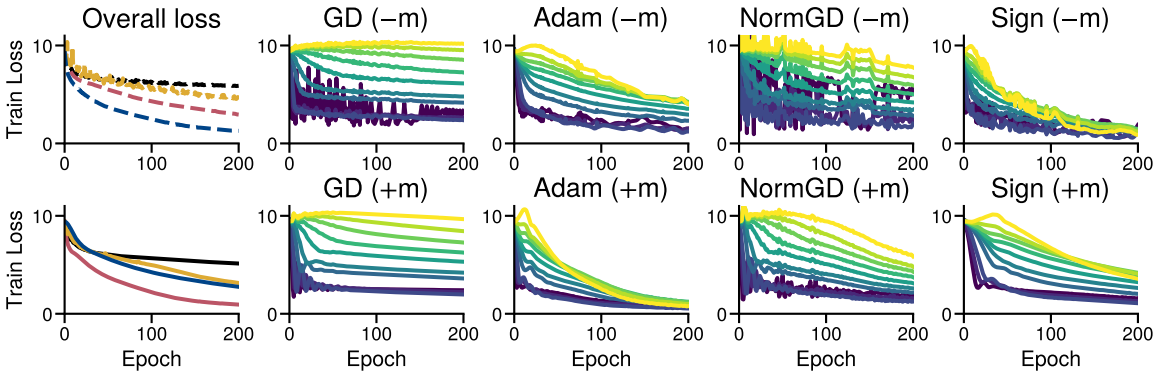


Figure A.14: **All optimizers on the transformer of Figure A.4.**

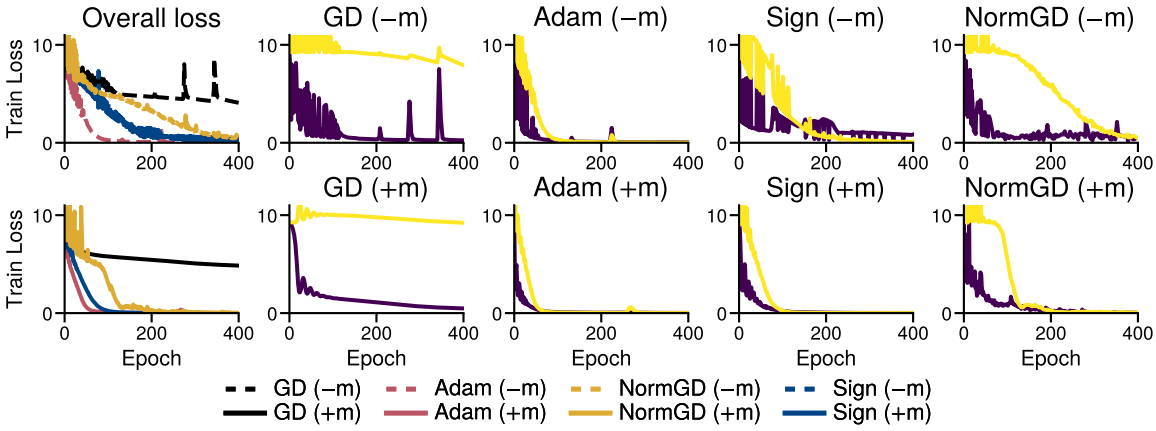


Figure A.15: **All optimizers on the CNN of Figure 2.2.** First column: Overall training loss. Remaining: Loss by frequency groups for each optimizer, with and without momentum (+m, bottom/−m, top).

A.5.1 Up-weighting low-frequency classes can improve the performance of SGD

To support Section 2.2.3, we show show that upweighting low-frequency classes helps reduce the performance gap between SGD and Adam on problems with heavy-tailed class imbalance, providing evidence that the optimization difficulties are associated with class imbalance.

While reweighting the loss of samples from class k by $1/\pi_k$ to address the class imbalance seems intuitive, optimizing the reweighted loss is no longer guaranteed to lead to progress on the original loss, especially if the weights are large. Indeed, we find that on some problems this reweighting does not improve performance (although SGD and Adam perform similarly on the reweighted loss, not shown). However, the less extreme reweighting of $1/\sqrt{\pi_k}$ appears to consistently outperform SGD.

In Figure A.16, we run SGD on the reweighted loss with the two weighting schemes, $1/\pi_k$ and $1/\sqrt{\pi_k}$ and plot its performance on the original, unweighted loss. We compare the performance of the two reweighting schemes with SGD and Adam, all with momentum, on the following 4 problems.

- The small transformer on PTB in Figure A.3 (stochastic training)
- The Linear model on synthetic data in Figure 2.4 (deterministic training)
- The CNN on MNIST+Barcoded dataset in Figure 2.2 (deterministic training)
- The ResNet18 on the Heavy-Tailed ImageNet dataset in Figure 2.3 (stochastic training)

We found that the combination of both Adam and reweighting did not improve over running Adam on the original loss and do no include it in Figure A.16.

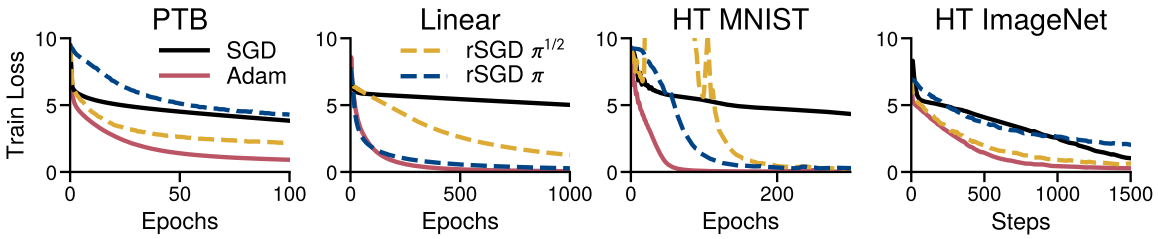


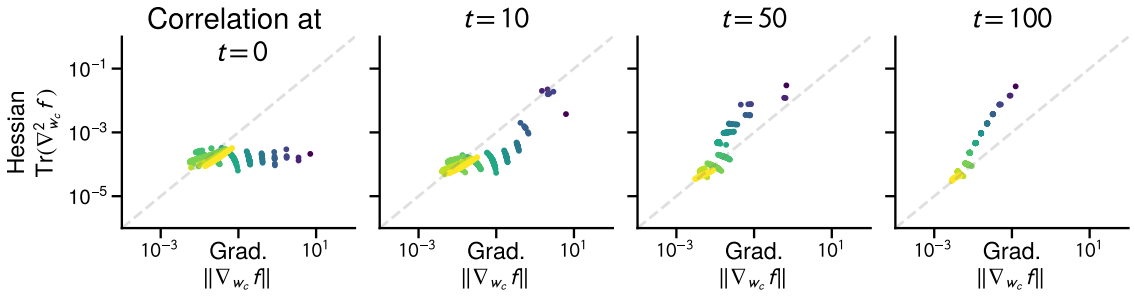
Figure A.16: **Reweighting the loss improves the performance of (S)GD on low-frequency classes.** The plots show the unweighted loss, while (S)GD and Adam optimize a reweighted loss. Reweighted (S)GD (r(S)GD) with weights $1/\sqrt{\pi_k}$ consistently outperforms plain SGD, although it can lead to spikes, as on the CNN on the MNIST dataset. Reweighting with weights $1/\pi_k$ is sometimes better (Linear, MNIST) but can be worse (PTB, ImageNet) as it optimizes a different objective. We use deterministic updates for the first 3 problems, labeled Epoch, and stochastic updates for the ResNet18 on heavy-tailed ImageNet.

A.6 Dynamics of the gradient and Hessian

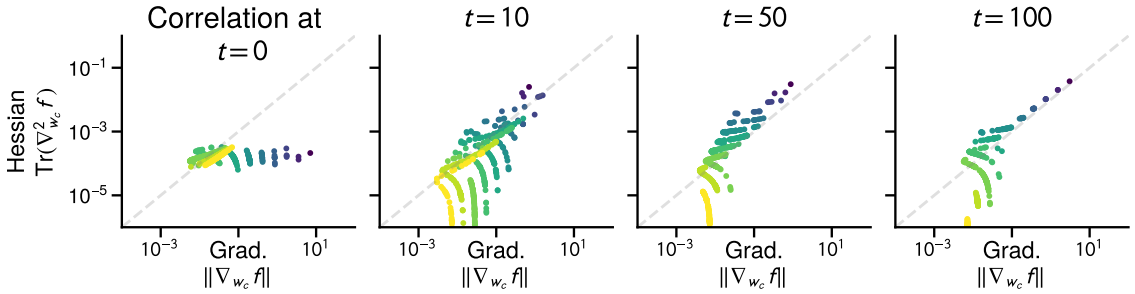
This section provides additional details on the dynamics of (S)GD and Adam discussed in Section 2.3.2.

- Figure A.17 shows the dynamics of GD and Adam on the linear model on synthetic data in Figure 2.4 (deterministic training). This figure complements Figure 2.7 which shows the dynamics over the path taken by Adam.
- Figure A.18 and additionally shows the average predicted probabilities p for each frequency group, showing that the deviation from the linear relationship for rare classes coincides with the predicted probabilities p for those classes going to 1.
- The following figures show the correlation on additional problems, on
  - Figure A.19 The GPT2-Small model on WikiText-103 in Figure 2.1 (stochastic training). This figure complements Figure 2.8 which shows the dynamics over the path taken by Adam.
  - Figure A.20 The CNN on the MNIST+Barcoded dataset in Figure 2.2 (deterministic training)
  - Figure A.22 The small transformer on PTB in Figure A.3 (stochastic training)
  - Figure A.21 The ResNet18 on the Heavy-Tailed ImageNet dataset in Figure 2.3 (stochastic training)
- Figure A.23 illustrates that this correlation does not hold globally and only emerges throughout training by showing that a *negative* correlation can instead be found by looking at the oppositve path of the path taken by Adam, $-\mathbf{W}_t$ (when $\mathbf{W}_t$ are the iterates generated by Adam).

A.6.1 Linear model on synthetic data



(a) **Dynamics over the path of GD**



(b) **Dynamics over the path of Adam**

Figure A.17: **Evolution of the gradient norm and Hessian trace through optimization.** Taken over the path of GD (a) and Adam (b) on the linear problem of Figure 2.4. The blocks correspond to the rows $\mathbf{w}_1, ..., \mathbf{w}_c$ of the parameter matrix $\mathbf{W}$. The color indicates the class frequency, showing that lower (higher) frequency classes have smaller (larger) gradient norm and Hessian trace. Figure A.17(b) is a replication of Figure 2.7, given here for convenience. The deviation from the correlation is explainable by the fact that difference classes are learned at difference speed, leading to a different value of p in Proposition 2, shown in Figure A.18. For GD, frequent classes are learned faster than infrequent ones, while for Adam, p is similar among the most frequent groups of classes while $p \to 1$ for the least frequent classes.

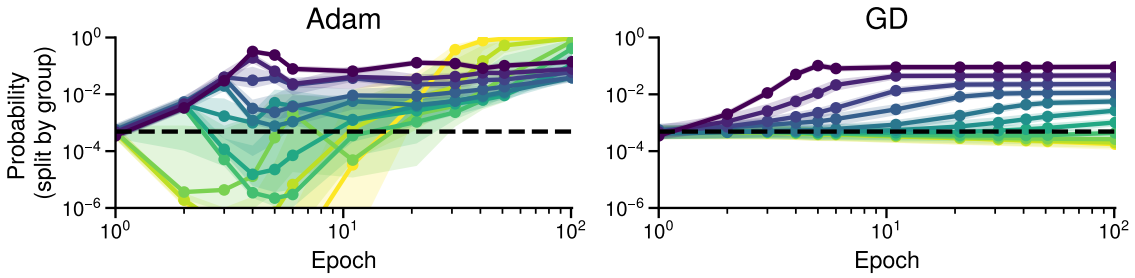
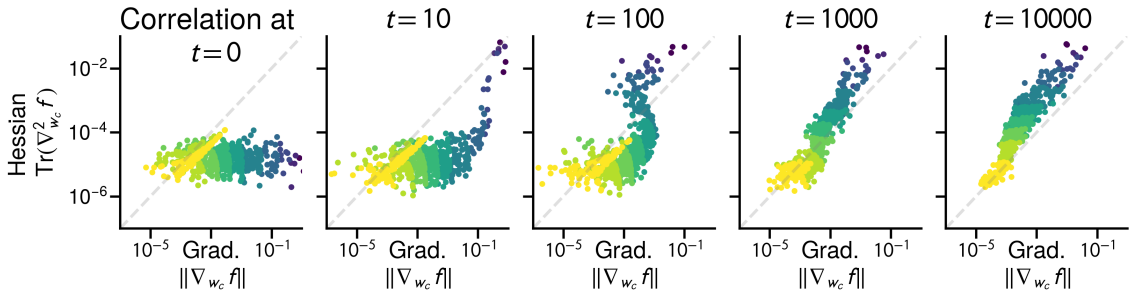
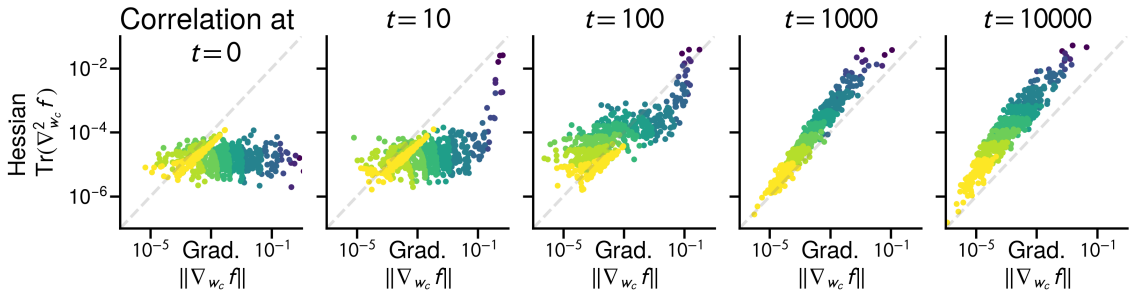


Figure A.18: **Evolution of the predicted probabilities for the correct class.** Complement to Figure A.17, taken over the path of GD and Adam on the linear problem of Figure 2.4. For GD, frequent classes are learned faster than infrequent ones, Adam has a similar behavior on the most frequent groups of classes but also increases the predicted probability for the correct class on infrequent groups. The color indicates the class frequency, showing that lower (higher) frequency classes have smaller (larger) gradient norm and Hessian trace.

A.6.2 GPT2-Small on WikiText-103



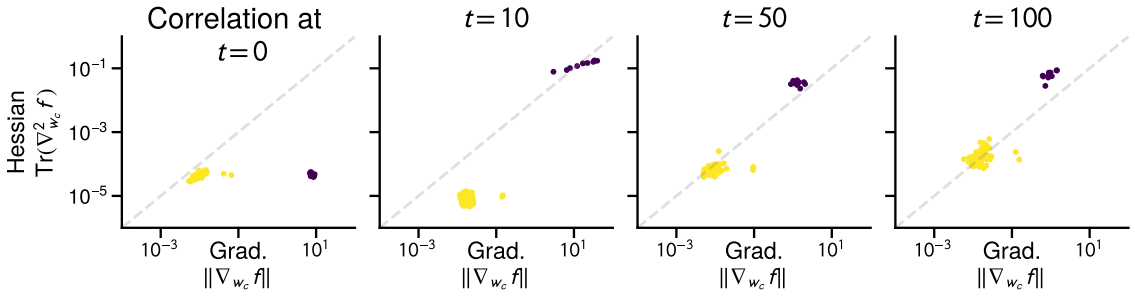
(a) **Dynamics over the path of SGD**



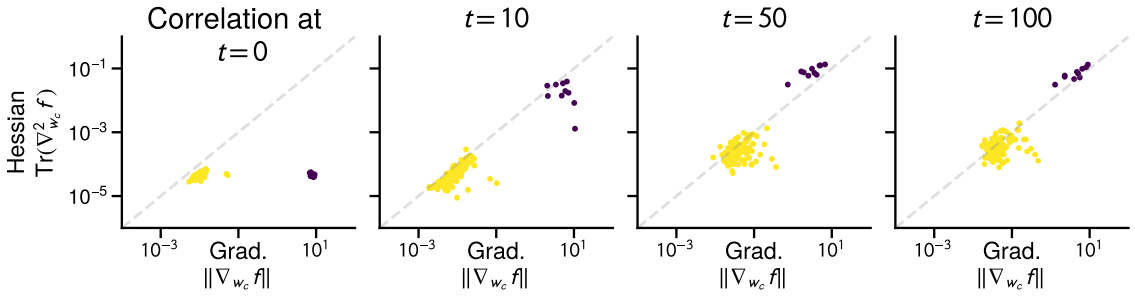
(b) **Dynamics over the path of Adam**

Figure A.19: **The gradient-Hessian blocks also become correlated in the last layer of large models.** Reproducing Figure 2.7 on the GPT2-Small/WikiText-103 problem of Figure 2.1. Evolution of the gradient norm and Hessian trace for each row $\mathbf{w}_c$ of the last layer throughout optimization, over the path taken by SGD (a) and Adam (b). The color indicates the class frequency, showing that lower (higher) frequency classes have smaller (larger) gradient norm and Hessian trace.

