Duality, Weight Decay, and Metrized Deep Learning

by

Laker Newhouse

S.B. Mathematics, Artificial Intelligence & Decision Making,
Massachusetts Institute of Technology, 2025

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2025

Authored by:	Laker Newhouse Department of Electrical Engineering and Computer Science May 16, 2025
Certified by:	Phillip Isola Associate Professor, Thesis Supervisor
Accepted by:	Katrina LaCurts Chair, Master of Engineering Thesis Committee

Duality, Weight Decay, and Metrized Deep Learning

by

Laker Newhouse

Submitted to the Department of Electrical Engineering and Computer Science
on May 16, 2025 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

The Muon optimizer has shown convincing evidence that it is faster and more scalable than AdamW for deep learning training, setting speed records for training NanoGPT and scaling up to models with 16B parameters. The theory that led to Muon is called *metrized deep learning*, a method that suggests assigning norms to each part of a neural network. Chapter 1 begins with an accessible explanation of metrized deep learning, including one of its recurring tools: odd polynomial iterations that act directly on singular values. Chapter 2 reviews duality, a way to modify the gradient that seeks to decrease the loss the most while disturbing the model the least. Pedagogically, duality links four popular optimizers—SGD, Adam, Shampoo, and Muon—under a common framework, steepest descent under a norm. Practically, experiments suggest that duality-based optimizers train faster than AdamW and transfer learning rate across width. Chapter 3 develops tools to enforce weight norm constraints during training, conferring provable and upfront Lipschitz guarantees for transformers. We find that optimizer dynamics matter: switching from AdamW to Muon improves standard weight regularization methods—weight decay and spectral normalization—allowing models to reach equal performance with a lower Lipschitz bound. Leveraging that Muon's update has a fixed spectral norm, we co-design a weight constraint method called *spectral cap* that improves the Lipschitz vs. performance tradeoff for MLPs and 2M parameter transformers. Our 4-Lipschitz transformer on Shakespeare text reaches validation accuracy 60%. Scaling to 145M parameters, our 600-Lipschitz transformer reaches 21% accuracy on internet text. However, to match the NanoGPT baseline validation accuracy of 39.4%, our Lipschitz upper bound increases to 10^{274}. Nonetheless, our Lipschitz transformers train without stability measures such as layer norm, QK norm, and tanh logit softcapping.

Thesis supervisor: Phillip Isola
Title: Associate Professor

Acknowledgments

Thank you to Phillip Isola for welcoming me and creating a lab that values curiosity and science. Thank you to Jeremy Bernstein for advising me, teaching me, and inspiring me. Thank you to my family for an unbounded set of gratitudes, quite unlike the bounds we shall impose on our neural networks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Note 1.0.1. "You can do anything as long as you don't need credit."

Quite apart from President Truman's words, intellectual credit is due to several collaborators to whom this thesis is indebted. A common thread is that Jeremy Bernstein has thought rigorously about neural networks for ten years, lending a way of thinking—culminating in the metrized deep learning theory with Tim Large—that produced the fruits of our collaboration this year. I am also very grateful to Keller Jordan for turning his outstanding engineering prowess toward birthing the Muon optimizer together with us in fall 2024. In sum, this thesis is indebted to these works:

- Modular norm theory: Large, Liu, Huh, Bahng, Isola, Bernstein
- Optimizer anthology: Bernstein and **Newhouse**
- Modular duality theory: Bernstein and **Newhouse**
- Muon optimizer: Jordan, Jin, Boza, You, Cesista, **Newhouse**, Bernstein
- Lipschitz transformers: **Newhouse**, Hess, Cesista, Zahorodnii, Bernstein, Isola

AdamW [Loshchilov and Hutter, 2019] has become the standard optimizer in machine learning. Recently, an alternative has shown evidence that it may be faster and more scalable. This alternative is both a concrete optimizer—Muon [Jordan et al., 2024b, Liu et al., 2025, Shah et al., 2025]—and also a general method that unites past optimizers and current strands of research under a common framework.

Chapter 1 explains the method of *metrized deep learning*, which came about in a series of papers by Jeremy Bernstein [Bernstein et al., 2021, Bernstein, 2022, Yang et al., 2023]. Metrized deep learning seeks to control the size of every part of a neural network, including activations, weights, and gradients. Why does size matter? What is the right way to measure the size of a matrix? Section 1.2 answers these questions and is designed to teach the reader the key thought patterns in metrized deep learning. Because the spectral norm will emerge as a useful measure of size, Section 1.3 previews a computational primitive used in later chapters—odd polynomial iterations—that allows acting directly on the singular values.

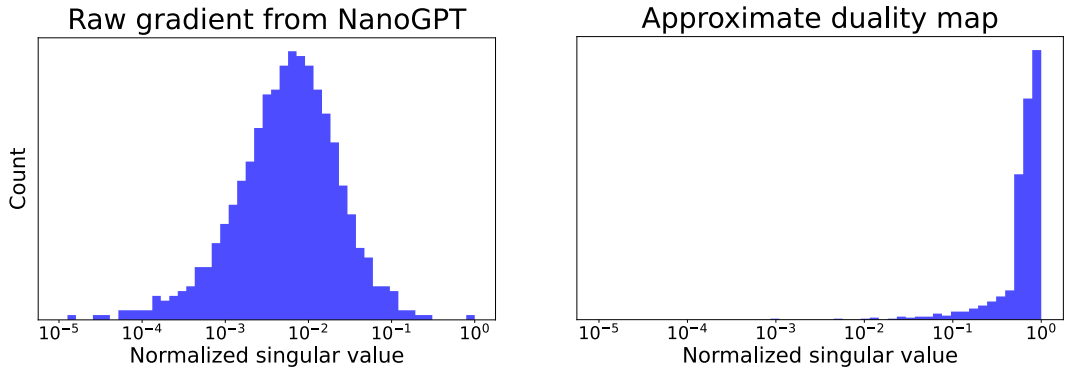

Figure 1.1: **Duality promotes signal from small singular values.** Left: Potentially useful signal could be lost because gradients are typically dominated by a few large singular values. Right: the spectral norm duality map recovers this signal. Muon [Jordan et al., 2024b] approximates the duality map using an odd polynomial that inflates singular values by at most 485×, staking an implicit claim about the scale at which singular values may truly be noise. The plot shows the spectrum of a gradient from the layer 4 query matrix at step 100 of training a 120M parameter NanoGPT on internet text [Karpathy, 2022].