A.6.3 CNN on Barcoded+MNIST



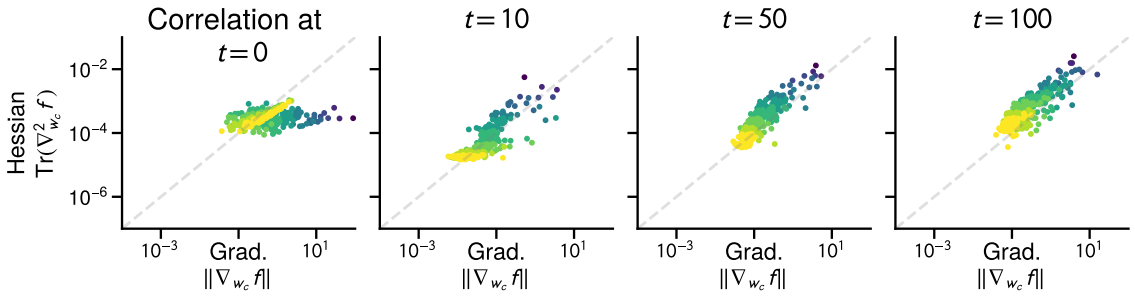
(a) **Dynamics over the path of GD**



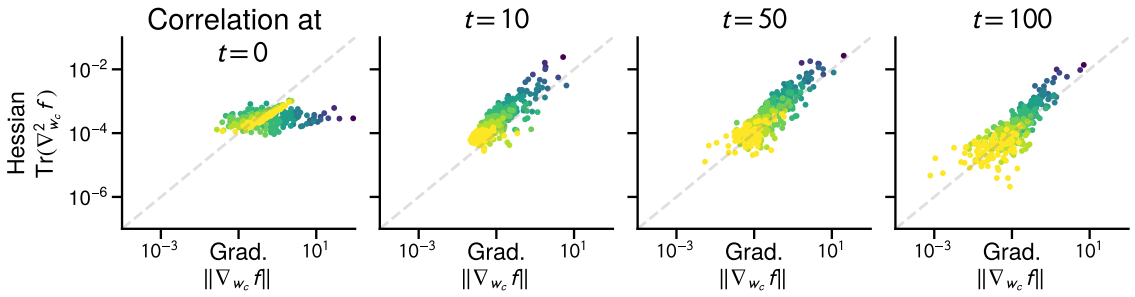
(b) **Dynamics over the path of Adam**

Figure A.20: **Evolution of the gradient norm and Hessian trace through optimization.** Taken over the path of GD and Adam on the CNN on imbalanced MNIST in Figure 2.2. Note that this problem only has two groups of classes with different frequencies; 10 classes have ≈5k samples while 10k classes have 5 samples.

A.6.4 ResNet18 on Heavy-Tailed ImageNet



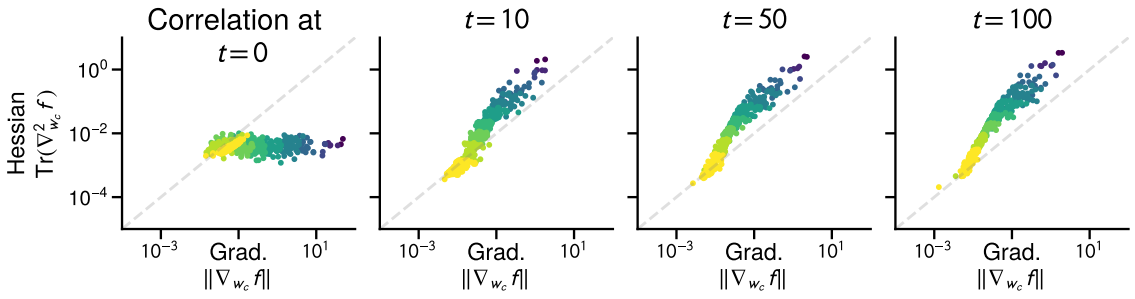
(a) **Dynamics over the path of SGD**



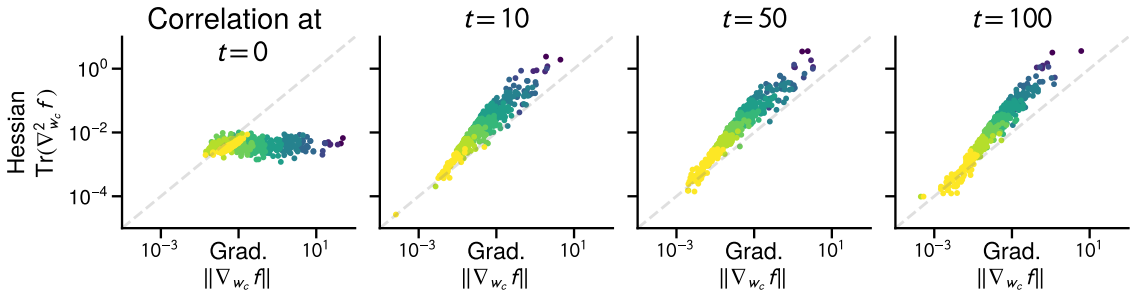
(b) **Dynamics over the path of Adam**

Figure A.21: **Evolution of the gradient norm and Hessian trace through optimization.** Taken over the path of SGD and Adam on the ResNet18 on Heavy-Tailed ImageNet in Figure 2.3.

A.6.5 Small Transformer on PTB



(a) **Dynamics over the path of SGD**



(b) **Dynamics over the path of Adam**

Figure A.22: **Evolution of the gradient norm and Hessian trace through optimization.** Taken over the path of SGD and Adam on the small Transformer on PTB in Figure A.3.

A.6.6 The correlation depends on the path

Proposition 2 requires that the optimizer make progress and assign samples to their correct classes. Indeed, the positive correlation observed in the previous figures is not a global property of the loss function. Not only does it not hold at initialization, where the Hessian is uniform, the correlation can even be reversed in some areas of the parameter space, as shown in Figure A.23.

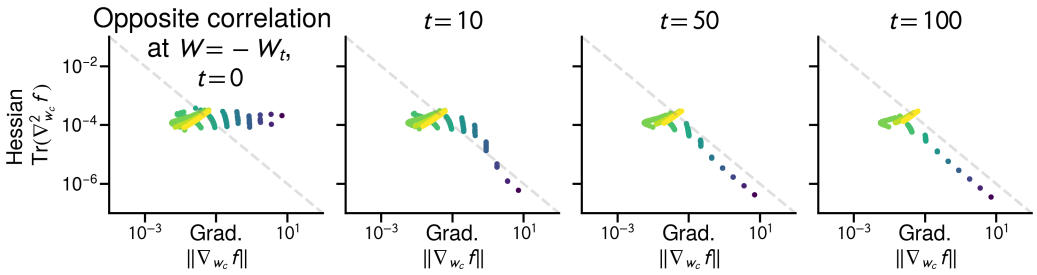


Figure A.23: **The correlation only holds while training.** Correlation between the gradient and Hessian blocks through the path $\{-\mathbf{W}_t\}$, where $\mathbf{W}_t$ are the iterates of Adam on the linear model of Figure 2.4.

A.7 Correlation between the gradient and Hessian across blocks

This section gives the proof of Proposition 2 in Section 2.3.2

Proposition 2. *If initialized at $\mathbf{W}_0 = 0$, the gradient and Hessian of the loss $\mathcal{L}$ w.r.t. $\mathbf{w}_k$ are*

$$\nabla_{\mathbf{w}_k} \mathcal{L}(\mathbf{W}_0) = \pi_k \bar{\mathbf{x}}^k - \tfrac{1}{c}\bar{\mathbf{x}}, \qquad \nabla^2_{\mathbf{w}_k} \mathcal{L}(\mathbf{W}_0) = \tfrac{1}{c}\big(1 - \tfrac{1}{c}\big)\bar{\mathbf{H}}, \tag{2.1}$$

During training, if the model correctly assigns samples to class k with probability p (Assumption 1),

$$\begin{aligned} \nabla_{\mathbf{w}_k} \mathcal{L} &= \ (1-p)\pi_k\, \bar{\mathbf{x}}^k \ + O\big(\tfrac{1}{c}\big), \\ \nabla^2_{\mathbf{w}_k} \mathcal{L} &= p(1-p)\pi_k\, \bar{\mathbf{H}}^k + O\big(\tfrac{1}{c}\big), \end{aligned} \quad \textit{and} \quad \|\nabla_{\mathbf{w}_k} \mathcal{L}\| \sim \left(\frac{1}{p}\frac{\|\bar{\mathbf{x}}^k\|}{\operatorname{Tr}(\bar{\mathbf{H}}^k)}\right) \operatorname{Tr}(\nabla^2_{\mathbf{w}_k} \mathcal{L}) \ \textit{as } c \to \infty, \tag{2.2}$$

for classes where the frequency does not vanish too quickly, $\pi_k = \omega(1/c)$.

The requirement that the class frequencies do not vanish, $\pi_k = \omega(1/c)$, is necessary to make it possible to discuss class frequencies as $c \to \infty$, unless the class frequencies do not depend on c. While the frequencies π_k and the number of classes c can be independent, for example if π_k follows an exponential decay, $\pi_k \propto 2^{-k}$, it does not hold for all distributions. While it may seem that this result only holds for relatively frequent classes, as it requires $\pi_k c \to \infty$, we can see that nearly all the data comes from classes where this correlation holds when the classes are distributed as $\pi_k \propto 1/k$. Denote by $H(c) = \sum_{k=1}^{c} 1/k = \Theta(\log c)$. After normalization, we have $\pi_k = 1/kH(c)$. The correlation result holds as long as $\pi_k c \to \infty$, and so it at least holds for the first $k \leq c/\log(c)^2$ classes as $\pi_k c \geq \log(c) \to \infty$. While this only cover a $1/\log(c)^2$ fraction of the classes, those classes account for nearly all the data as

$$\sum_{k=1}^{\left\lceil \frac{c}{\log(c)} \right\rceil} \pi_k = \frac{H\big(\lceil c/\log(c)^2 \rceil\big)}{H(c)} = \Theta\left(\frac{\log(c) - 2\log\log(c)}{\log(c)}\right) \to 1.$$

Proof of Proposition 2. We first recall the gradient and Hessian for each block $\mathbf{w}_1, ..., \mathbf{w}_c$;

$$\nabla_{\mathbf{w}_k} \ell(\mathbf{W}, \mathbf{x}, \mathbf{y}) = (y = k - \mathbf{p}(\mathbf{x})_k)\mathbf{x}, \qquad \nabla^2_{\mathbf{w}_k} \ell(\mathbf{W}, \mathbf{x}, \mathbf{y}) = \mathbf{p}(\mathbf{x})_k(1 - \mathbf{p}(\mathbf{x})_k)\mathbf{x}\mathbf{x}^\top,$$

and the definitions of the moments of the data, per class and overall.

$$\bar{\mathbf{x}}^k = \tfrac{1}{n_k}\textstyle\sum_{i=1:y_i=k}^{n} \mathbf{x}_i, \quad \bar{\mathbf{x}} = \tfrac{1}{n}\textstyle\sum_{i=1}^{n} \mathbf{x}_i, \quad \bar{\mathbf{H}}^k = \tfrac{1}{n_k}\textstyle\sum_{i=1:y_i=k}^{n} \mathbf{x}_i\mathbf{x}_i^\top, \quad \bar{\mathbf{H}} = \tfrac{1}{n}\textstyle\sum_{i=1}^{n} \mathbf{x}_i\mathbf{x}_i^\top.$$

Our first step is to rewrite the sums for the gradient and Hessian to separate the influence of the samples of the correct class k and the other samples.

$$\begin{aligned}
\nabla_{\mathbf{w}_k}\mathcal{L}(\mathbf{W}) &= \frac{1}{n}\sum_{i=1}^{n}(y_i = k - \mathbf{p}(\mathbf{x}_i)_k)\mathbf{x}_i, \\
&= \frac{1}{n}\sum_{j=1}^{c}\sum_{i:y_i=j}(y_i = k - \mathbf{p}(\mathbf{x}_i)_k)\mathbf{x}_i, && \text{(Split by class)} \\
&= \sum_{j=1}^{c}\frac{\pi_j}{n_j}\sum_{i:y_i=j}(y_i = k - \mathbf{p}(\mathbf{x}_i)_k)\mathbf{x}_i, && \text{(Use class frequencies } \pi_j = n_j/n) \\
&= \pi_k\frac{1}{n_k}\sum_{i=1:y_i=k}^{n}(1 - \mathbf{p}(\mathbf{x}_i)_k)\mathbf{x}_i + \sum_{j=1,j\neq k}^{c}\frac{\pi_j}{n_j}\sum_{i:y_i=j}(-\mathbf{p}(\mathbf{x}_i)_k)\mathbf{x}_i. \\
\nabla^2_{\mathbf{w}_k}\mathcal{L}(\mathbf{W}) &= \frac{1}{n}\sum_{i=1}^{n}\mathbf{p}(\mathbf{x}_i)_k(1 - \mathbf{p}(\mathbf{x}_i)_k)\mathbf{x}_i\mathbf{x}_i^\top, \\
&= \frac{\pi_k}{n_k}\sum_{i:y_i=k}\mathbf{p}(\mathbf{x}_i)_k(1 - \mathbf{p}(\mathbf{x}_i)_k)\mathbf{x}_i\mathbf{x}_i^\top + \sum_{j=1,j\neq k}^{c}\frac{\pi_j}{n_j}\sum_{i:y_i=j}\mathbf{p}(\mathbf{x}_i)_k(1 - \mathbf{p}(\mathbf{x}_i)_k)\mathbf{x}_i\mathbf{x}_i^\top.
\end{aligned}$$

We can simplify the first terms using the assumption that $p(\mathbf{x}_i)_k = p$ for samples of the correct class,

$$\frac{\pi_k}{n_k}\sum_{i=1:y_i=k}^{n}(1 - \mathbf{p}(\mathbf{x}_i)_k)\mathbf{x}_i = (1-p)\pi_k\bar{\mathbf{x}}^k, \quad \frac{\pi_k}{n_k}\sum_{i:y_i=k}\mathbf{p}(\mathbf{x}_i)_k(1 - \mathbf{p}(\mathbf{x}_i)_k)\mathbf{x}_i\mathbf{x}_i^\top = p(1-p)\pi_k\bar{\mathbf{H}}^k.$$

We introduce the following shorthands for the second terms,

$$\mathbf{d}_k = c\sum_{j=1,j\neq k}^{c}\frac{\pi_j}{n_j}\sum_{i:y_i=j}(-\mathbf{p}(\mathbf{x}_i)_k)\mathbf{x}_i, \qquad \mathbf{D}_k = c\sum_{j\neq k}\frac{\pi_j}{n_j}\sum_{i:y_i=j}\mathbf{p}(\mathbf{x}_i)_k(1 - \mathbf{p}(\mathbf{x}_i)_k)\mathbf{x}_i\mathbf{x}_i^\top.$$

Using those simplifications, we obtain that

$$\nabla_{\mathbf{w}_k}\mathcal{L}(\mathbf{W}) = (1-p)\pi_k\bar{\mathbf{x}}^k + \frac{1}{c}\mathbf{d}_k, \qquad \nabla^2_{\mathbf{w}_k}\mathcal{L}(\mathbf{W}) = p(1-p)\pi_k\bar{\mathbf{H}}^k + \frac{1}{c}\mathbf{D}_k.$$

The terms $\mathbf{d}_k$, $\mathbf{D}_k$ are averages of terms weighted by $c\mathbf{p}(\mathbf{x}_i)_k$, which by assumption is $O(1)$, and as such both $\|\mathbf{d}_k\|$ and $\mathrm{Tr}(\mathbf{D}_k)$ are $O(1)$. The ratio between the two will be dominated by the contribution of their first term as long as π_k dominates $1/c$, in the sense that $\lim_{c\to\infty}\frac{1}{\pi_k c} \to 0$, as

$$\begin{aligned}
\lim_{c\to\infty}\frac{\|\nabla_{\mathbf{w}_k}\mathcal{L}\|}{\mathrm{Tr}(\nabla^2_{\mathbf{w}_k}\mathcal{L})} &= \lim_{c\to\infty}\frac{\left\|(1-p)\pi_k\bar{\mathbf{x}}^k + \frac{1}{c}\mathbf{d}_k\right\|}{\mathrm{Tr}(p(1-p)\pi_k\bar{\mathbf{H}}^k + \frac{1}{c}\mathbf{D}_k)} \\
&= \lim_{c\to\infty}\frac{\left\|(1-p)\bar{\mathbf{x}}^k + \frac{1}{c\pi_k}\mathbf{d}_k\right\|}{\mathrm{Tr}(p(1-p)\pi_k\bar{\mathbf{H}}^k + \frac{1}{c\pi_k}\mathbf{D}_k)} = \frac{1}{p}\frac{\|\bar{\mathbf{x}}^k\|}{\mathrm{Tr}(\bar{\mathbf{H}}^k)}.
\end{aligned}$$

□

A.7.1 Off-diagonal blocks are orders of magnitude smaller than diagonal blocks

Our discussion Section 2.3.2 ignored the impact of off-diagonal blocks. In this section, we show that they are small. The diagonal and off-diagonal blocks of the matrix for $k \neq k'$.

$$\begin{aligned} & & \mathbf{H}_{kk} &:= \nabla^2_{\mathbf{w}_k} \ell(\mathbf{W}, \mathbf{x}, y) &&= \mathbf{p}(\mathbf{x})_k(1 - \mathbf{p}(\mathbf{x})_k)\mathbf{x}\mathbf{x}^\top, \\ \text{and for } j \neq k, & & \mathbf{H}_{kj} &:= \nabla_{\mathbf{w}_k}\nabla_{\mathbf{w}_{k'}} \ell(\mathbf{W}, \mathbf{x}, \mathbf{y}) &&= \mathbf{p}(\mathbf{x})_k(\ \ - \mathbf{p}(\mathbf{x})_{k'})\mathbf{x}\mathbf{x}^\top. \end{aligned}$$

From this, we can see that, on average, the magnitude of the off-diagonal blocks will be smaller than that of the diagonal blocks, as

$$\mathbf{H}_{kk} = -\sum_{j=1, j\neq k}^{c} \mathbf{H}_{kj},$$

because $\sum_{k'=1,k'\neq k}^{c} \mathbf{p}(\mathbf{x})_k \mathbf{p}(\mathbf{x})_{k'} = \mathbf{p}(\mathbf{x})_k(1 - \mathbf{p}(\mathbf{x})_k)$, This means that the matrix $\mathbf{T} : [c \times c]$ formed by taking the trace of the blocks, $\mathbf{T}_{jk} = \mathrm{Tr}(\mathbf{H}_{jk})$, is diagonally dominant.

Figures 2.9 and A.24 show that the magnitude of the entries of the Hessian in off-diagonal blocks is orders of magnitude smaller than those of the diagonal blocks. Instead of plotting the $[cd \times cd]$ Hessian, we subsample 40 classes and 40 input dimensions and plot the resulting $[160 \times 160]$ entries at different points throughout the trajectory of Adam on the problem of Figure 2.4. Figure 2.9 shows the matrices with classes sampled uniformly and Figure A.24 with classes sampled log-uniformly

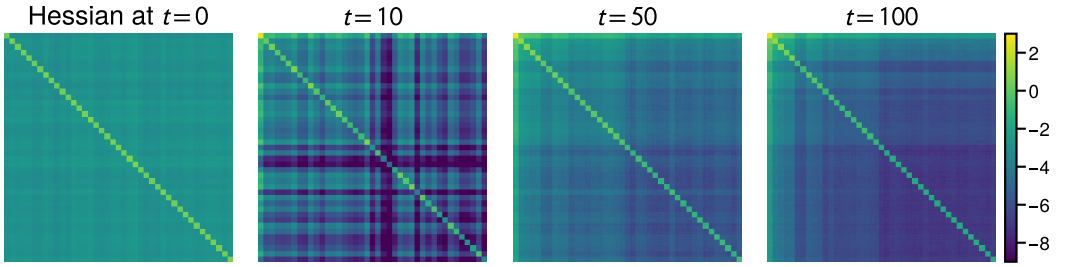


Figure A.24: **The off-diagonal blocks are much smaller than the diagonal blocks.** Showing the magnitude $\log_{10}(|(\nabla^2\mathcal{L})_{ij}|)$ for a $[160 \times 160]$ subset of the Hessian, sampling 40 classes and 40 input dimensions uniformly.

A.8 Continuous time GD and sign descent on a simple imbalanced problem

We give the proof of Theorem 3 on the simple imbalanced setting, restated here for convenience.

Simple imbalanced setting. *Consider c classes with frequencies $\pi_1, ..., \pi_c$ where all samples from a class are the same, $\mathbf{x}_i = \mathbf{e}_k$ if $y_i = k$, where $\mathbf{e}_k$ is the kth standard basis vector in $\mathbb{R}^c$.*

Theorem 3. *On the* simple imbalanced setting*, gradient flow and continuous time sign descent initialized at $\mathbf{W} = 0$ minimize the loss of class k, $\ell_k(t) = -\log(\sigma(\mathbf{W}(t)\mathbf{e}_k)_k)$, at the rate*

$$\textit{Gradient flow:} \quad \ell_k(t) = \Theta(1/\pi_k t), \quad \textit{Continuous time sign descent:} \quad \ell_k(t) = \Theta\big(e^{-ct}\big).$$

We separate the proof for gradient flow into 3 parts. Lemma 4 simplifies the dynamics into smaller, independent differential equations, Lemma 5 solves the differential equation and Lemma 6 bounds the loss. The proof uses similar tools as for the gradient flow dynamics studied by Cabannes et al. (2024), but we focus instead on the loss per class. We treat continuous time sign descent separately in Lemma 7.

Notation. If $\mathbf{W}$ is a $[a \times b]$ matrix, then $\mathbf{w}_1, ..., \mathbf{w}_a$ are the rows and $\mathbf{w}^1, ..., \mathbf{w}^b$ are the vectors, and w_{ij} is the entry at the ith column, jth row. For brevity, we use $z = c - 1$ as the term appears often.

Lemma 4 (Separation of the dynamics)**.** *The dynamics of the parameter matrix $\mathbf{W}$ separate into c 2-dimensional differential equations, $w_{kk}(t) = a_k(t)$ and $w_{jk}(t) = b_k(t)$ for $j \neq k$, where*

$$\begin{aligned} a_k(0) &= 0, & a_k &= \pi_k\left(1 - \frac{\exp(a_k)}{\exp(a_k) + (c-1)\exp(b_k)}\right), \\ b_k(0) &= 0, & b_k &= \pi_k\left(\ \ - \frac{\exp(b_k)}{\exp(a_k) + (c-1)\exp(b_k)}\right). \end{aligned}$$

Proof. Our goal is to simplify the dynamics starting at $\mathbf{W}(0) = 0$ and following the gradient flow $\dot{\mathbf{W}} = -\nabla\mathcal{L}(\mathbf{W})$, where $\mathbf{W} : [c \times d]$. For the simplified setting, we have that $d = c$ are the inputs are the standard basis vectors in $\mathbb{R}^c$. The derivative of $\mathcal{L}$ w.r.t. a single element w_{kj} is

$$\partial_{w_{kj}}\mathcal{L}(\mathbf{W}) = -\pi_k k = j + \pi_j \sigma(\mathbf{w}^j)_k.$$

As $\partial_{w_{kj}}$ only depends on $\mathbf{w}^j$ for all k, The dynamics are independent across the columns of $\mathbf{W}$, giving c independent equations in $\mathbb{R}^c$,

$$\mathbf{w}^j(0) = 0, \qquad \dot{\mathbf{w}}^j = \pi_j(\mathbf{e}_j - \sigma(\mathbf{w}^j)).$$

To further simplify the dynamics, we use the fact that the weights that are not associated with the correct class have the same dynamics. For any indices i, j different from k, $w_{ik}(t) = w_{jk}(t)$. They have the same derivatives if they have the same value, as

$$-w_{ik} = \pi_k \sigma(\mathbf{w}^k)_i = \pi_k \frac{\exp(w_{ik})}{\sum_{k'} \exp(w_{k'k})} = \pi_k \frac{\exp(w_{jk})}{\sum_{k'} \exp(w_{k'k})} = \pi_k \sigma(\mathbf{w}^k)_j = -w_{jk},$$

so they will have the same dynamics and the equation can be reduced to a system of 2 variables, $w_{kk} = a_k$ and $w_{jk} = b_k$ for any $j \neq k$, with

$$\begin{aligned} a_k(0) &= 0, & a_k &= \pi_k \left(1 - \frac{\exp(a_k)}{\exp(a_k) + (c-1)\exp(b_k)}\right), \\ b_k(0) &= 0, & b_k &= \pi_k \left(- \frac{\exp(b_k)}{\exp(a_k) + (c-1)\exp(b_k)}\right). \end{aligned}$$

□

Lemma 5 (Solution of the dynamics)**.** *For a given class with frequency π, the dynamics of the parameters a and b in Lemma 4 evolve as follows, using the shortcuts $f(t) = 1 + c\pi t$ and $z = c - 1$,*

$$a(t) = \frac{1}{c}\left(f(t) - zW\left(\frac{1}{z}\exp\left(\frac{1}{z}f(t)\right)\right)\right) \qquad b(t) = -\frac{1}{z}a(t),$$

Proof. We want the solution to the differential equation

$$\begin{aligned} a(0) &= 0 & a &= \pi \left(1 - \frac{\exp(a)}{\exp(a) + (c-1)\exp(b)}\right), \\ b(0) &= 0 & b &= \pi \left(- \frac{\exp(b)}{\exp(a) + (c-1)\exp(b)}\right). \end{aligned}$$

The general solution, ignoring the initial conditions, uses the Lambert W function and constants K_1, K_2.[3] For brevity, we introduce the shortcut $z = c - 1$.

$$\begin{aligned} a(t) = & \quad \frac{1}{zc}\left(ce^{-K_1}K_2 + cz\pi t - z^2 W\left(\frac{1}{z}\exp\left(\frac{c}{z^2}\left(z\pi t + e^{-K_1}K_2\right) - K_1\right)\right)\right), \\ b(t) = & \quad K_1 - \frac{1}{z^2 c}\left(ce^{-K_1}K_2 + cz\pi t - z^2 W\left(\frac{1}{z}\exp\left(\frac{c}{z^2}\left(z\pi t + e^{-K_1}K_2\right) - K_1\right)\right)\right). \end{aligned}$$

We need to set K_1, K_2 to satisfy the initial conditions $a(0) = b(0) = 0$. As $b(t) = K_1 - a(t)/z$, we must have that $K_1 = 0$, giving the simplification

$$a(t) = \frac{1}{zc}\left(cK_2 + cz\pi t - z^2 W\left(\frac{1}{z}\exp\left(\frac{c}{z^2}(z\pi t + K_2) - K_1\right)\right)\right), \qquad b(t) = -\frac{1}{z}a(t).$$

[3] WolframAlpha solution for $\pi = 1$: `https://www.wolframalpha.com/input?i=d/dt+x(t)+=+1-exp(x(t)/(exp(x(t))+c*exp(y(t))),+d/dt+y(t)+=+-exp(y(t))/(exp(x(t))+c*exp(y(t)))`

To set K_2, we need to have

$$0 = zca(0) = cK_2 - z^2 W\left(\frac{1}{z}\exp\Big(K_2\frac{c}{z^2}\Big)\right) \implies W\left(\frac{1}{z}\exp\Big(K_2\frac{c}{z^2})\Big)\right) = \frac{c}{z^2}K_2$$

Since $W(xe^x) = x$ for $x > 0$, the equation is satisfied for $K_2 = \frac{z}{c}$, as we get $W\left(\frac{1}{z}e^{\frac{1}{z}}\right) = \frac{1}{z}$, giving

$$a(t) = \frac{1}{c}\left(1 + c\pi t - zW\left(\frac{1}{z}\exp\left(\frac{1}{z}(1+c\pi t)\right)\right)\right) \qquad b(t) = -\frac{1}{z}a(t). \qquad \square$$

Lemma 6 (Bound for the loss)**.** *For t sufficiently large such that $1 + c\pi_k t \geq z\log z + 1$,*

$$\ell_k(t) = \Theta\left(\frac{1}{\pi_k t}\right).$$

Using the simplification derived in Lemma 4 and the solution of the differential equation in Lemma 5, we can rewrite the loss for a specific class as a function of time as

$$L_k(\mathbf{W}) \coloneqq -\log(\sigma(\mathbf{W}\mathbf{e}_k)_k) = -\log\left(\frac{\exp(w_{kk})}{\sum_{j=1}^{c}\exp(w_{jk})}\right),$$

$$\ell_k(t) \coloneqq L_k(\mathbf{W}(t)) = -\log\left(\frac{\exp(a_k(t))}{\exp(a_k(t)) + (c-1)\exp(b_k(t))}\right) = \log(1 + (c-1)\exp(cb_k(t))),$$

where the equality uses that $a_k(t) = (c-1)b_k(t)$. For brevity, we will drop the index k in a_k, b_k, ℓ_k and π_k and use the shortcut $z = c - 1$, bounding the quantity

$$\ell(t) = \log(1 + z\exp(cb(t))).$$

Expanding the definition of $b(t)$ using Lemma 5, we have

$$z\exp(cb(t)) = z\exp\left(-\frac{1}{z}\left(f(t) - zW\left(\frac{1}{z}\exp\left(\frac{1}{z}f(t)\right)\right)\right)\right), \quad \text{where} \quad f(t) = 1 + c\pi t.$$

To simplify the W function, we use the fact that for $x > e$ (Hoorfar and Hassani, 2008, Theorem 2.7)

$$W(x) = \log(x) - \log(\log(x)) + \delta(x) \qquad \text{where} \qquad \frac{1}{2} \leq \delta(x)\frac{\log(x)}{\log(\log(x))} \leq \frac{e}{e-1}.$$

To use this bound on $W\big(\frac{1}{z}\exp\big(\frac{1}{z}f(t)\big)\big)$, we need $\frac{1}{z}\exp\big(\frac{1}{z}f(t)\big) \geq e$, which is satisfied for t sufficiently large, once $f(t) \geq z(\log z + 1)$.

Using that $\log\big(\frac{1}{z}\exp\big(\frac{1}{z}f(t)\big)\big) = \frac{1}{z}f(t) - \log(z)$, and writing $h(t) = \delta\big(\frac{1}{z}\exp\big(\frac{1}{z}f(t)\big)\big)$, we have

$$\begin{aligned} f(t) - zW\left(\frac{1}{z}\exp\left(\frac{1}{z}f(t)\right)\right) &= f(t) - z\left(\frac{1}{z}f(t) - \log(z) - \log\left(\frac{1}{z}f(t) - \log(z)\right) + h(t)\right), \\ &= z(\log(f(t) - z\log(z)) - h(t)), \end{aligned}$$

giving the simplification

$$z\exp(cb(t)) = z\exp\left(-\frac{1}{z}\left(f(t) - zW\left(\frac{1}{z}\exp\left(\frac{1}{z}f(t)\right)\right)\right)\right),$$
$$= z\exp(-\log(f(t) - z\log(z)) + h(t)) = \frac{z\exp(h(t))}{f(t) - z\log z},$$

This gives the average loss

$$\ell(t) = \log(1 + z\exp(cb(t))) = \log\left(1 + \frac{z\exp(h(t))}{f(t) - z\log z}\right)$$

To bound this expression, we can use that $\frac{z\exp(h(t))}{f(t)-z\log z} \geq 0$ after $f(t) \geq z\log z$, which we have already assumed to apply the bound on the W function, and use the bounds $\frac{x}{1+x} \leq \log(1+x) \leq x$ to get

$$\frac{z\exp(h(t))}{f(t) - z\log z + z\exp(h(t))} \leq \ell(t) \leq \frac{z\exp(h(t))}{f(t) - z\log z}.$$

As $h(t)$ is upper bounded by a constant and $\lim_{t\to\infty} h(t) = 0$, $\lim_{t\to\infty}\exp(h(t)) = 1$, we have

$$\ell(t) = \Theta\left(\frac{z}{f(t) - z\log z}\right) = \Theta\left(\frac{1}{\pi t}\right).$$

Lemma 7. *The loss at time t for continuous time sign descent is* $\ell_k(t) = \log(1 + (c-1)\exp(-ct))$

Proof. The same decomposition as in Lemma 4 hold, with the dynamics

$$a_k(0) = 0, \qquad a_k = 1, \qquad a_k(t) = t, \qquad\qquad b_k(0) = 0, \qquad b_k = -1, \qquad b_k(t) = -t,$$

leading to the following loss

$$\ell_k(t) = \log(1 + (c-1)\exp(-ct)) = \Theta(z\exp(-ct)).$$ □

Appendix B

Feature Based Ill Conditioning

B.1 Models and Datasets

For each model and dataset, we search over $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}\}$ for the learning rate and display the best results. All optimizers use a momentum parameter (β and β_1) of 0.9. Weight decay is not used for any models.

B.1.1 Datasets

We use the following datasets, with summary statistics provided in Table B.1.

Cora & PubMed Both datasets are citation graphs. Each node represents a document, and the node features are constructed from bag-of-words representations of these documents. An edge exists between two nodes if and only if one document cites the other. The task for these datasets is to classify each node into one of the predefined document classes. We use the standard split for training, which consists of twenty samples per class (Yang et al., 2016).

Computers & Photo These datasets are part of the Amazon co-purchase graph (McAuley et al., 2015). Each node represents a product on the Amazon website, and an edge exists between two nodes if the corresponding products were frequently purchased together. Node features are derived from a bag-of-words summary of the reviews for each product. The task is to classify the nodes into different product categories (Shchur et al., 2018). We use a random train/validation/test split with a 60/20/20 ratio for training.