In Chapter 2, metrizing deep learning leads to *duality*, a way to modify the gradient according to a geometry that is well suited to it. This chapter is based on two papers, Bernstein and Newhouse [2024a,b], for which all of the credit for the ideas goes to Jeremy Bernstein. Section 2.1 tells the story of how three popular optimizers—SGD, Adam, and Shampoo—can be viewed under a common framework as duality maps under different choices of norm. Section 2.2 derives Muon as arising naturally from the spectral norm duality map, similar to Shampoo, but implemented with odd polynomials instead of inverse matrix powers. Keller Jordan added several more engineering innovations. Section 2.3 concludes with intriguing properties of duality, including new experiments and open questions. Figure 1.1 visualizes how useful signal could be lost in smaller singular values, which a duality map can recover.

In Chapter 3, metrizing deep learning is applied to training transformers with strict weight norm constraints. This chapter is based on forthcoming work done jointly with Jeremy Bernstein and other collaborators. Section 3.1 motivates extending the benefits of small Lipschitz bounds—intuitively, the sensitivity of a network to input perturbations—to transformers beyond initialization. To develop a toolkit for training transformers with an enforced Lipschitz constant, Section 3.2 compares several methods for constraining weight norm. Perhaps surprisingly, optimizer choice appears to matter: standard methods such as weight decay [Krogh and Hertz, 1991] and spectral normalization [Yoshida and Miyato, 2017] improve a Lipschitz vs. performance tradeoff more with Muon than AdamW. Beyond standard methods, the fixed update norm of Muon inspires designing a weight constraint method called *spectral soft cap*, which enforces a desired maximum spectral norm $\sigma_{\max}$ by approximating the map $\sigma \mapsto \min(\sigma_{\max}, \sigma)$ on all singular values σ in parallel by iterating odd polynomials on the weights. In Section 3.3, a 4-Lipschitz transformer is capable of achieving 60% accuracy on Shakespeare text. Scaling to the NanoGPT speedrun benchmark [Jordan et al., 2024a], we train 145M parameter transformers with an enforced Lipschitz constant without layer norm [Ba et al., 2016] or QK norm [Henry et al., 2020].

In any endeavor, it is worthwhile to examine carefully the existing foundation. Two common practices in training with AdamW provide footholds into the ideas in later chapters. Therefore, the first step on this journey is to look at the way many researchers currently train models.

1.1 Optimizing with AdamW

This section aims to capture the current "word on the street" of how to optimize deep learning models. At the end, we point out two items that the following chapters will reconsider: gradient clipping and weight decay. The recipe looks as follows:

1. Use AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$ [Jordan et al., 2024a, Kumar et al., 2024].
2. Sweep learning rate in logspace [Bengio, 2012], typically around 10^{-5} to 10^{-3}.
3. Use weight decay 0.1 [Brown et al., 2020, Ahmadian et al., 2023, DeepSeek-AI, 2025].
4. Clip gradient norm: $g \mapsto g/\|g\|_2$ if $\|g\|_2 > 1$ [Kumar et al., 2024, DeepSeek-AI, 2025].
5. Use $\epsilon = 10^{-15}$ to avoid instabilities from tiny gradient norms [Wortsman et al., 2023].
6. Use a linear or cosine learning rate schedule [Bengio, 2012, Jordan et al., 2024a].

Note 1.1.1. Improving gradient clipping

As a preview of Chapter 2, the motivation behind clipping the gradient is to stabilize training. But an ideal optimizer can be viewed as holding two goals at once: minimize the loss the most, while disturbing the model the least. Gradient clipping makes two implicit choices: 1) that the gradient already points in a useful "direction" and 2) that the right geometry to measure model disturbance in is Euclidean via the ℓ_2 norm $\|\cdot\|_2$. Duality revisits these choices, suggesting that different choices of norm yield different optimizers—SGD, AdamW, and Muon are special cases if momentum is turned off—that may be better suited to neural networks.

Note 1.1.2. Upgrading weight decay to a strict norm constraint

As a preview of Chapter 3, when weight decay is combined with a bounded weight update in some norm, the weight in that same norm cannot exceed $1/\lambda$ [Pethick et al., 2025]. The reason is an equilibrium occurs between the update step and weight decay when the weight norm w satisfies $w = w(1 - \lambda\eta) + \eta$ for learning rate $\eta > 0$. Chapter 3 leverages this observation and extends it, since Muon's updates have fixed spectral norm, although Muon accomplishes this property not via clipping but with duality.

1.2 Metrizing deep learning

Contribution statement: Jeremy Bernstein developed the metrized deep learning method—a framework for assigning norms, or a way to measure size, to the weight matrices in a neural network—in a series of papers, especially "A Spectral Condition for Feature Learning," "Optimisation and Generalisation in Networks of Neurons," and "Scalable Optimization in the Modular Norm" [Yang et al., 2023, Bernstein, 2022, Large et al., 2024].

Consider a thought experiment. Suppose $x = 1$, and we wish to minimize the function $f(x) = x^2$. The negative gradient is -2, not far from the step -1 that minimizes the loss. But what if the curvature increased to $f(x) = 100x^2$? The gradient becomes far too large. What if the function flattened to $f(x) = x^2/100$? The gradient becomes far too small. It appears that the magnitude of the gradient is not always helpful information. In this thought experiment, the direction is what is useful.