Roman-Empire The dataset is constructed from Wikipedia articles. Each node represents a word, and node embeddings are derived from fastText embeddings (Grave et al., 2018). An edge exists between two nodes if they appear consecutively in the text or are connected in the dependency tree of the sentence. The task is to classify the nodes into their respective grammatical roles within the sentences (Platonov et al., 2023). We use a random train/validation/test split with a 60/20/20 ratio.

Table B.1: Dataset statistics. Standard evaluation metric on all these dataset is Accuracy.

Dataset	Number of Nodes	Number of Edges	Node Features Size	Classes
Cora	2,708	10,556	1,433	7
PubMed	19,717	88,648	500	3
Photo	7,487	238,162	745	8
Computers	13,381	491,722	767	10
Roman-Empire	22,662	32,927	300	18

B.1.2 Models

General GNNs A graph consists of a set of nodes $\mathcal{V}$ and edges $\mathcal{E}$, often denoted as $G = (\mathcal{V}, \mathcal{E})$. Nodes typically have associated features represented by a matrix $\mathbf{X} \in \mathbb{R}^{n \times d_V}$, where n is the number of nodes and d_V is the dimension of node features. Edges connect pairs of nodes, and in some cases, they also have corresponding features represented by a matrix $\mathbf{E} \in \mathbb{R}^{m \times d_E}$, where m is the number of edges and d_E is the dimension of edge features. If for each node v we consider $h_v^{(0)} = x_v$, or the initial embeddings are the node features, each layer of a message passing network can be written as

$$h_v^{(\ell)} = \phi\left(h_v^{(\ell-1)}, \bigoplus_{u \in \mathcal{N}(v)} \psi\left(h_v^{(\ell-1)}, h_u^{(\ell-1)}, e_{uv}\right)\right) \tag{B.1}$$

where $h_v^{(\ell)}$ is the embedding for node v in layer ℓ, ψ is a message function getting the input of the receiver node, sender node, and edge features between them as input. In many variants of the GNNs the input are a subset of these (e.g. only the sender node embeddings). The aggregation function over the neighbors $\bigoplus$ is a simple function such as sum or mean, and ϕ is a node-level update function.

GCN Graph Convolutional Networks (GCN) extend the convolution operation to the graph domain (Kipf and Welling, 2016). A layer of this network can be formulated as:

$$H^{(\ell+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{\ell}\mathbf{W}\right),$$

where $\tilde{A} = A + I$, A is the adjacency matrix and I is the identity matrix, $\tilde{D}$ is the diagonal degree matrix for adjacency matrix $\tilde{A}$, $\mathbf{W}$ is the learnable weight matrix, and σ is an activation function. The formula can be also written as a message-passing function,

$$h_v^{\ell+1} = \sigma\left(\mathbf{W}^{\top} \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{d_v d_u}} h_u^{(\ell)}\right),$$

where $\mathcal{N}(v)$ are the neighbors of v in the graph after adding self-loops and d_v is the degree v.

GAT Graph Attention Networks (GAT) add the attention mechanism to the convolution. Instead of normalizing the neighbors based on their degree, they let the method learn to weight the messages from the neighbors (Veličković et al., 2018). The attention mechanism can be formulated as

$$\alpha_{vu} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{W}h_v \| \mathbf{W}h_u\right]\right)\right)}{\sum_{i\in\mathcal{N}(v)\cup\{v\}}\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{W}h_v \| \mathbf{W}h_i\right]\right)\right)},$$

where $\mathbf{W}$ is a weight matrix shared between the attention mechanism and embedding mappings, $\mathbf{a}$ is a learnable vector and $\|$ is the concatenation operation. The message passing can be formulated as:

$$h_v^{(\ell+1)} = \sigma\left(\sum_{u\in\mathcal{N}(v)\cup\{v\}} \alpha_{uv}\mathbf{W}h_u^{(\ell)}\right).$$

GraphSAGE Instead of performing convolution over all neighbors, GraphSAGE samples a fixed number of neighbors for each node at each convolution operation, hoping to improve the ability to handle inductive setups and help generalize to unseen nodes during training (Hamilton et al., 2017).

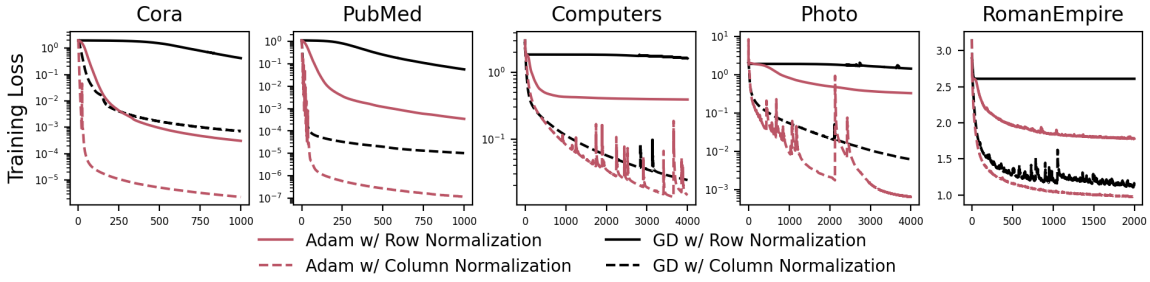
Linear Networks For the linear model, we use one layer of a GCN network, which is equivalent to a logistic regression with a graph convolution over the data with cross entropy loss,

$$\mathbf{Y} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}\mathbf{X}\mathbf{W}\right),$$

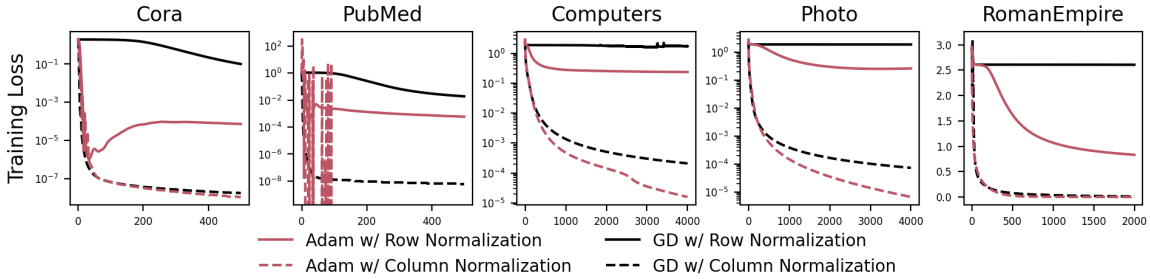
where σ is the softmax function and the graph structure matrices are as described above.

We use the standard PyTorch-Geometric (Fey and Lenssen, 2019) library implementation for GCN, GAT, and GraphSAGE models. In these models we use two layers and a hidden dimension of size 16.

B.2 Results with other GNNs



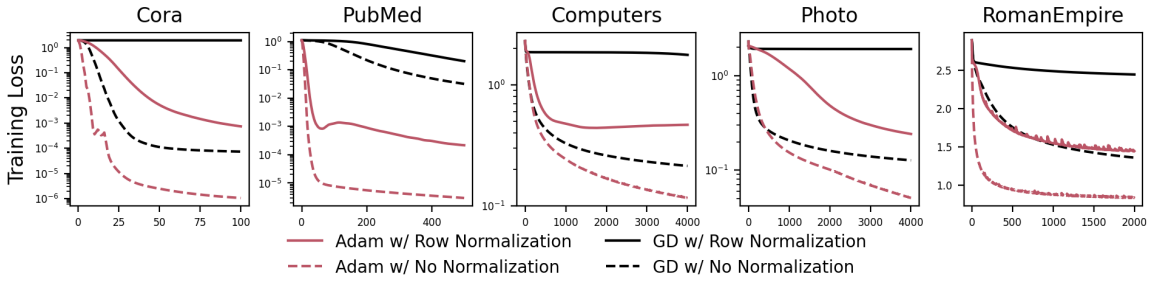
(a) Graph Attention Network



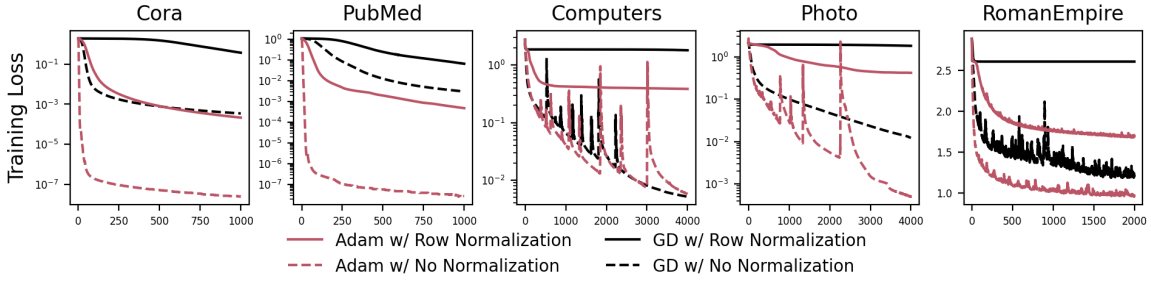
(b) GraphSAGE Network

Figure B.1: **GD struggles on GNNs with standard row normalization, but optimization becomes much easier with columns normalization.** Training with **GD** and **Adam**, using row normalization (solid) and column normalization (dashed).

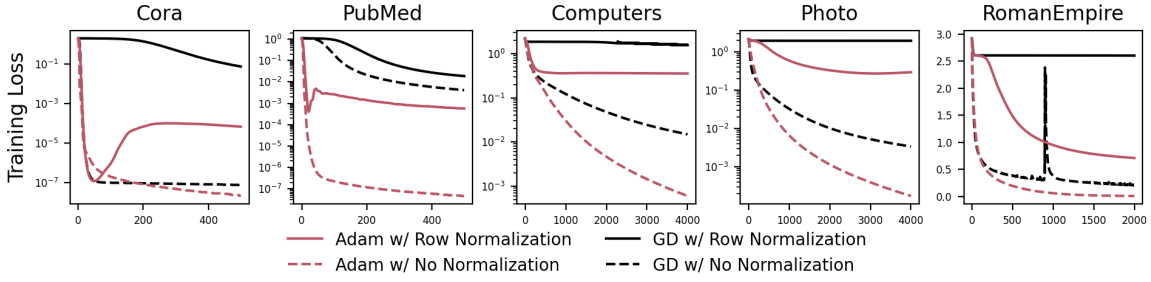
B.3 Row Normalization can be worse than no preprocessing



(a) Graph Convolutional Network



(b) Graph Attention Network



(c) GraphSAGE Network

Figure B.2: **GD performs worse with standard row-normalization than without any normalization.** Training with **GD** and **Adam**, using row normalization (solid) and no normalization (dashed).

Appendix C

Basis Sensitivity

C.1 Sampling Random Rotations in High Dimension

This section explains our method of sampling random rotations for high-dimensional spaces and the implementation details.

C.1.1 High-Dimensional Rotations

Even small modern machine learning models typically have millions of parameters. Consequently, storing a $d \times d$ rotation matrix is often intractable, let alone performing the dot product required to rotate the gradient vector. To address this issue, we sample a $n \times n$ rotation matrix $\mathbf{R}_n$ with $n \ll d$ uniformly (in the sense of the Haar measure) from the special orthogonal group $SO(n)$, and a random permutation π of $0, \ldots, d-1$. For now, we assume $\frac{d}{n} \in \mathbb{N}$, see Appendix C.1.3 for a general case. To rotate a gradient g, we compute:

$$g^{(\mathbf{R}_n, \pi)} := \pi^{-1} \circ \left(\left[\bigoplus_{i=1}^{d/n} R_n \right] (\pi \circ g) \right), \tag{C.1}$$

$$= \pi^{-1} \circ \begin{bmatrix} R_n & & & \\ & R_n & & 0 \\ & & R_n & \\ & 0 & & \ddots \\ & & & & R_n \end{bmatrix} (\pi \circ g), \tag{C.2}$$

where $\bigoplus$ denotes the direct sum operation, producing a block-diagonal matrix with d/n blocks R_n. This procedure effectively computes a rotation by blocks of size n picked from a random partition of indices, constituting a valid rotation.

Intuitively, if n is sufficiently large, we expect this procedure to approximate well the effect of random rotations sampled uniformly from $SO(d)$, due to the law of large numbers homogenizing geometric properties across coordinates. To confirm this intuition, we perform an ablation study in Figure C.1, finding that the impact on Adam's performance saturates well below our operational values.

Our approximation reduces the memory cost from $O(d^2)$ to $O(n^2+d)$, and the computational cost from $O(d^2)$ to $O(nd)$. Since batch matrix multiplications required for the rotation can be performed efficiently on modern GPUs, the final overhead of applying rotations is extremely small.

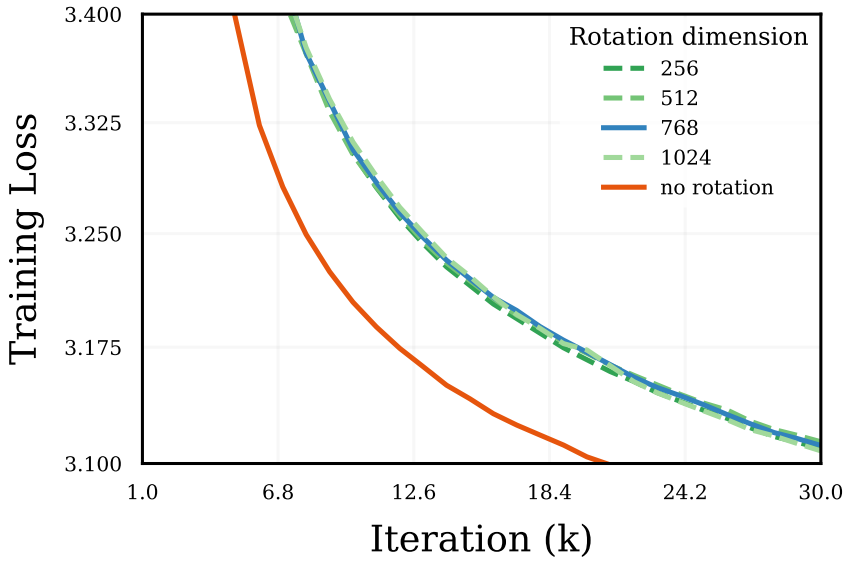


Figure C.1: Training loss of GPT-2 when training with different rotation dimension n. The loss of performance is consistent across n at our range.

C.1.2 Reflections and Sampling From The Haar Measure

To sample R_n uniformly from $SO(n)$ with respect to the Haar measure, we employ the QR decomposition trick (Mezzadri, 2007; Ozols, 2009), which samples from the Haar measure μ of the orthogonal group $O(n)$. Let us consider the projection $\pi : O(n) \to SO(n)$, such that $\pi(\mathbf{R})$ is $\mathbf{R}$ when $\mathbf{R} \in SO(n)$, and $\pi(\mathbf{R})$ simply multiplies the first column of $\mathbf{R}$ by -1 when $\mathbf{R} \in O(n) \setminus SO(n)$. The push forward of μ by π is the Haar measure on $SO(n)$. Since Adam is reflection equivariant, rotating with $\pi(\mathbf{R})$ and with $\mathbf{R}$ will lead to identical performance for any $\mathbf{R} \in O(n)$. Thus we can omit to apply π, and simply sample from μ using the QR decomposition method.

Similarly, Adam is permutation equivariant, thus we omit to apply the inverse permutation before providing the rotated gradients to Adam, and to apply the permutation before rotating the update, as removing these two steps do not affect performances.

C.1.3 Rotation Residual

Based on the type of rotation and the chosen dimension n, the number of blocks may not divide evenly, i.e., $\frac{d}{n} \notin \mathbb{N}$. To address this issue, we introduce an additional rotation

matrix, which we refer to as the *residual* matrix, to complete the missing dimensions. More formally, let d represent the dimensionality of the parameter space, and let n denote the block dimensions of the rotation. We define $b \triangleq \lfloor \frac{d}{n} \rfloor$ as the number of complete blocks. The **residual** matrix $\mathcal{R}$ is then sampled from $SO(p)$, where $p \triangleq d - nb$. Therefore, Equation (C.1) becomes

$$g^{(\mathbf{R}_n, \mathcal{R}, \pi)} := \pi^{-1} \circ \left([B \oplus \mathcal{R}] \left(\pi \circ g \right) \right), \tag{C.3}$$

$$\tag{C.4}$$

$$= \pi^{-1} \circ \begin{bmatrix} B & \\ & \mathcal{R} \end{bmatrix} (\pi \circ g), \tag{C.5}$$

$$= \pi^{-1} \circ \begin{bmatrix} R_n & & & & \\ & R_n & & 0 & \\ & & \ddots & & \\ & 0 & & R_n & \\ & & & & \mathcal{R} \end{bmatrix} (\pi \circ g). \tag{C.6}$$

where $B = \bigoplus_{i=1}^{b} R_n$.

C.1.4 Overall Validation and Impact of Flash Attention

In Figure C.2, we present the training loss when training GPT-2 with SGD without rotations, with global random rotations using flash attention, and with global random rotations without flash attention. In particular, we confirm two important observations:

- Without flash attention (the setting we use for our experiments) the performances of SGD under global random rotation and under no rotations are identical. This validates that our experimental setting is behaving as expected.
- When we use flash attention with rotations, we observe a slight difference in performance. As explained in Section 4.2.2, this is due to flash attention amplifying numerical errors from the application of the rotation. Interestingly, likely due to a slight regularization effect, it it increases training performance.