Function	Negative Gradient	Optimal Step
x^2	-2	-1
$100x^2$	-200	-1
$0.01x^2$	-0.02	-1

Table 1.1: Thought experiment: to minimize quadratics with different curvatures, it is helpful to ignore the *magnitude* of the gradient in favor of the *direction* information.

Many optimizers such as Adam are motivated from second-order theory—leveraging curvature information—and other optimizers explicitly approximate the Hessian [Nocedal and Wright, 2006, Nesterov and Polyak, 2006]. This thought experiment suggests a different idea: *normalizing* the gradient, and ignoring curvature, may be enough. However, that is not the full story. In Chapter 2, duality indeed limits the update magnitude, but it also seeks to *minimize the loss*. Duality will connect SGD, Adam, Shampoo, and Muon (with momentum turned off) as arising from different choices of *norm*—or different ways to abstract away the size information from the gradient. This control over the gradient forms a step in a more general three-step agenda for building an architecture-aware optimizer:

1. Bound the change in loss in terms of changes in the network output.
2. Relate changes in network output to changes to the weights.
3. Bound changes to the weights.

This agenda gives a first-principles way to reason about creating an optimizer that will update the weights with the end goal in mind—reducing the loss. Jeremy Bernstein's PhD thesis proposes this agenda [Bernstein, 2022], building on previous work towards architecture-aware optimizers [Bernstein et al., 2021, 2023, Liu et al., 2021], including the majorization-minimization recipe [Lange, 2016, Streeter, 2023].

What is the right way to carry out this agenda for deep learning? Large et al. [2024] propose developing such bounds for the atoms of deep learning—Linear, Embed, Conv2D,

and other layers—and stitching the bounds together to create a recursive, architecture-aware bound for any neural network.

The central idea is to equip input and output activation spaces with a norm to measure size. Different atoms may benefit from different norms. For example, while Linear and Embed are both linear maps with weights in $\mathbb{R}^{m\times n}$, Linear expects dense input vectors while Embed expects one-hot input vectors. Different choices of norm will give rise to different optimizers. Chapter 2 links the ℓ_∞ norm with Adam and the RMS norm—a rescaled ℓ_2 norm that considers the dense all-ones vector to be unit norm—with Muon.

Metrized deep learning provides no principle that proves that one norm is better than another in a particular setting. Instead, which norms are best for different situations may depend on many factors, and nailing down a precise answer is an open question. The framework posits that a choice of norm, however, is a critical design decision.

Nonetheless, there is evidence that a rescaled spectral norm may be the most natural choice for Linear layers. The so-called RMS $\to$ RMS ("RMS to RMS") operator norm equals 1 when the spectral norm equals $\sqrt{d_{\text{out}}/d_{\text{in}}}$ for a weight matrix $W \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$. For a first intuition of why this norm makes sense, some activation functions like GeLU [Hendrycks and Gimpel, 2023] are designed for activations with entries near 1. If the all-ones vector in d dimensions is to have unit norm, the only scaling of an ℓ_2 norm that works is $\|\cdot\|_{\text{RMS}} = \frac{1}{\sqrt{d}}\|\cdot\|_2$. Because of this scaling, the RMS norm is dimensionless and is useful across widths. For a second intuition, if feature learning is to mean changes in activations measured in a Euclidean space—which may make sense if representation spaces have linear structure [Park et al., 2024]—then a rescaled ℓ_2 norm is the only choice. A major upside of using the RMS $\to$ RMS operator norm size is that it recovers the initialization and update rules from maximal-update parameterization, or μP, a way to scale networks up while preserving the optimal hyperparameters [Yang et al., 2022, 2023]. The RMS norm therefore reproduces a path toward learning rate transfer across width, allowing one to tune the learning rate on a small network and then use the same optimal learning rate in the larger setting.

Once one has selected an input and output norm, one can norm the overall matrix.

Definition 1.2.1. Operator norm of a matrix

Let W be a linear map from a vector space A to a vector space B. Equipping A and B with norms $\|\cdot\|_A$ and $\|\cdot\|_B$ induces an *operator norm* $\|\cdot\|_{A\to B}$ defined by

$$\|W\|_{A\to B} = \max_{x\in A:\|x\|_A=1} \|Wx\|_B.$$

Operator norms are flexible and offer an array of design choices. Some are recognizable:

- For input ℓ_1 and output ℓ_∞, the operator norm is the max absolute entry.
- For input ℓ_2 and output ℓ_2, the operator norm is the spectral norm $\|\cdot\|_{2\to 2}$.
- For input RMS and output RMS, the operator norm is scaled as $\sqrt{\frac{d_{\text{in}}}{d_{\text{out}}}}\|\cdot\|_{2\to 2}$.

A common and important operator norm is the spectral norm.

Definition 1.2.2. Singular value decomposition (SVD) and spectral norm

The *singular value decomposition* of a matrix $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ always exists and is written $W = U\Sigma V^T$, where U and V are orthogonal matrices and Σ is diagonal. The entries of Σ are nonnegative and are known as the singular values of W. The *spectral norm* $\|\cdot\|_{2\to 2}$ of W is its largest singular value. The singular values record the amount that W can stretch inputs in the ℓ_2 norm. The orthogonal matrices U and V, which satisfy $UU^T = I$ and $VV^T = I$, are rotations that do not affect the ℓ_2 norm at all.

The three-step agenda for an architecture-aware loss bound is now possible to carry out. First, near the current network output y, set out a local quadratic model of the loss function $\mathcal{L}$ of the form $\mathcal{L}(y) + \nabla_y \mathcal{L}(y)^\top \Delta y + \frac{\lambda}{2} \cdot \|\Delta y\|^2$. Crucially, the sharpness parameter λ and norm $\|\cdot\|$ are chosen a-priori without ever touching an (approximate) Hessian during training.

Second, operator norms enable bounding changes to network output in terms of changes to weights. A matrix W acts on an input vector x to produce an output vector y like

$$y = Wx. \tag{1.1}$$

If the input space has norm $\|\cdot\|_A$ and the output space has norm $\|\cdot\|_B$, then a change to the weights ΔW changes the network output an amount bounded by the operator norm:

$$\|\Delta y\|_B \leqslant \|\Delta W\|_{A\to B} \|x\|_A. \tag{1.2}$$

Equation (1.2) says that, if the activations are kept well-normed, then controlling the change in network output amounts to controlling the operator norm of the weight update. All in all, the loss function admits a local approximation, this time in terms of the weights:

$$\mathcal{L}(W + \Delta W) \approx \mathcal{L}(W) + \langle \nabla_W \mathcal{L}(W), \Delta W \rangle + \frac{\lambda}{2} \cdot \|\Delta W\|^2_{A\to B}, \tag{1.3}$$

where $\langle \cdot, \cdot \rangle$ denotes the Frobenius inner product—the sum of the entrywise product of two matrices. To finish out with the third step, bounding weight updates ΔW in the $\|\cdot\|_{A\to B}$ operator norm will now control the change to the loss from every step of training.

To bound a weight update, the first thing one might think to do is to *normalize* it like $\Delta W \mapsto \Delta W / \|\Delta W\|_{A\to B}$, recalling what worked in the thought experiment (Table 1.1). This approach is called gradient clipping, often done with the ℓ_2 or ℓ_∞ norm on the flattened matrix. But gradient clipping is not ideal in higher dimensions. For the spectral norm, Figure 1.1 visualizes how the gradient matrix typically has a few large singular values and many more small ones. The small singular values may contain important training signal, but normalizing the gradient risks suppressing this signal. Instead of normalizing, Chapter 2 develops an alternative: *dualizing* the gradient. For a gradient matrix $G \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, duality arises by solving a steepest descent problem under a norm [Boyd and Vandenberghe, 2004b, Bernstein and Newhouse, 2024a]. The steepest descent problem aims to reconcile two tensions: a desire to move infinitely in the direction of linearized improvement (i.e., $\langle G, \Delta W \rangle$) and a desire to stay close to the region of validity of the linearization (i.e., $\frac{\lambda}{2}\|\Delta W\|^2$). Namely,

$$\Delta W_{\text{steepest}} = \underset{\Delta W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}}{\arg\min} \left[\langle G, \Delta W \rangle + \frac{\lambda}{2} \|\Delta W\|^2 \right]. \tag{1.4}$$

The art becomes to choose norms that will lead to tight approximations to Equations (1.2) and (1.3). Large et al. [2024] took early steps toward turning this art into a science by developing the *modular norm*, which recursively extends norm choices on atomic modules to define a single norm for the entire neural network. Unlike prescient work by [Flynn, 2017], the modular norm takes a max over atomic norms, with scale factors that automatically track and control the sensitivity of the network output, even up to second order.

While applying a duality map to gradients is one application of metrized deep learning (Chapter 2), another is to enforce bounds on the weight matrix norm (Chapter 3). A common thread is that controlling the singular values is a core operation. But it is inefficient to compute a singular value decomposition $W = U\Sigma V^T$ on a GPU, which would be necessary to apply a function $f(x)$ to every singular value directly. For weight decay, a miracle occurs: $(1-\lambda)U\Sigma V^T = U(1-\lambda)\Sigma V^T$, where $\lambda > 0$ is the weight decay parameter. Though apparently mundane, multiplying by a constant commutes to act on the singular values, preserving spectral structure without ever computing an SVD. The next section generalizes this observation to construct a GPU-friendly computational primitive for control over the singular values.

1.3 A primitive for controlling singular values

Contribution statement: The idea to apply odd polynomial iterations to matrices is not new, but classically the coefficients were derived based on Taylor approximations [Kovarik, 1970, Björck and Bowie, 1971, Higham, 2008]. Jeremy Bernstein told me about the method and proposed using these iterations to orthogonalize gradient matrices [Bernstein and Newhouse, 2024a]. Jeremy Bernstein also popularized tuning the coefficients graphically using Desmos to find polynomials that converge faster [Bernstein, 2025]. I proposed applying odd polynomials directly to weight matrices, for instance to approximate $\min(1, x)$ *to spectrally cap the singular values at* 1 *while also allowing them to decay to* 0.

Suppose one wants to apply a continuous function $f : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ to all the singular values $\sigma \geqslant 0$ of a matrix, with the only condition that $f(0) = 0$. In other words, if the matrix has singular value decomposition $U\Sigma V^T$, the goal is to map

$$U\Sigma V^T \mapsto U f(\Sigma) V^T. \tag{1.5}$$

Two earlier methods each suffer from a problem:

1. **Do the SVD**. This method applies the SVD routine to compute U, Σ, and V, returning $Uf(\Sigma)V^T$. But the SVD is slow to run on a GPU as it is not significantly parallelizable.

2. **Do sketching.** Sketching is a randomized method [Martinsson and Tropp, 2020] that approximates the SVD. While more tractable to compute, sketching acts only on the top k singular values while setting the rest to 0. The small singular values may matter.

A third method solves both problems, because it is efficient to run on a GPU and acts on every singular value:

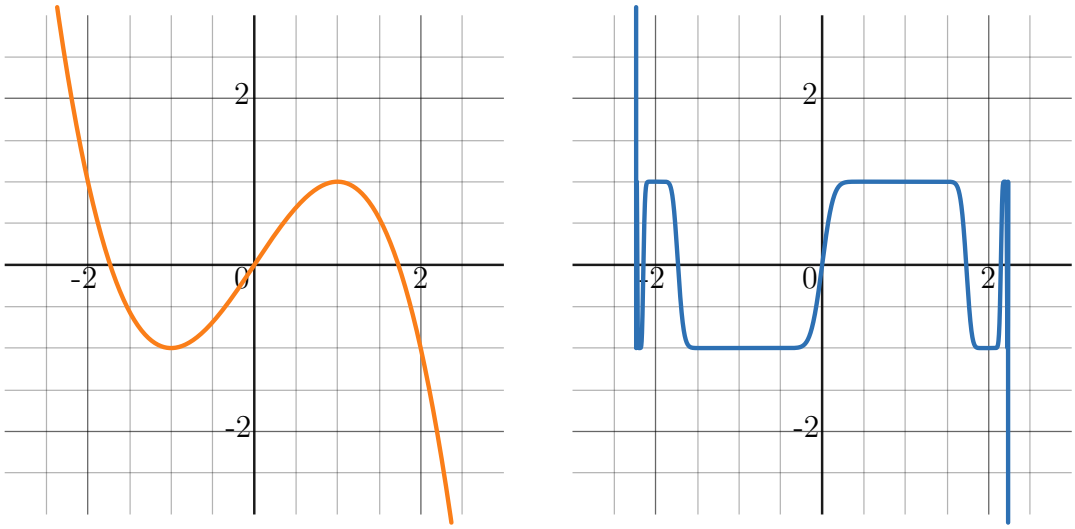

Figure 1.2: Preview of Chapter 2: approximating the function $\text{sign}(x)$ on the singular values of a gradient implements the spectral norm duality map. One odd polynomial approximation comes from iterating $p(x) = \frac{3}{2}x - \frac{1}{2}x^3$, which converges to $\text{sign}(x)$ in the range $(-\sqrt{3}, \sqrt{3})$. The linear coefficient controls the convergence speed. Plots reproduced from Bernstein [2025].

3. **Iterate odd polynomials.** Suppose $p_1, \ldots, p_n$ are odd, single-variable polynomials that compose to approximate a continuous function $f : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ with the condition that $f(0) = 0$. Weierstrass's theorem shows that an approximation can be made arbitrarily good: approximate $g : \mathbb{R} \to \mathbb{R}$, defined as $f(x)$ on $x \geqslant 0$ and $-f(-x)$ on $x < 0$, then use the approximation on $x \geqslant 0$. Because p_k is odd, applying it to a matrix acts on the singular values directly as $p_k(U\Sigma V^T) = Up_k(\Sigma)V^T$, as the theorem below shows. Therefore, applying p_1 up to p_n in succession will apply the approximation of $f(x)$ to every singular value of a matrix.

Theorem 1.3.1. Odd polynomials act directly on the singular values

If $p(x)$ is an odd polynomial and $U\Sigma V^T$ is a singular value decomposition, then $p(U\Sigma V^T) = Up(\Sigma)V^T$.

Proof. Let $W = U\Sigma V^T$ be a singular value decomposition. Consider one odd power term W^{2k+1}. To make the shapes work for rectangular matrices, the matrix power is defined to be $W^{2k+1} = (WW^T)^k W$. The trick is that U and V are orthogonal, which means $UU^T = I$ and $VV^T = I$. Expanding gives

$$W^{2k+1} = (WW^T)^k W = (U\Sigma^2 U^T)^k W = U\Sigma^{2k} U^T W = U\Sigma^{2k+1} V^T. \tag{1.6}$$

Since Σ^{2k+1} means raising each entry of the diagonal matrix to the power of $2k+1$, a single odd power of the matrix applies itself directly to the singular values. Linearity extends the proof to all odd polynomials. □

Example 1: Weight decay. The simplest example of an odd polynomial is scalar multiplication, $p(x) = (1 - \lambda\eta)x$, which implements weight decay for decay parameter $\lambda > 0$

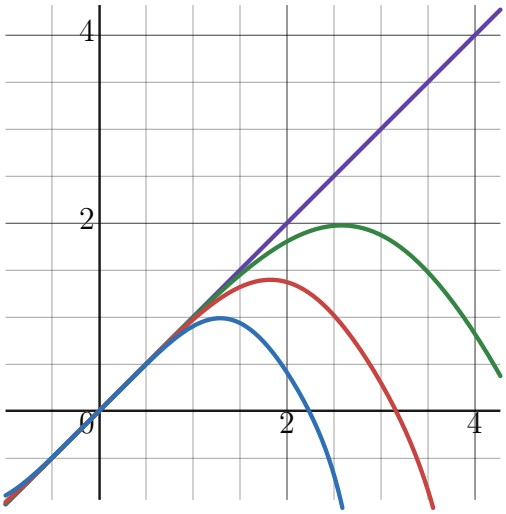

Figure 1.3: Preview of Chapter 3: *spectral soft cap* is a weight constraint method that applies the above odd polynomial to all singular values of a weight matrix in parallel. It composes $p_1(x) = x - \alpha x^3$ with $p_2(x) = x + \alpha x^3$. The result is depicted for $\alpha = 0.2$ (blue), $\alpha = 0.1$ (red), $\alpha = 0.05$ (green), $\alpha = 0$ (purple).

and learning rate $\eta > 0$. Because $(1 - \lambda\eta)U\Sigma V^T = U(1 - \lambda\eta)\Sigma V^T$, weight decay acts on the singular values of the weight matrix and preserves its spectral structure.

Example 2: Duality. The spectral norm duality map for a gradient says to set all the singular values to 1, equivalent to applying the odd function $\text{sign}(x)$ for $x > 0$. Figure 1.2 shows that iterating the odd polynomial $p(x) = \frac{3}{2}x - \frac{1}{2}x^3$ provides one such approximation, valid in the range $(0, \sqrt{3})$. To squash the singular values into this range, one can first divide by the Frobenius norm of the gradient, which upper bounds the spectral norm. Tuning the coefficients can speed up convergence, such as $p(x) = 3x - 3.2x^3 + 1.2x^5$. You [2025] discovered a six-step iteration that converges quickly and stably, shown in Figure 1.4.