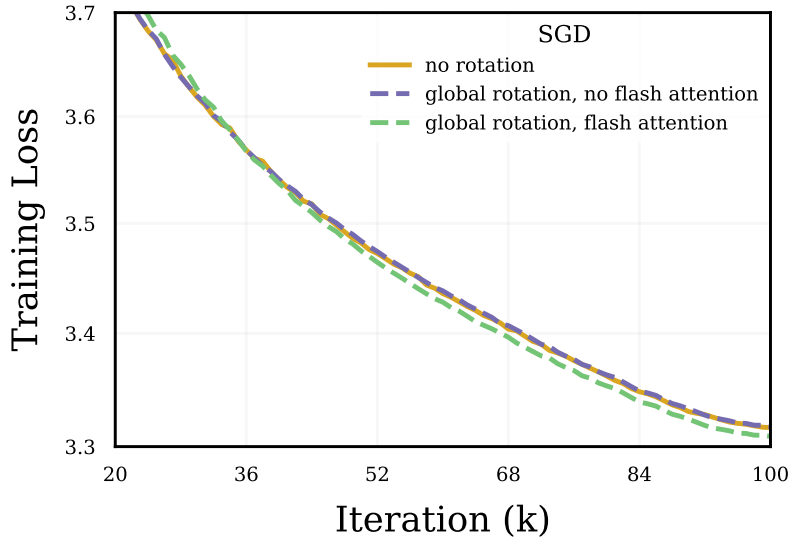


Figure C.2: SGD performance when applying global random rotations, with and without FlashAttention.

C.2 Experimental Details

This section provides additional details about the hyperparameters used for the architecture mentioned in the paper, as well as their optimizer and rotations.

C.2.1 Rotations Design Choices

By default, for random rotations we fix the dimension of our rotation matrix $\mathbf{R}_n$ at 768 (which is the hidden dimension and thus makes residual rotations unnecessary for most rotation types). The matrix is sampled at the start of training, remains fixed throughout, and is shared across blocks the entire training process.

SVD Rotation. Following (Zhao et al., 2024a), this is the only rotation that is dynamic rather than static. Specifically, we compute the full-rank SVD decomposition of the gradient for each layer every 250 steps (recommended frequency in (Zhao et al., 2024a)).

Rotation in Transformers. By default, many implementations store the query, key, and value parameters within a single linear layer. Thus, we split them to treat them as separate layers, reflecting the fundamental differences in how their parameters are involved in forward computations. Additionally, PyTorch stores parameters as tensors in the shape `(output_dim, input_dim)`, but embeddings are stored as lookup tables in the shape `(input_dim, output_dim)`. For output neuron and input neuron rotations to behave intuitively, we thus transpose embedding layers before and after rotations.

C.2.2 Architectures

GPT2 (Transformer). We trained a GPT-2 model with 124M parameters on the OpenWebText dataset (Gokaslan and Cohen, 2019) using a configuration designed for efficient pretraining. The model architecture includes 12 layers, 12 attention heads, and a 768-dimensional embedding space, with no bias in LayerNorm or Linear layers. We employed the AdamW optimizer with a peak learning rate of 6e-4, $\beta_1 = 0.9$, $\beta_2 = 0.95$, and a weight decay of 1e-1, applying gradient clipping of 1.0. Training ran for 100,000 iterations (or 30,000 for some smaller ablations), with learning cosine rate decay starting after a 2,000-iteration warm-up, decaying to a minimum of 6e-5. We used micro batch size of 12 with gradient accumulation steps to simulate an effective batch size of 480. All experiments were performed on four A100 80GB GPUs, leveraging mixed precision. Unless otherwise specified, all optimizer hyperparameters were shared across experiments and set to the default values specified in Karpathy (2022).

ViT (Vision Transformer). We trained a Vision Transformer (ViT) model on the ImageNet-1K dataset (Deng et al., 2009) using the SimpleViT architecture (Beyer et al., 2022). The model consists of 12 layers, 6 attention heads, a hidden dimension of 384, and an MLP dimension of 1536, with a patch size of 16 and input image size of 224. The AdamW

optimizer was employed with a learning rate of 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e-8$, and a weight decay of 0.1. We used a cosine learning rate schedule with 5 warm-up epochs. The training was conducted for 100 epochs with a batch size of 1024. All experiments were performed with mixed precision.

ResNet-50 (CNN). We trained a ResNet-50 model (He et al., 2016) on the ImageNet-1K dataset (Deng et al., 2009) using the AdamW optimizer. The optimizer was configured with a learning rate of 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e-8$, and a weight decay of 0.0001. We employed a cosine learning rate schedule with 5 warm-up epochs. The training ran for 100 epochs with a batch size of 256.

C.2.3 Assumptions Estimation

We now outline how we computed empirical estimations of assumptions in Section 4.4.

L_∞-bounded gradient. Algorithm 3 describes the process we use to estimate the bound constant $\tilde{C}$ of stochastic gradients under L_∞ norm, as detailed in Section 4.4.1.

Algorithm 3 Empirical Gradient Bound Estimation for Adam

Require: T: total number of iterations (1000)
Require: $w_{\mathbf{R}'}$: last checkpoint obtained by running Adam under rotation $\mathbf{R}'$
Initialize $\tilde{C} \leftarrow 0$ ▷ Maximum infinity norm of gradients
for $t \leftarrow 1$ **to** T **do**
  Sample a minibatch B_i ▷ Select one minibatch
  $g_{B_i} \leftarrow \nabla f_{B_i}^{(\mathbf{R})}(w_{\mathbf{R}'})$ ▷ Compute gradient for minibatch
  $\tilde{C}' \leftarrow \|g_{B_i}\|_\infty$ ▷ Compute infinity norm of the gradient
  $\tilde{C} \leftarrow \max(\tilde{C}, \tilde{C}')$ ▷ Update the maximum gradient bound
end for
return $\tilde{C}$ ▷ Return the estimated gradient bound

$(1,1)$-Norm. Using the Hessian rows sampled from GPT-2 checkpoints that were trained under various rotations in Section 4.4.1, we estimate $\frac{\|H\|_{(1,1)}}{d}$ by averaging the L_1 norm of sampled rows. While this could induce a large variance from the sampling of rows, we find that variations of the L_1 norms from rotations are fairly homogeneous across rows.

C.3 Additional Results

C.3.1 Main Experiments

We provide additional results from our main line of experiments.

ViT/S (ImageNet). Figure C.3 extends the results from Figure 4.1(b) with validation loss and accuracy

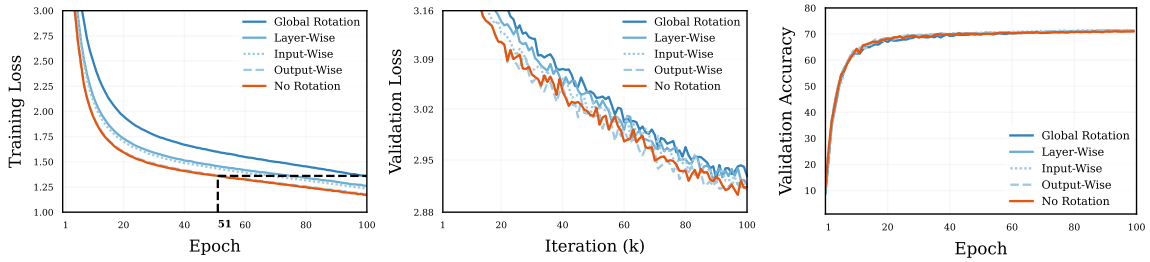


Figure C.3: SimpleViT - ImageNet training loss, validation loss and top-1 validation accuracy

ResNet50 (ImageNet). Figure C.4 demonstrates that Adam maintains its performance well under rotational transformations for ResNets. This robustness to rotation implies that Adam gains little advantage from the standard basis structure in this setting. This finding aligns with the fact that SGD with extensive tuning can outperform Adam when training these networks.

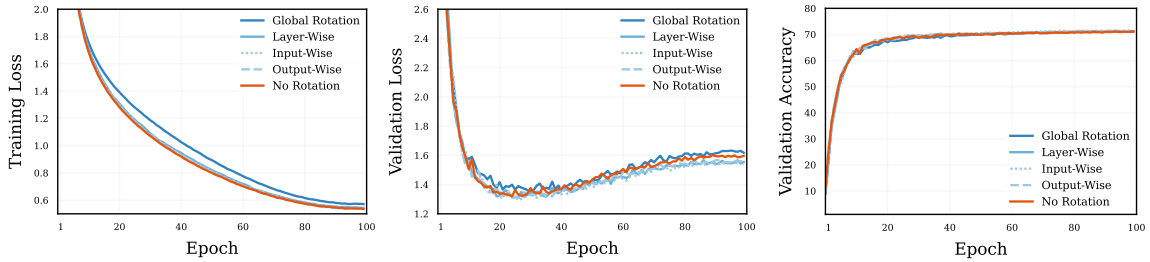


Figure C.4: Training loss, validation loss and top 1 % validation accuracy, when training a ResNet-50 with Adam on ImageNet across different scopes of rotations.

C.3.2 Architecture Aware Rotation.

We seek to identify whether certain transformer layer types are more sensitive to rotations than other, and contribute more to the overall performance degradation observed in Figure 4.1(a) when using layer-wise rotations.

Figure C.5 shows the loss curves when rotating only one layer type at a time. We find that the performance degradation induced by layer-wise rotations is small for most layer types, seemingly balanced across these layers, with the exception of value and embedding layers.

Layerwise rotation of value layers seem to impact the loss more noticeably than with other layer types. In Figure C.7, we find that reducing the scope of rotations to output neuron wise does not improve the performance when rotating value layers.

The biggest drop of performances is observed for embedding layers, which we conjecture to be linked to the discrepancy in frequency across tokens. Figure C.6 shows indeed that when rotating the embedding layer by output neuron (i.e., within weights corresponding to a same token) the degradation becomes unnoticeable.

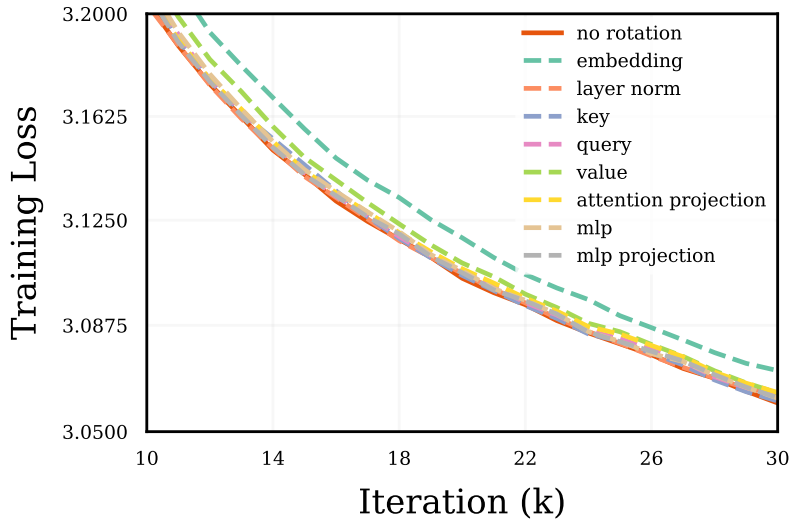


Figure C.5: Layer-wise rotation applied to only specific layer types

C.3.3 Hessian Rows

We use the end checkpoint of GPT-2 to sample rows from the Hessian in different rotated parameter spaces (see Section 4.4.1).

From Figure C.8 to Figure C.14, we present the same figure as in Figure 4.9, but for rows taken from different layer types, confirming that the behavior we observed is consistent across parameter types. Except for embeddings, rows are always taken from the second Transformer block.

Figure C.15 shows a row in the attention projection layer of the 8-th transformer block, showing our observations seem also consistent across depth.

Figure C.16 uses checkpoints trained with the same rotations as the one applied to the

Hessian. We find the same behavior for no rotations, global and output-wise, but we find that with the SVD-rotated checkpoints, there is increased variance in the Hessian values outside of the layer.

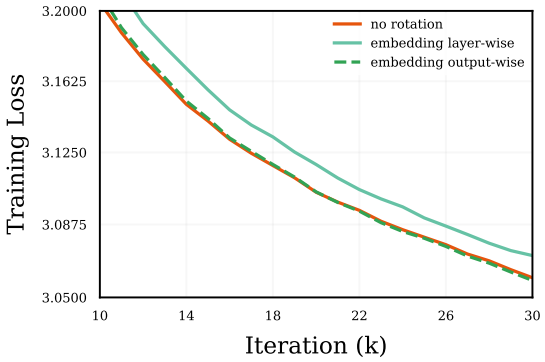


Figure C.6: Layer-wise rotation and output-wise rotation on embedding layers only

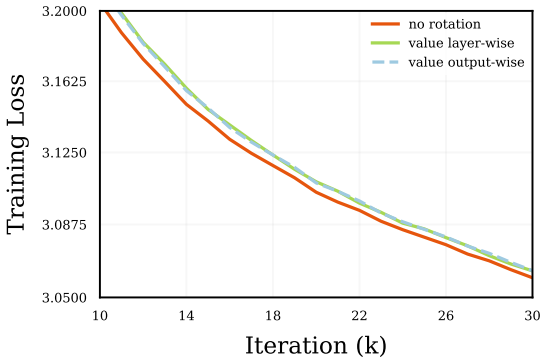


Figure C.7: Layer-wise rotation and output-wise rotation on attention values only

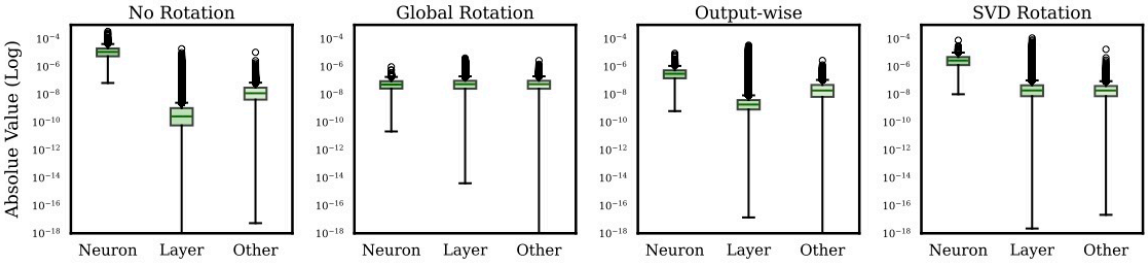


Figure C.8: Hessian value distribution of a row in embedding layer.

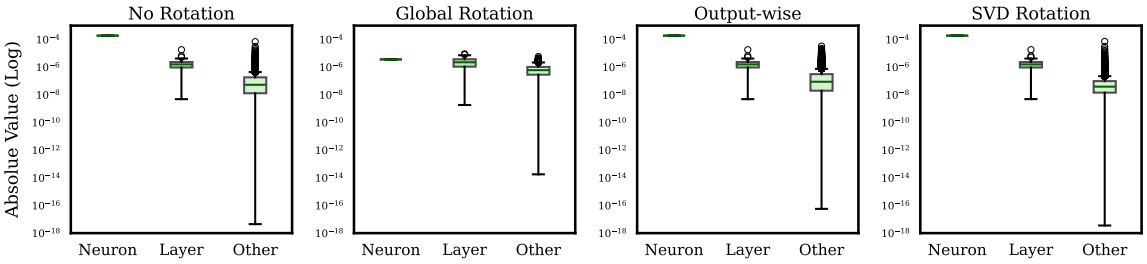


Figure C.9: Hessian value distribution of a row in second Transformer layer norm layer.

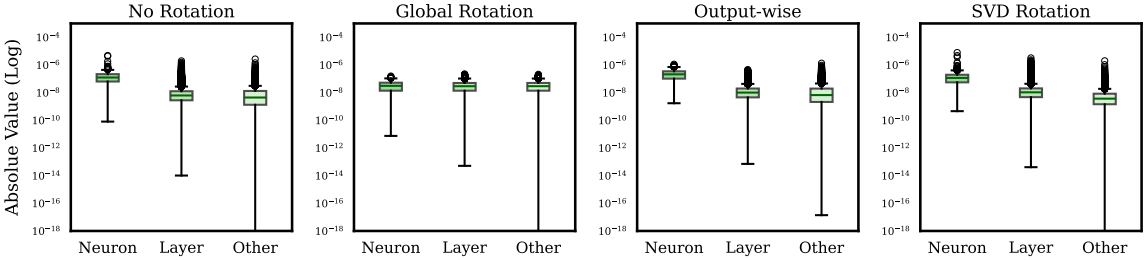


Figure C.10: Hessian value distribution of a row in second Transformer key layer.

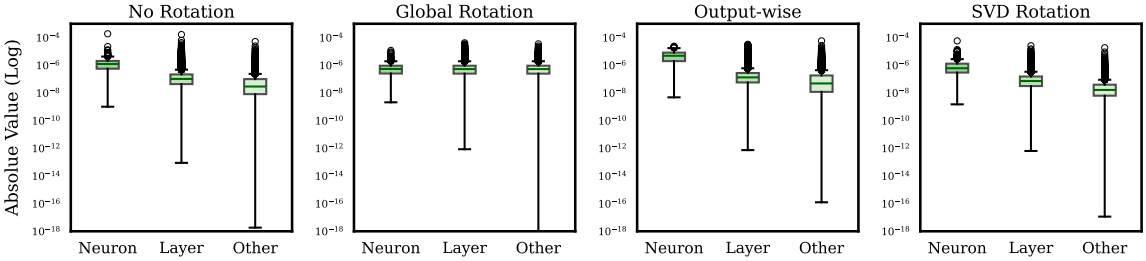


Figure C.11: Hessian value distribution of a row in second Transformer query layer.

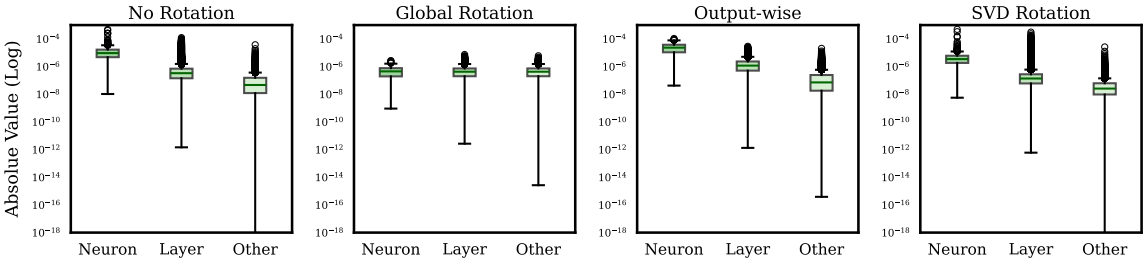


Figure C.12: Hessian value distribution of a row in second Transformer value layer.

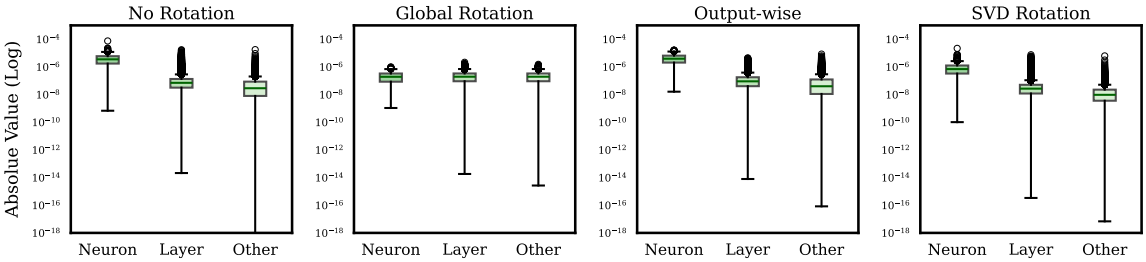


Figure C.13: Hessian value distribution of a row in second Transformer mlp layer.

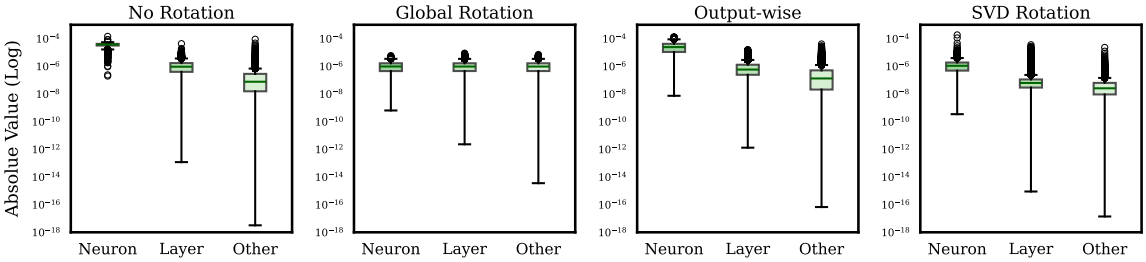


Figure C.14: Hessian value distribution of a row in second Transformer mlp projection layer.

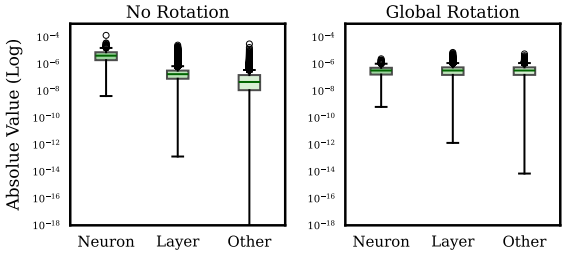


Figure C.15: Hessian value distribution of a row in eighth Transformer attention projection layer.

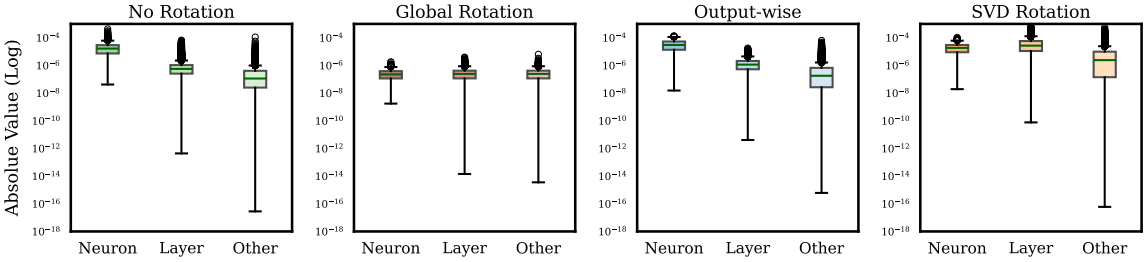


Figure C.16: Hessian value distribution of a row in second Transformer attention projection layer from checkpoints that are *trained with different rotations*.

C.3.4 Update orthogonality

We aim to measure the orthogonality of the update. Since weight decay is applied directly in parameter space rather than to the gradient, the update rule (ignoring the bias corrections) becomes:

$$W_{t+1} - W_t = -\alpha_t R^\top \frac{M_t}{\sqrt{V_t} + \epsilon} - \alpha_t \lambda W_t, \tag{C.7}$$

where

$$\begin{aligned} M_t &= \beta_1 M_{t-1} + (1-\beta_1) R\nabla\mathcal{L}(W), \\ V_t &= \beta_2 V_{t-1} + (1-\beta_2)\left(R\nabla\mathcal{L}(W) \circ R\nabla\mathcal{L}(W)\right). \end{aligned}$$

To isolate the effective update direction, we rewrite the expression as:

$$\frac{W_{t+1} - (1-\alpha_t\lambda)W_t}{-\alpha_t} = R^\top \frac{M_t}{\sqrt{V_t} + \epsilon} := A. \tag{C.8}$$

Coefficient of variation. Suppose matrix A has singular values $\{s_i\}$. For a scale-invariant measure of orthogonality, we minimize the normalized root mean square deviation between singular values and a constant,

$$\min_\alpha \frac{1}{\mu}\sqrt{\frac{1}{n}\sum_i (s_i - \alpha)^2},$$

where $\mu_s = \frac{1}{n}\sum_i s_i$ is the mean of $\{s_i\}$. The normalization facilitates the comparison between data with different scales. The solution for this objective is the *coefficient of variation (CV)*, defined as

$$\mathrm{CV}(s_i) = \frac{\sigma_s}{\mu_s},$$

where σ_s is the standard deviation of the singular values. CV captures the *relative dispersion* of the singular values and is invariant to uniform rescaling of A. This makes it especially suitable for comparing updates or transformations that differ in magnitude but share underlying structure. Since an orthogonal matrix has all singular values equal to 1, a lower CV indicates that the singular values are more tightly clustered, suggesting that A is closer to being orthogonal up to a global scaling.

In Figure 4.11, we provided a mean variation of the singular values in the update for each layer type, averaged across depth of the network using GPT-2. We now expand on this and show the variation for each individual weight matrix in the network. We display the coefficient of variation of the singular values of each linear layer's update. Each of the GPT-2 model's twelve encoder layer has six linear layers, four in the attention layer after splitting (to compute Q, K, V and the output projection O) and two in the MLP block (labeled W_1 and W_2). We compute the SVD of each update, and compute the coefficient of variation of

the singular values and show the results in Figure C.17. For completeness, we also show this result at the beginning and end of training in Figure C.18 and Figure C.19 respectively. The trend is more clear and steady at the end of training.

Other metrics. In our experimentation, we considered other metrics to measure orthogonality, with each finding similar results.

One alternative method is to measure how far the scaled singular values are from 1:

$$\frac{1}{n}\sum_i (\alpha s_i - 1)^2$$

where α is a scaling factor shared across the singular values s_i of the update. For a set of singular values $\{s_1, \ldots, s_n\}$, the optimal scaling factor is

$$\alpha^* = \sum_{i=1}^{n} \frac{s_i}{\sum_{i=1}^n s_i^2}.$$

We display measure at the start, middle, and end of training in figures Figure C.20 respectively. We see that note that is very strongly correlates with the coefficient of variation of $\{s_i\}$.

Similarly, another measure of semi-orthogonality is how close $AA^\top$ is to the identity matrix I. The objective being

$$\min_\alpha \frac{1}{\mu}\sqrt{\frac{1}{n}\|AA^\top - \alpha I\|_F^2} = \frac{\sigma_\lambda}{\mu_\lambda} = \mathrm{CV}(\lambda),$$

where $\lambda = s_i^2$ are the eigenvalues of $AA^\top$, and σ_λ and μ_λ are the standard deviation and mean of λ.

To see this, since $AA^\top$ is symmetric, so it has an eigendecomposition by the Spectral theorem, i.e. $AA^\top = Q\Lambda Q^{-1}$ where Q is orthogonal matrix. Then,

$$\begin{aligned} Q\Lambda Q^{-1} - \alpha I \\ Q\Lambda Q^{-1} - \alpha Q Q^{-1} \\ Q\Lambda Q^{-1} - \alpha Q Q^{-1} \\ Q(\Lambda - \alpha I)Q^{-1} \end{aligned}$$

Then, using the circulant property of the trace, we have

$$
\begin{aligned}
\|Q(\Lambda-\alpha I)Q^{-1}\|_F^2 &= \text{trace}\left((Q(\Lambda-\alpha I)Q^{-1})^\top Q(\Lambda-\alpha I)Q^{-1}\right) \\
&= \text{trace}\left(Q(\Lambda-\alpha I)^\top Q^{-1}Q(\Lambda-\alpha I)Q^{-1}\right) \\
&= \text{trace}\left(Q(\Lambda-\alpha I)^\top(\Lambda-\alpha I)Q^{-1}\right) \\
&= \text{trace}\left(Q^{-1}Q(\Lambda-\alpha I)^\top(\Lambda-\alpha I)\right) \\
&= \text{trace}\left((\Lambda-\alpha I)^\top(\Lambda-\alpha I)\right) \\
&= \sum_{i=1}^{n}(\lambda_i-\alpha)^2 \\
&= n\sigma_\lambda^2 \qquad \text{Substituting optimal } \alpha^*
\end{aligned}
$$

The optimal α^* in this case is μ_λ. The normalization by mean assures scale invariance, and yields coefficient of variation of λ. We show this metric in Figure C.21.

Instead, if the scaling factor is used on the symmetric matrix $AA^\top$, we have the objective

$$
\min_\alpha \sqrt{\frac{1}{n}\|\alpha AA^\top - I\|_F^2},
$$

which is already scale invariant and yield the optimal scaling factor $\alpha^* = \sum_{i=1}^n \frac{\lambda_i}{\sum_{i=1}^n \lambda_i^2}$. We also plot this metric in Figure C.22.

Additionally, Liu and Su (2025) use a metric inspired by the signal processing literature called the SVD entropy to study Muon updates, which is defined as follows.

$$
H(s) = -\frac{1}{\log(n)}\sum_{i=1}^{n}\frac{s_i^2}{\sum_{j=1}^n s_j^2}\log\left(\frac{s_i^2}{\sum_{j=1}^n s_j^2}\right)
$$

We also computed this metric, and it again strongly correlated with the coefficient of variation. We display measure at the start, middle, and end of training in figure Figure C.23 and note that larger is better for this metric, in contrast to the others we consider. While all metrics showed roughly the same trends, we chose to focus on coefficient of variation of singular values as we felt it was the simplest.

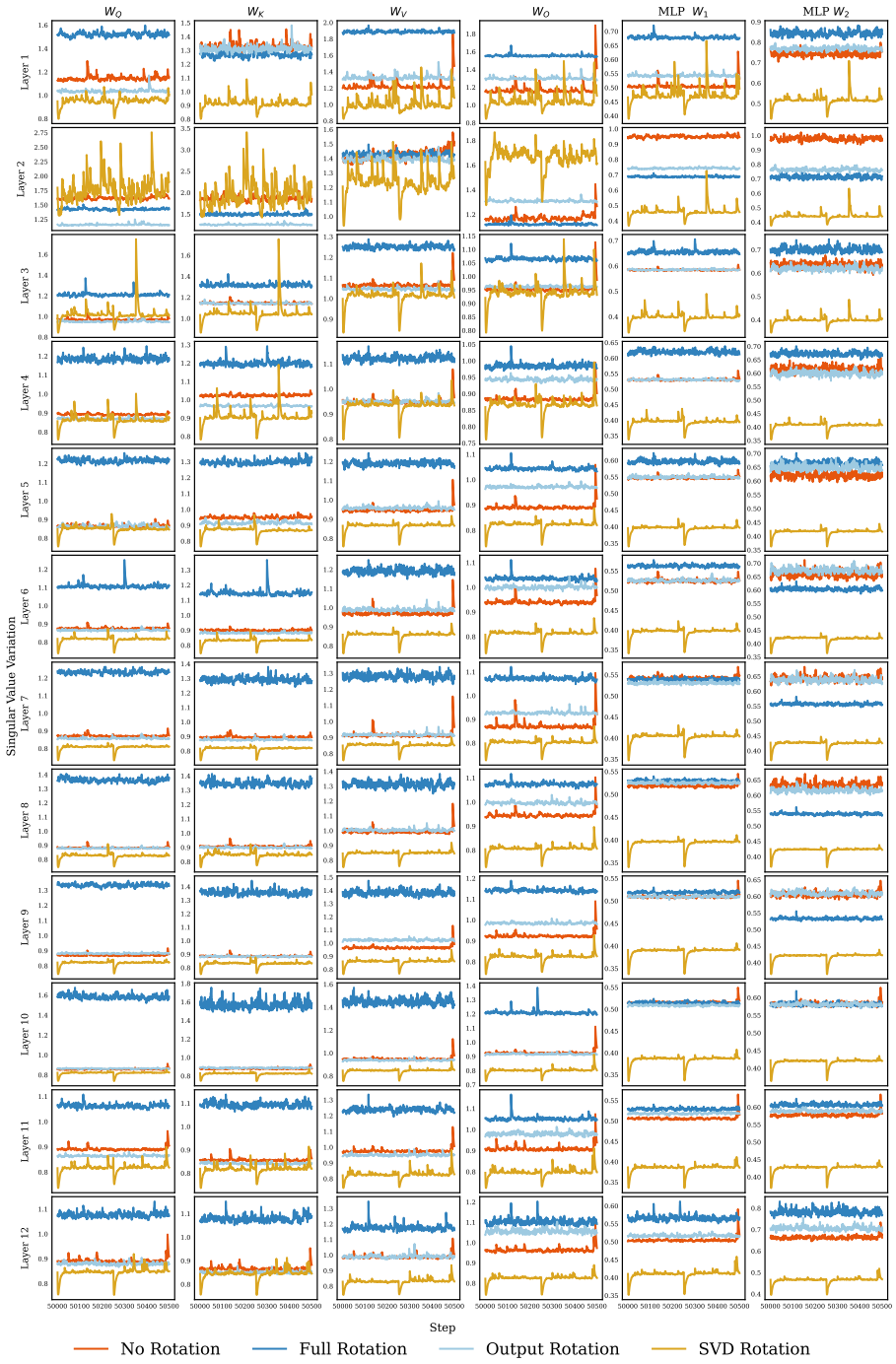


Figure C.17: Singular value variation during training. We measure the variation of the singular values of each update of AdamW under various rotations for every linear layer in each GPT2 encoder block. While not universal, we find that a majority of the time, the SVD rotations leads to the lowest variation, while the global rotations leads to the highest. Additionally, we can see that recomputing the SVD matrices (at 50000 and 50250 steps) leads to a downward spike in the variation.