Example 3: Generalized weight decay. Beyond linear polynomials, consider the polynomials $p_1(x) = x - \alpha x^3$ and $p_2(x) = x + \alpha x^3$, for strength parameter $\alpha > 0$. When $\alpha = 0.05$, the polynomial $p_2(p_1(x))$ never exceeds 2. As a result we say that this polynomial *spectrally caps* the singular values at 2. Figure 1.3 plots this polynomial for $\alpha \in \{0.2, 0.1, 0.05, 0\}$. Polynomials that approximate $\min(\sigma_{\max}, x)$ for a desired maximum spectral norm $\sigma_{\max} > 0$ become important in Chapter 3.

Note 1.3.1. Accelerating odd polynomial iterations

GPU kernels for odd polynomial iterations can be accelerated around 30% by exploiting that many of the matrix multiplications are symmetric, like $AA^\top$. I proposed this idea and wrote a proof-of-concept kernel [Newhouse et al., 2024], but it had a bug. Our code used ThunderKittens, a framework for simple and fast kernels [Spector et al., 2024]. Lin [2025] fixed the bug and called the algorithm Flash-Muon.

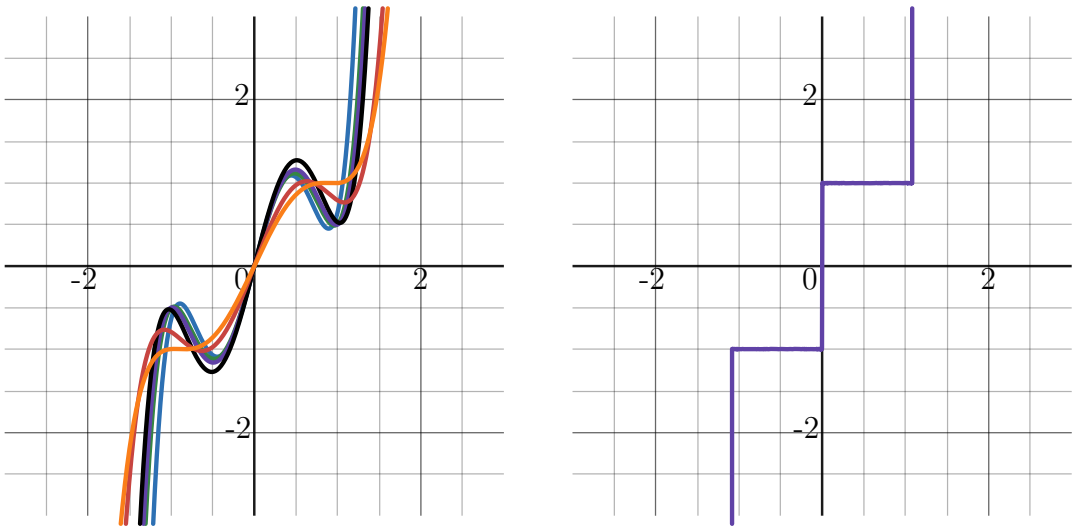

Figure 1.4: You [2025] discovered six odd polynomials that compose to closely approximate $\text{sign}(x)$, which implements the spectral norm duality map (Chapter 2) when applied to the singular values of a gradient matrix. You [2025]'s method is to use numerical optimization to search for good coefficients. Plots reproduced from Bernstein [2025].

Chapter 2

Duality in deep learning

2.1 Old optimizer, new norm

Contribution statement: The ideas in this section are primarily due to Jeremy Bernstein. The narrative adapts two papers we wrote together, "Old Optimizer, New Norm: An Anthology" and "Modular Duality in Deep Learning" [Bernstein and Newhouse, 2024a,b].

This section tells the story of how three popular optimizers—SGD, Adam, and Shampoo—can each be understood from a common concept called *duality*. The upshot is that the first step of metrizing deep learning—equipping activation spaces with *norms*—links past optimizers under a common framework and may provide inspiration for designing faster, more scalable optimizers. The section after this one will show how duality leads to the Muon optimizer, which set speed records for training NanoGPT [Jordan et al., 2024a] and has since been validated at scale [Liu et al., 2025, Shah et al., 2025].

The spirit of duality is that an ideal optimizer wants two things at once:

1. Reduce the loss the most.
2. Disturb the model the least.

But to reduce the loss, the model must change. The compromise is constrained optimization:

$$\arg\min_{\Delta W} \ \mathrm{tr}(G^\top \Delta W) \ \text{ such that } \ \|\Delta W\| \leqslant \eta, \tag{2.1}$$

where G is the gradient, ΔW is the weight update, and $\mathrm{tr}(G^\top \Delta W)$ is the linearized improvement in the loss. A question arises: how to measure the size of the disturbance ΔW? This question is at the heart of duality. Different optimizers make different choices, summarized in Table 2.1. This section tells the story of three such choices. Credit to Jianlin Su for putting the spirit of duality in these terms [Su, 2025].

One motivation behind duality is that there is a type mismatch in the most basic form of gradient descent. To pass the type check, the gradient must be passed through a *duality map* before being multiplied by the learning rate and subtracted:

```
weight − LR ∗ weight.grad              type error!
weight − LR ∗ dualize(weight.grad)       all good!
```

Domain	Norm	Duality Map	Optimizer	Cousin
$\mathbb{R}^n$	Euclidean ℓ_2	$g \mapsto \frac{g}{\|g\|_2}$	gradient descent	SGD
$\mathbb{R}^n$	infinity ℓ_∞	$g \mapsto \text{sign}(g)$	sign descent	Adam
$\mathbb{R}^{m\times n}$	spectral $\ell_{2\to 2}$	$G \mapsto UV^\top$	spectral descent	Shampoo, Muon

Table 2.1: Popular optimizers are related to duality maps under different norms. In spectral descent, the gradient is viewed as a matrix. Its singular value decomposition is $G = U\Sigma V^\top$.

The reason is that a gradient is a *dual* vector, an entirely different object from a weight. The difference is well understood in physics, where Professor Netta Engelhardt at MIT once opened a general relativity lecture by asking, "How many of you have been lied to that the gradient is a vector?" It is also understood in mirror descent [Nemirovsky and Yudin, 1983], natural gradient descent [Amari, 2016], and steepest descent under a norm [Boyd and Vandenberghe, 2004a]. A duality map must first map the gradient to the primal space.