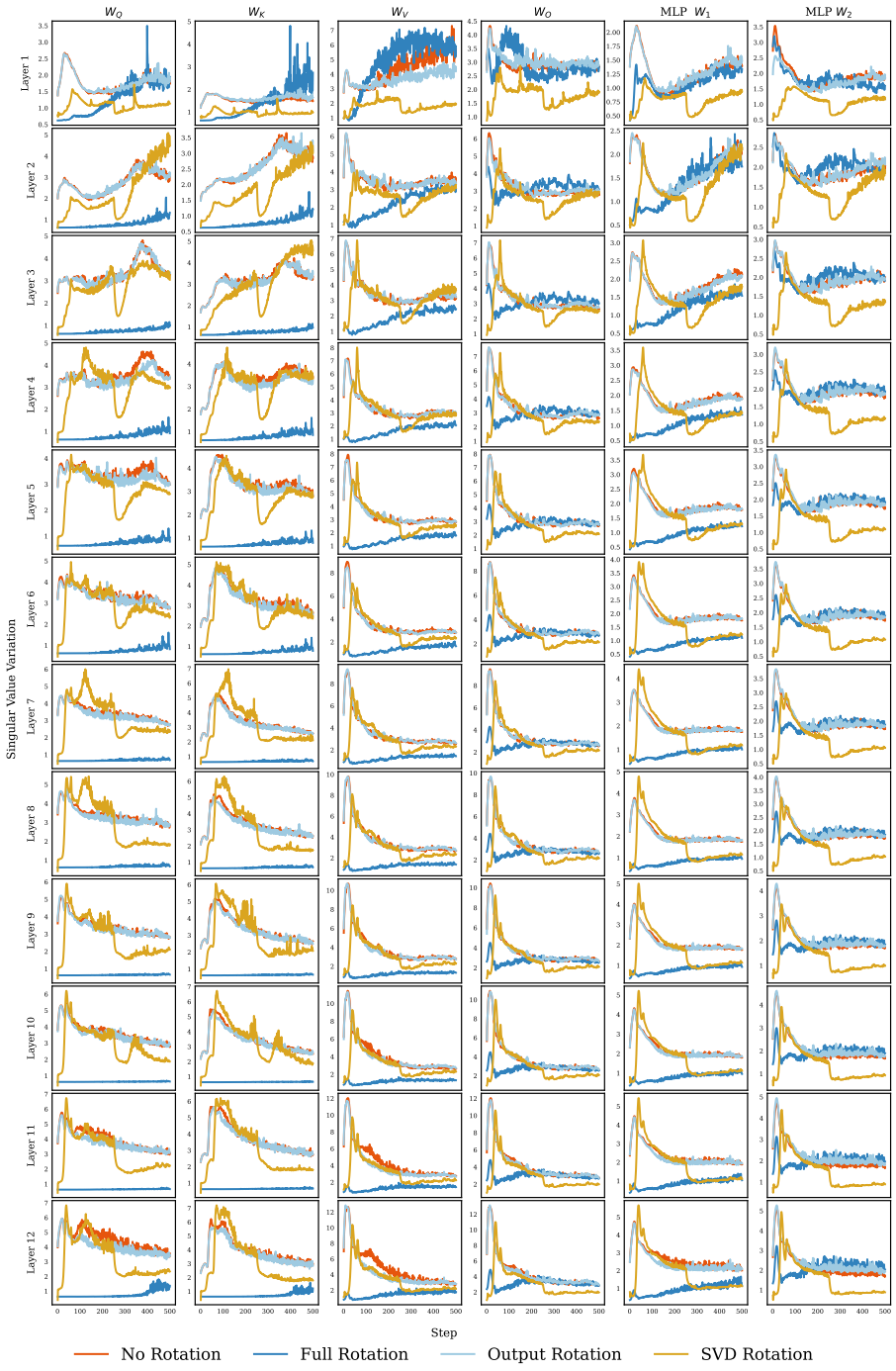


Figure C.18: Singular value variation at the beginning of training. While the variation increases at initialization, we see it begin to trend downwards, notably after the SVD is recomputed.

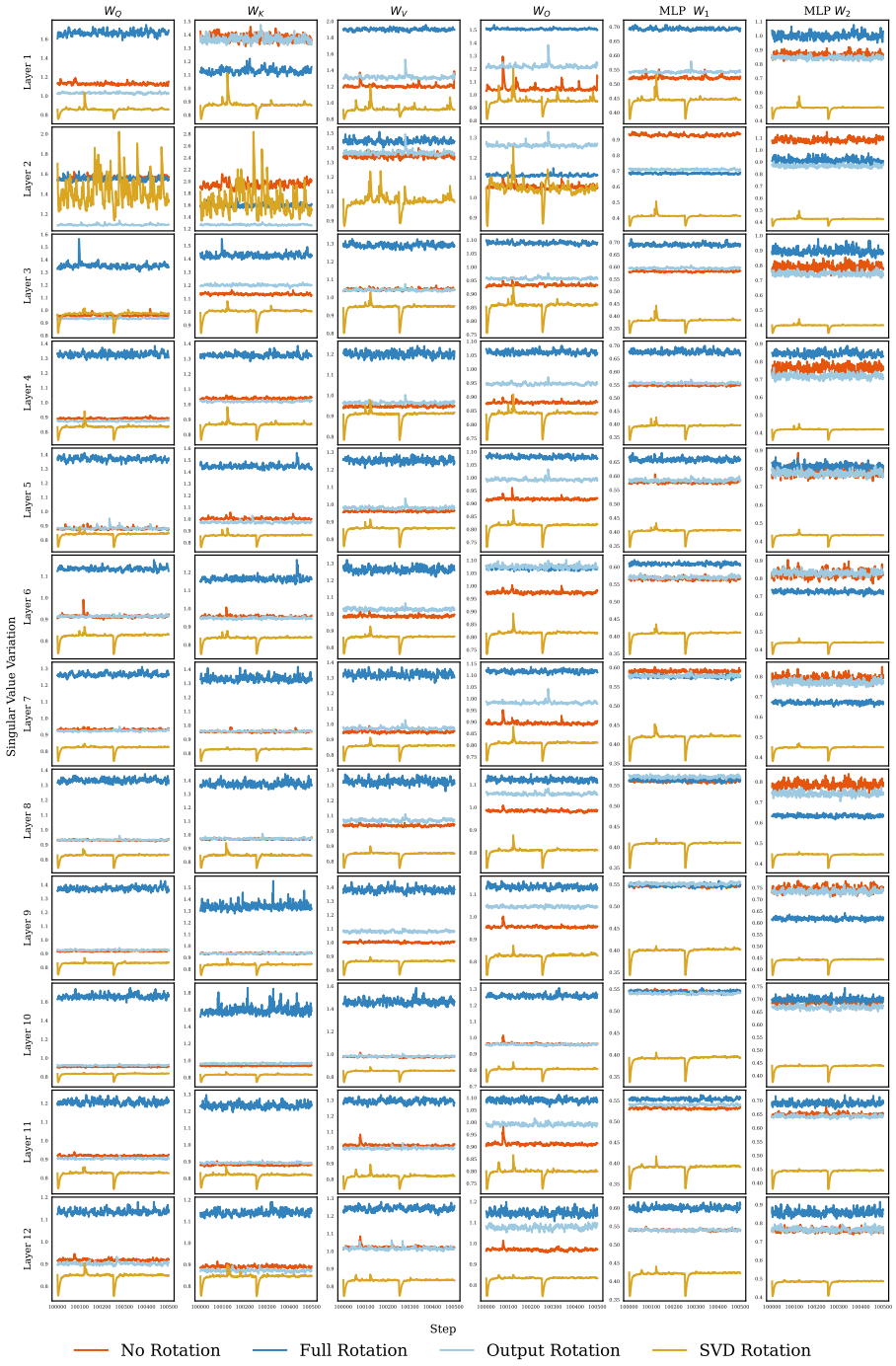


Figure C.19: Singular value variation at the end of training. With the exception of a few layers, we see the variation hold relatively stably aside from when the SVD is recomputed.

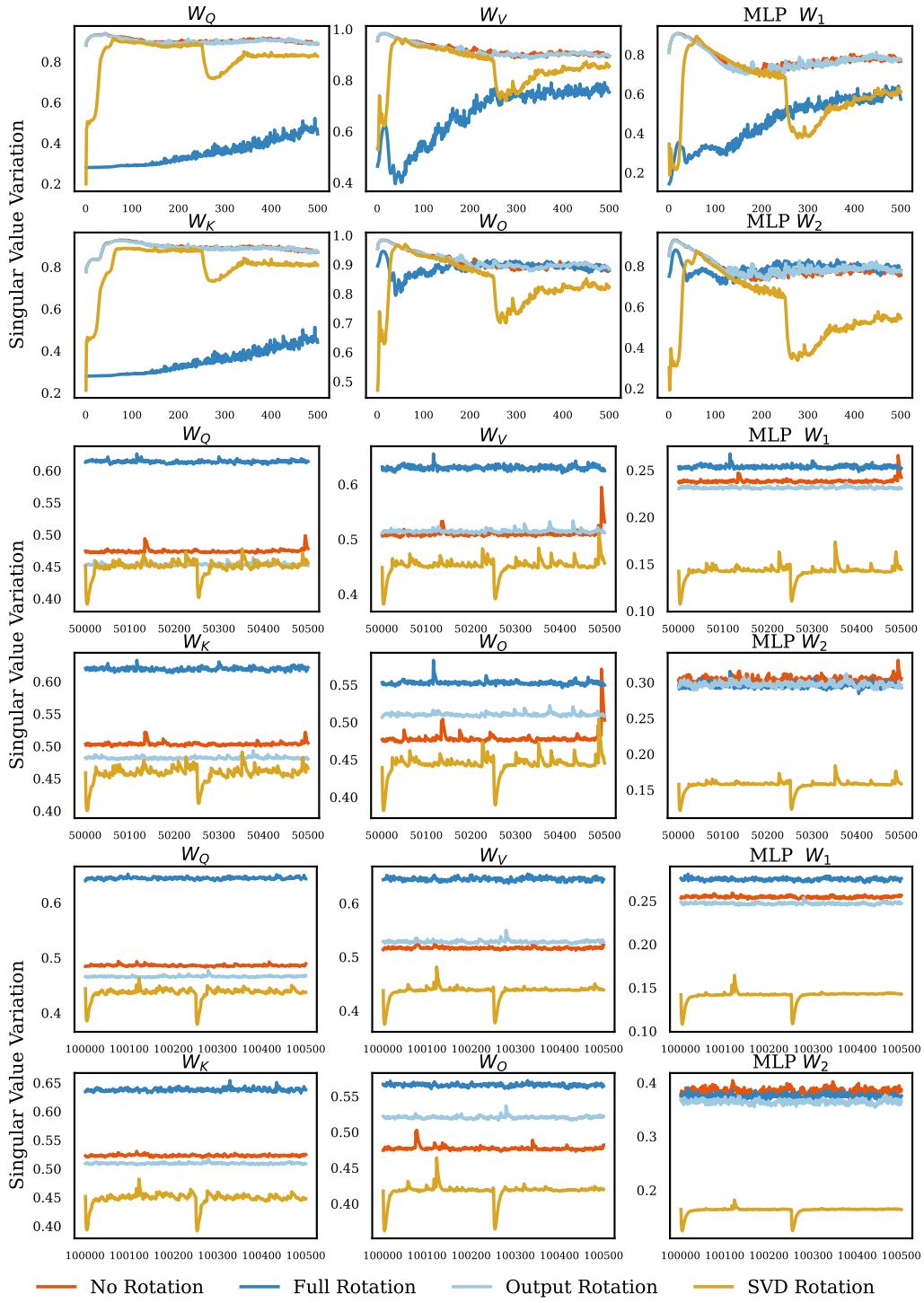


Figure C.20: An alternative variation metric for singular values described in Appendix C.3.4 throughout training, averaged over network depth. We see similar results to the coefficient of variation.

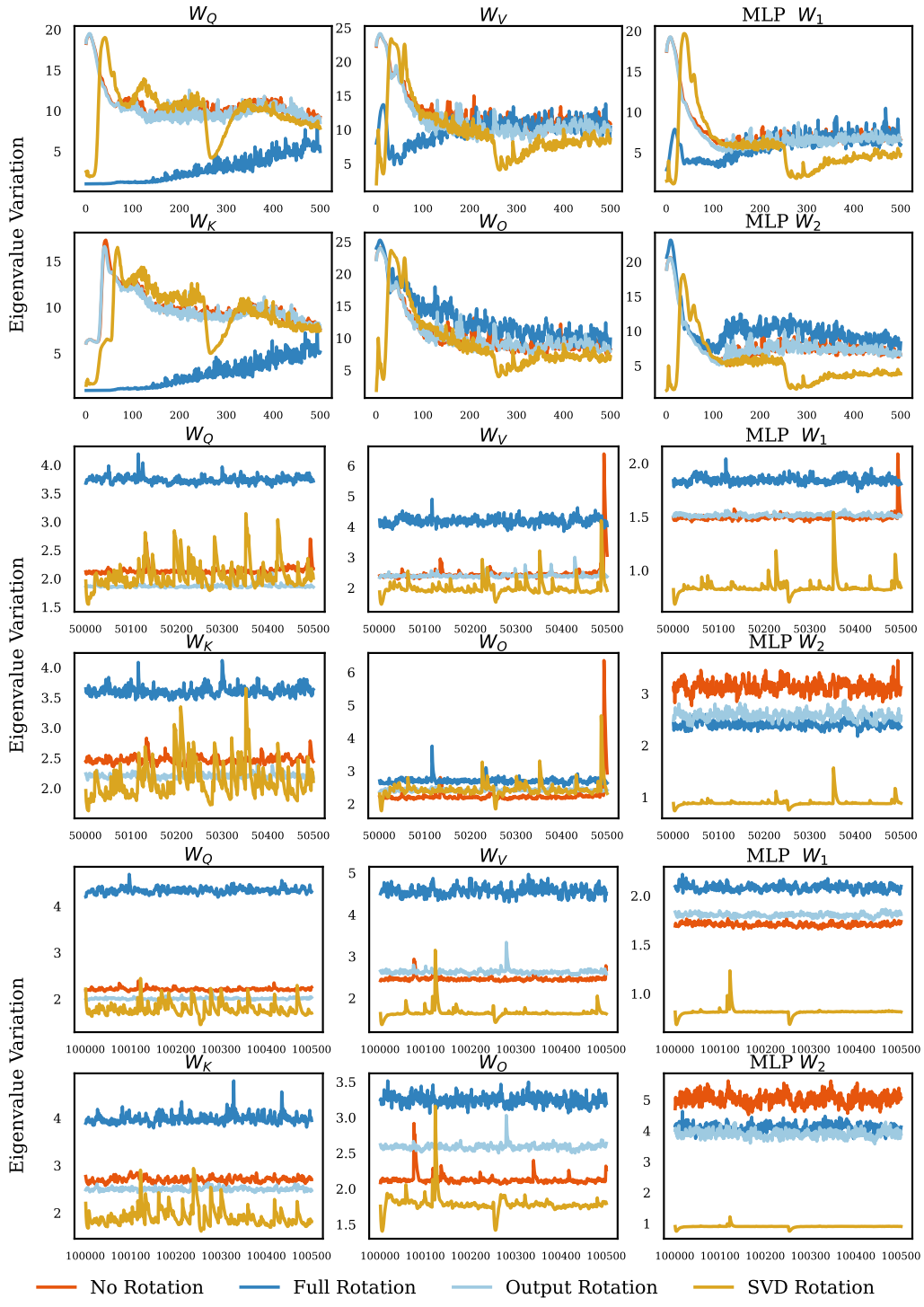


Figure C.21: The coefficient of variation of the eigenvalues described in Appendix C.3.4 throughout training, averaged over network depth. We see similar results to the coefficient of variation.