Definition 2.1.1. Dual vector, dual space, and duality map

Given a vector space V, a *dual vector* is any linear function $f : V \to \mathbb{R}$. For example, row vectors are functions that map column vectors to real numbers via a dot product. The *dual space* V^* of V is the set of dual vectors on V. The dual space is itself a vector space, with addition defined $(f + g)(x) = f(x) + g(x)$ and scalar multiplication defined $(\alpha f)(x) = \alpha f(x)$. A *duality map* is any function that sends a member of the dual space V^* to a member of the primal space V. We do not require that applying a duality map twice must equal the identity map. Given a dual vector $f \in V^*$, its functional form $f(\cdot)$ is often written as an inner product $\langle f, \cdot \rangle$, which is possible by the Riesz representation theorem [Riesz, 1907].

Why is the gradient a dual vector? Let $\mathcal{L} : \mathcal{W} \to \mathbb{R}$ denote a differentiable loss function for a machine learning model with weight space $\mathcal{W}$. The gradient is a dual vector because, in the Taylor expansion of the loss,

$$\mathcal{L}(W + \Delta W) = \mathcal{L}(W) + \langle \nabla_W \mathcal{L}(W), \Delta W \rangle + \text{higher-order terms}, \tag{2.2}$$

the gradient appears as a function $\langle \nabla_W \mathcal{L}(W), \cdot \rangle$. This function takes in a weight perturbation ΔW and outputs a real number representing the linearized change in loss due to the weight perturbation. Therefore the gradient lives in a different vector space from the weights. A priori, there is no clear way to add the two.

The weight space may well be $\mathbb{R}^{m\times n}$, and the gradient may well be the exact same shape. But enforcing the restriction not to add them is a reminder that the loss function may have highly heterogenous curvature. The raw gradient might not respect this heterogeneity, unless a good duality map corrects the size and direction of the gradient to better attune it to the curvature in the loss function.

One path to arrive at duality is steepest descent under a norm. As in Chapter 1, suppose

that there is some norm $\|\cdot\| : \mathcal{W} \to \mathbb{R}$ and sharpness parameter $\lambda > 0$ that serve as a good local approximation of the loss,

$$\mathcal{L}(W + \Delta W) \approx \mathcal{L}(W) + \langle \nabla_W \mathcal{L}(W), \Delta W \rangle + \frac{\lambda}{2} \cdot \|\Delta W\|^2. \tag{2.3}$$

If this approximation is good, a weight update should find a ΔW to minimize it. The solution splits into a *step size* based on a dual norm and a *step direction* based on a duality map.

Definition 2.1.2. Dual norm and duality map based on a norm

Given a norm $\|\cdot\| : V \to \mathbb{R}$ on a vector space V, the *dual norm* $\|\cdot\|^\dagger$ of a dual vector $G \in V^*$ is

$$\|G\|^\dagger := \max_{T \in V : \|T\| = 1} \langle G, T \rangle.$$

The *duality map based on a norm* is

$$\text{dualize}_{\|\cdot\|}\, G := \arg\max_{T \in V : \|T\| = 1} \langle G, T \rangle,$$

where, if the arg max is not unique, $\text{dualize}_{\|\cdot\|}$ returns any maximizer.

Theorem 2.1.1. Steepest descent under a norm

For any dual vector $G \in V^*$ thought of as "the gradient," any sharpness $\lambda \geqslant 0$, and any norm $\|\cdot\| : V \to \mathbb{R}$ with dual norm $\|\cdot\|^\dagger$ and duality map $\text{dualize}_{\|\cdot\|}$, steepest descent splits into a step size based on the dual norm and a step direction based on the duality map:

$$\arg\min_{\Delta W \in V} \left[\langle G, \Delta W \rangle + \frac{\lambda}{2} \|\Delta W\|^2 \right] = -\frac{\|G\|^\dagger}{\lambda} \cdot \text{dualize}_{\|\cdot\|}(G).$$

The proof of this theorem is based on the proof given in Bernstein and Newhouse [2024a].

Proof. Consider the minimization problem under a change of variables $\Delta W = cT$, where $c \geqslant 0$ encodes the "magnitude" and T is a unit vector ($\|T\| = 1$) that encodes the "direction":

$$\min_{\Delta W \in V} \left[\langle G, \Delta W \rangle + \frac{\lambda}{2} \|\Delta W\|^2 \right] = \min_{c \geqslant 0} \min_{T \in V : \|T\| = 1} \left[c \langle G, T \rangle + \frac{\lambda}{2} c^2 \|T\|^2 \right] \tag{2.4}$$

$$= \min_{c \geqslant 0} \left[c \cdot \min_{T \in V : \|T\| = 1} [\langle G, T \rangle] + \frac{\lambda}{2} c^2 \right] \tag{2.5}$$

$$= \min_{c \geqslant 0} \left[-c \cdot \|G\|^\dagger + \frac{\lambda}{2} c^2 \right]. \tag{2.6}$$

Inspecting Equation (2.5), the minimizer for the direction T is given by

$$T = \arg\min_{T \in V : \|T\| = 1} [\langle G, T \rangle] = -\arg\max_{T \in V : \|T\| = 1} [\langle G, T \rangle] = -\,\text{dualize}_{\|\cdot\|}(G). \tag{2.7}$$

And similarly, inspecting Equation (2.6), the minimizer for the magnitude c is given by

$$c = \arg\min_{c \geqslant 0} \left[-c \cdot \|G\|^\dagger + \frac{\lambda}{2} c^2 \right] = \frac{\|G\|^\dagger}{\lambda}. \tag{2.8}$$

Multiplying these expressions yields the minimizer for ΔW, proving the result. □

In practice, the step size is often ignored in favor of a standard learning rate schedule.