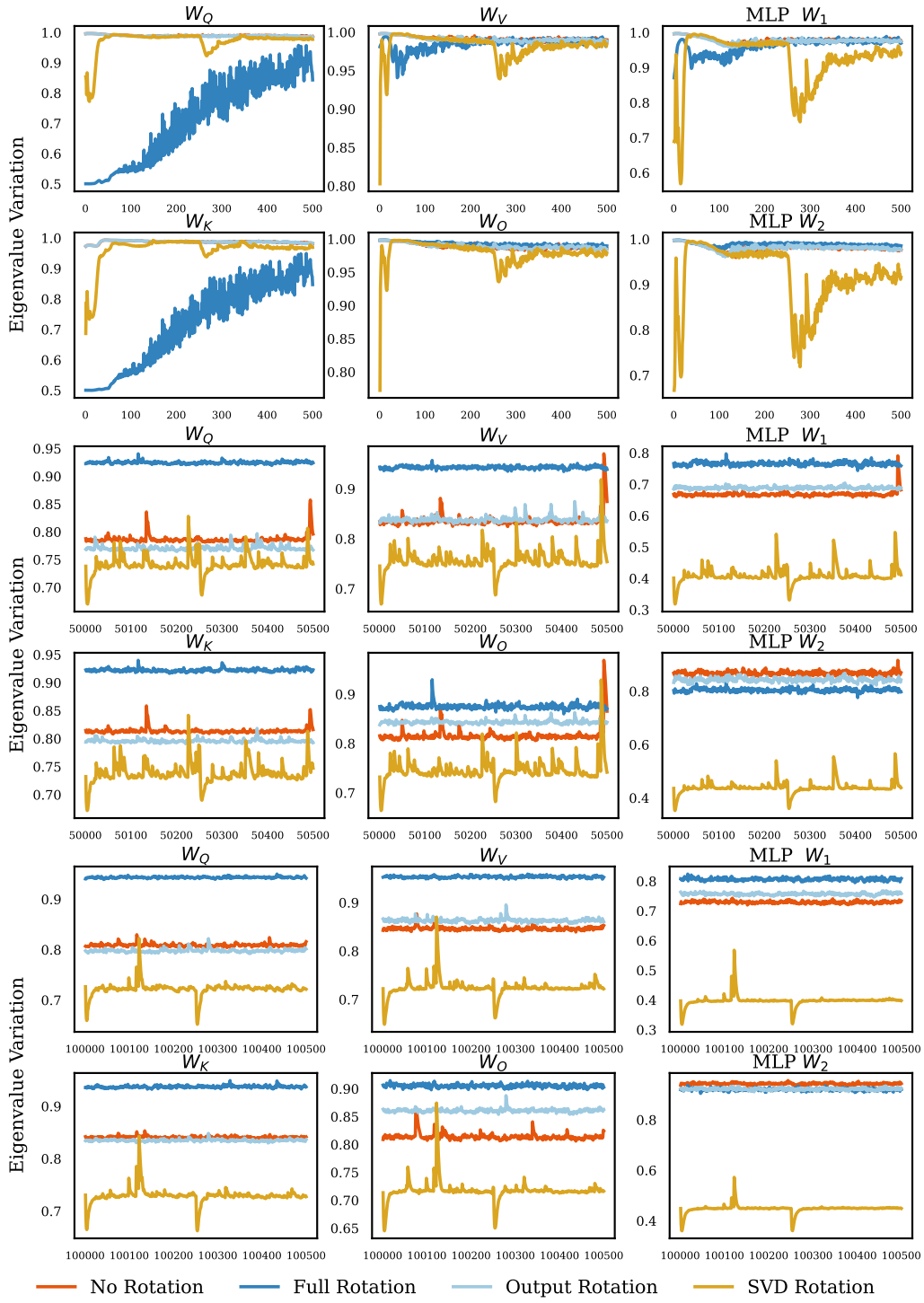


Figure C.22: The variation eigenvalues of the scaled $AA^\top$ described in Appendix C.3.4 throughout training, averaged over network depth. We see similar results to the coefficient of variation.

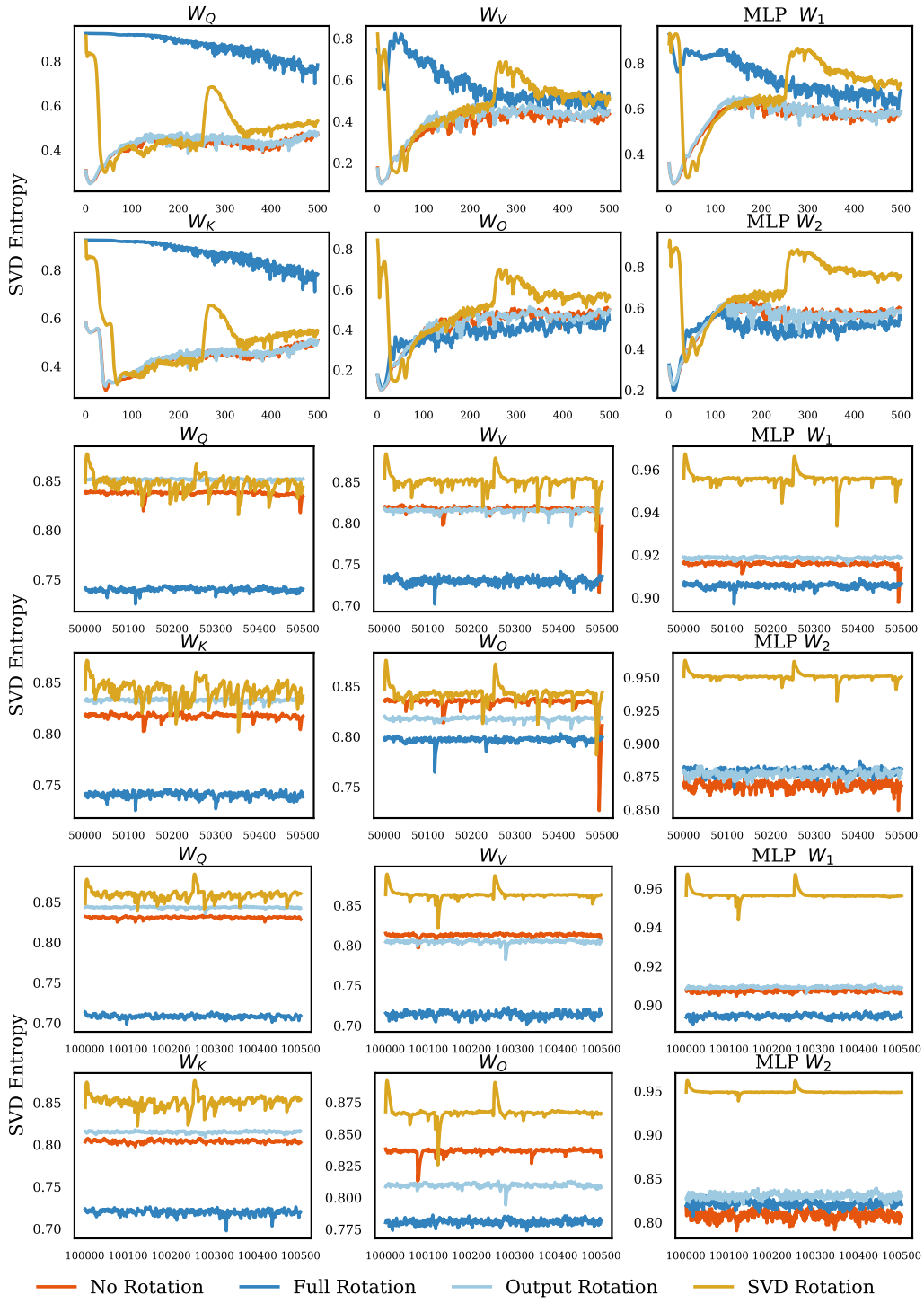


Figure C.23: The SVD Entropy metric described in (Liu and Su, 2025) throughout training, averaged over network depth. We see similar results to the coefficient of variation, noting that higher is better for this metric.

C.4 Optimization Algorithms with Rotations

We remind here the SGD-M algorithm in Algorithm 4, AdamW algorithm (pseudocode) in Algorithm 5, and provide a rotated version in Algorithm 6.

Algorithm 4 SGD Momentum Optimization Algorithm

Require: α: stepsize
Require: β: momentum parameter
Require: λ: weight decay coefficient
Require: $f(\theta)$: stochastic objective function with parameters θ
1: Initialize θ_0, $t \leftarrow 0$
2: **while** θ_t not converged **do**
3: $t \leftarrow t + 1$
4: $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ ▷ Get gradients w.r.t. stochastic objective at timestep t
5: $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot g_t + \beta(\theta_{t-1} - \theta_{t-2}) - \alpha \cdot \lambda \cdot \theta_{t-1}$ ▷ Update parameters
6: **end while**
7: **return** θ_t ▷ Return the final parameters

Proof. Using the notation of Section 4.2, we consider SGD with momentum[4] with learning rate η, momentum parameter β, and fixed batches B_t:

$$w_{t+1} = \mathcal{A}(\{w_i\}, f, t) = w_t - \eta \nabla f_{B_t}(w_t) + \beta(w_t - w_{t-1}),$$

By the chain rule, $\nabla f_{B_t}^{(\mathbf{R})}(w_t) = \mathbf{R}\nabla f_{B_t}(\mathbf{R}^\top w_t)$, hence:

$$\begin{aligned} w_{t+1}^{(\mathbf{R})} &:= \mathcal{A}(\{\mathbf{R}w_i\}_{i=0\ldots t}, f^{(\mathbf{R})}, t) \\ &= \mathbf{R}w_t - \eta\mathbf{R}\nabla f_{B_t}(\mathbf{R}^\top\mathbf{R}w_t) + \beta(\mathbf{R}w_t - \mathbf{R}w_{t-1}) \\ &= \mathbf{R}w_t - \eta\mathbf{R}\nabla f_{B_t}(w_t) + \beta\mathbf{R}(w_t - w_{t-1}) \\ &= \mathbf{R}w_{t+1} \end{aligned}$$

matching Definition 1. □

While the SVD rotation we use in the AdamW algorithm can be represented as in Algorithm 6 mathematically for a specific choice in R, for clarity and to match our implementation we write the SVD rotated AdamW in Algorithm 7. In our experiments, the SVD update frequency F was set to 250 steps. We note that while the other rotations we study are written as matrix-vector products, the SVD rotation is written as a a left and right matrix product on the gradient matrix. These can be shown to be mathematically equivalent, but we clarify given the standard practice of writing the gradient as a vector.

[4]We consider the heavy ball formulation here (Polyak, 1964), but the same would hold for Nesterov Accelerated Gradient (Nesterov, 1983).

Algorithm 5 AdamW Optimization Algorithm

Require: α: stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: exponential decay rates for moment estimates
Require: λ: weight decay coefficient
Require: ϵ: small constant for numerical stability
Require: $f(\theta)$: stochastic objective function with parameters θ
Initialize θ_0, $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ ▷ Get gradients w.r.t. stochastic objective at timestep t
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ ▷ Update biased first moment estimate
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ ▷ Update biased second raw moment estimate
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ ▷ Compute bias-corrected first moment estimate
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ ▷ Compute bias-corrected second raw moment estimate
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) - \alpha \cdot \lambda \cdot \theta_{t-1}$ ▷ Update parameters
end while
return θ_t ▷ Return the final parameters

Algorithm 6 AdamW Optimization Algorithm with Rotation

Require: α: stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: exponential decay rates for moment estimates
Require: λ: weight decay coefficient
Require: ϵ: small constant for numerical stability
Require: $f(\theta)$: stochastic objective function with parameters θ
Initialize θ_0, $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ ▷ Get gradients w.r.t. stochastic objective at timestep t
    $\tilde{g}_t = R \cdot g_t$ ▷ Apply rotation to gradients
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \tilde{g}_t$ ▷ Update biased first moment estimate
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot \tilde{g}_t^2$ ▷ Update biased second raw moment estimate
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ ▷ Compute bias-corrected first moment estimate
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ ▷ Compute bias-corrected second raw moment estimate
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot R^{-1} \cdot (\hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)) - \alpha \cdot \lambda \cdot \theta_{t-1}$ ▷ Update parameters
end while
return θ_t ▷ Return the final parameters

Algorithm 7 AdamW Optimization Algorithm with SVD Rotation

Require: α: stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: exponential decay rates for moment estimates
Require: λ: weight decay coefficient
Require: ϵ: small constant for numerical stability
Require: F: a frequency at which to update the SVD matrices
Require: $f(\theta)$: stochastic objective function with parameters θ

Initialize θ_0, $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ ▷ Get gradients w.r.t. stochastic objective at timestep t
  if $t \mod F = 0$ **then**
    $U, S, V^\top \leftarrow \text{SVD}(g_t)$ ▷ Calculate the Singular Value Decomposition of g_t
  end if
  $\tilde{g}_t = U^\top \cdot g_t \cdot V$ ▷ Apply rotation to gradients
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \tilde{g}_t$ ▷ Update biased first moment estimate
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot \tilde{g}_t^2$ ▷ Update biased second raw moment estimate
  $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ ▷ Compute bias-corrected first moment estimate
  $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ ▷ Compute bias-corrected second raw moment estimate
  $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot U \cdot (\hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)) \cdot V^\top - \alpha \cdot \lambda \cdot \theta_{t-1}$ ▷ Update parameters
end while
return θ_t ▷ Return the final parameters

Algorithm 8 Muon Optimization Algorithm

Require: α: step size
Require: μ: Nesterov momentum parameter
Require: λ: weight decay coefficient
Require: ϵ: small constant for numerical stability
Require: $f(\mathbf{\Theta})$: stochastic objective function with parameters $\mathbf{\Theta}$

Initialize θ_0, $\mathbf{B}_0 \leftarrow \mathbf{0}$, $t \leftarrow 0$
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $\mathbf{G}_t \leftarrow \nabla_\theta f_t(\mathbf{\Theta}_{t-1})$ ▷ Get gradients w.r.t. stochastic objective at timestep t
  $\mathbf{B}_t \leftarrow \mu \mathbf{B}_{t-1} + \mathbf{G}_t$ ▷ Update momentum buffer
  $\mathbf{B}'_t \leftarrow \mu \mathbf{B}_t + \mathbf{G}_t$ ▷ Apply Nesterov Momentum
  $\tilde{\mathbf{B}}_t \leftarrow \mathbf{B}_t / (\|\mathbf{B}_t\|_F + \epsilon)$ ▷ Normalize the update
  $\mathbf{O}_t \leftarrow \text{NewtonSchulz5}(\tilde{\mathbf{B}}_t)$ ▷ Approximately orthogonalize the update
  $\mathbf{\Theta}_t \leftarrow \mathbf{\Theta}_{t-1} - \alpha \mathbf{O}_t - \alpha \cdot \lambda \cdot \boldsymbol{\theta}_{t-1}$ ▷ Update parameters
end while
return $\mathbf{\Theta}_t$ ▷ Return the final parameters

C.5 SVD Rotations and Muon

Recently, (Jordan et al., 2024) proposed Muon, an optimization algorithm for the internal linear layers of neural networks. This algorithm departed from various modifications of Adam on favour of using an "orthogonalized" matrix update. We write the Muon algorithm in Algorithm 8. We note that a more simple version of the algorithm is described in Jordan et al. (2024), however their implementation and the description in subsequent work is as described in Algorithm 8. We additionally make a notational switch to emphasize that Muon acts only on the matrices of internal layers, (Jordan et al., 2024) recommends using a different update scheme (e.g., AdamW) on vector valued parameters such as bias vectors or LayerNorm parameters (along with the embedding and prediction layer in transformers). We write parameters at time t as $\mathbf{\Theta}_t$ and their gradients as $\mathbf{G}_t$.

Muon's normalization and orthogonalization step aims to drive the singular values of the update towards one. That is, if $\mathbf{B}'_t$ has the Singular Value Decomposition $\mathbf{USV}^\top$, Muon aims to have the update approximate $\mathbf{UV}^\top$. The approximation is computed through an iterative algorithm inspired by the Newton-Schulz method. We show that under simplifications, this is the same update recovered by SVD Rotated Adam(W). If we let $\beta_1 = \beta_2 = \epsilon = 0$ in Adam(W) and compute a single step update with rotation, the numerator and denominator terms become the singular values of the gradient matrix, which cancel, and the rotation back to the original basis leaves us with $\mathbf{UV}^\top$. Mathematically, let $\mathbf{USV}^\top$ be the SVD of gradient $\mathbf{G}$. Then, the SVD Rotated Adam(W) numerator becomes $\mathbf{M} = \mathbf{U}^\top \mathbf{GV} = \mathbf{U}^\top \mathbf{USV}^\top \mathbf{V} = \mathbf{S}$. Similarly, the denominator is the entry-wise square of the rotated gradient, which leaves us with $\mathbf{V} = \mathbf{S}^2$. Then, computing the update $\mathbf{M}/\sqrt{\mathbf{V}}$ leaves us with $\mathbf{S}/\mathbf{S}$ which is the identity.

The final step in the algorithm is to rotate this update back, which is done by $\mathbf{UIV}^\top$, leaving us with the Muon update.

While this setting is an oversimplification (the momentum parameters are often crucial for performance), it does offer an interesting connection between Adam's update in a different basis and more recent algorithms like Muon or Shampoo (Gupta et al., 2018).

C.6 Common Assumptions in First-Order Optimization Theory

We present a non-exhaustive summary of common assumptions used in theoretical works for first-order optimization, see Table C.1. For each assumption, we indicate whether it is rotation invariant.

Assumption	**Rotation-Invariant**
(Strong-) Convexity	✓
Polyak-Lojasiewicz (Polyak, 1963)	✓
Star-(Strong)-Convexity (Guille-Escuret et al., 2020)	✓
Quadratic Growth (Goujaud et al., 2022)	✓
L-Smoothness (L_2 norm) (Défossez et al., 2022; Zhou et al., 2024)	✓
Gradient Growth Condition (Zhang et al., 2022)	✓
Bounded Expected Gradient Squared Norm (Zou et al., 2019)	✓
(L_0, L_1)-Smoothness (Li et al., 2023b)	✓
Restricted Secant Inequality (Guille-Escuret et al., 2022)	✓
Error Bound (Luo and Tseng, 1993; Guille-Escuret et al., 2024)	✓
L-smoothness (L_∞ norm)(Guo et al., 2022)	✗
Coordinate-wise (L_0, L_1)-Smoothness (Crawshaw et al., 2022)	✗
Coordinate-wise "Affine" Variance Noise (Li and Lin, 2024)	✗
Bounded Gradient (L_∞) (Reddi et al., 2018)	✗

Table C.1: Common assumptions involved in first-order optimization algorithms, indicating whether they are rotation-invariant. Rotation-dependent assumptions are comparatively rare in the literature.