A few prescient works explored duality maps under norms, including steepest descent under the spectral norm [Carlson et al., 2016, 2015a,b, Flynn, 2017]. In fact, duality extends a classical literature including natural gradient descent that seeks to modify the gradient based on geometry. The main difference is that natural gradient descent is designed for inner product geometries, but not every norm comes from an inner product. The spectral norm is an important example, making normed geometries more diverse and potentially better suited to neural network optimization.

Duality reproduces popular optimizers (with momentum turned off) under different choices of norm. While not necessary for SGD or Adam, the main conceptual leap required for Shampoo and Muon is to think of the gradient as a matrix, rather than a vector of independent parameters.

Gradient descent as duality under the Euclidean norm. The ℓ_2 norm duality map applied to a gradient vector $g \in \mathbb{R}^d$ yields $\text{dualize}_{\|\cdot\|_2}(g) = g/\|g\|_2$. This map rescales the gradient to be an ℓ_2 norm unit vector. Since the ℓ_2 norm equals its dual norm, the steepest descent step cancels out the division to give $\Delta w = -\frac{1}{\lambda} g$. This update is exactly gradient descent with an inverse learning rate λ.

The same duality map can be viewed as arising from a norm on a matrix. The Frobenius norm $\|\cdot\|_F$ is like a dot product for a matrix: it squares the entries, sums them, and takes a square root. The duality map $\text{dualize}_{\|\cdot\|_F}$ sends a gradient matrix $G \in \mathbb{R}^{m \times n}$ to a weight update $G/\|G\|_F$, exactly as if the gradient had been a vector under the ℓ_2 norm. The Frobenius norm is not an induced matrix norm, however, meaning it does not presuppose a way to measure size in the input and output spaces.

Adam as duality under the $\ell_1 \to \ell_\infty$ norm. Adam is often considered the standard optimizer for deep learning, with well over 100,000 citations for the original paper [Kingma and Ba, 2015] and over 25,000 citations for its cousin AdamW [Loshchilov and Hutter, 2019]. Adam has been motivated in various ways, including through convex analysis [Kingma and Ba, 2015] and as an approximate second-order method [Sun and Spall, 2021]. However, there are more direct explanations: with exponential moving averages (EMA) switched off, Adam is just *sign gradient descent* [Balles and Hennig, 2018, Bernstein et al., 2018], equivalent to steepest descent under the infinity norm [Carlson et al., 2015a]. Viewed another way, Adam is also steepest descent under an induced matrix norm, though with a catch at the end.

To begin, ignoring bias correction and numerical stabilization, Adam is given by the following updates:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t, \tag{2.9}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2, \tag{2.10}$$

$$w_{t+1} = w_t - \eta \cdot m_t / \sqrt{v_t}, \tag{2.11}$$

where t denotes the time step, $g_t \in \mathbb{R}^n$ the gradient vector, and $\eta > 0$ the step size. The exponential moving average (EMA) time scales of the gradient's first moment m_t and second moment v_t are set by $0 \leqslant \beta_1, \beta_2 < 1$. All operations are conducted entrywise. Switching off the EMA by setting $\beta_1 = \beta_2 = 0$, the Adam updates reduce to just sign gradient descent:

$$w_{t+1} = w_t - \eta \cdot g_t / \sqrt{g_t^2} \tag{2.12}$$

$$= w_t - \eta \cdot \operatorname{sign}(g_t). \tag{2.13}$$

Why should sign descent be a good idea in deep learning? One reason might be that it solves a duality problem under the ℓ_∞ norm. Given a vector $g \in \mathbb{R}^d$ thought of as the gradient, the duality map is $\operatorname{dualize}_{\|\cdot\|_\infty}(g) = \operatorname{sign}(g)$, where the sign function is applied entrywise and $\operatorname{sign}(0) = 0$. In other words, the vector that minimizes the linearized loss under an ℓ_∞ constraint is the sign vector.

But this answer is not quite satisfying, because the ℓ_∞ norm operates on vectors while throwing away tensor information in the gradient. There is a way out, however, by observing that the ℓ_∞ norm satisfies a special property summed up by the slogan "a max of a max is a max." In particular, letting $\operatorname{row}_i(G)$ denote the ith row of a gradient matrix $G \in \mathbb{R}^{m \times n}$, the maximum entry of G can be written in various ways as

$$\max(G) = \|\operatorname{flatten}(G)\|_\infty = \max_i \|\operatorname{row}_i(G)\|_\infty = \|G\|_{1\to\infty}, \tag{2.14}$$

where $\|\cdot\|_{1\to\infty}$ is the ℓ_1 to ℓ_∞ induced operator norm. It is the max of row ℓ_∞ norms, as shown in Proposition 8 of Bernstein and Newhouse [2024a]. So Adam does account for the tensor structure of the gradient, implicitly assigning the ℓ_1 norm to the input space and the ℓ_∞ norm to the output space. Its update, with momentum disabled, is a duality map under the $\|\cdot\|_{1\to\infty}$ operator norm. This matrix-aware update may be one reason why Adam usually outperforms vanilla gradient descent.

One catch, though, is that while Adam accounts for the matrix geometry, it may do so inconsistently. The $\ell_{1\to\infty}$ operator norm supposes an ℓ_1 geometry on the input space and an ℓ_∞ geometry on the output space. But applying Adam to one layer after another would require switching the interpretation of the middle activation space mid-flight. Could a hypothetical optimizer alternate between duality maps under $\ell_{1\to\infty}$ and $\ell_{\infty\to 1}$, passing the type check and perhaps improving performance? Tropp [2004] showed in his PhD thesis that this hope is not tractable: the $\ell_{\infty\to 1}$ norm is NP-hard to compute. Table 2.2 summarizes nine common operator norms. The six tractable norms provide a ruleset for how to choose input and output norms that stitch together into an overall, architecture-aware optimizer. The next optimizer we examine, Shampoo, stitches operator norms together in just this way.

Shampoo as duality under the spectral norm. The Shampoo optimizer [Gupta et al., 2017, 2018], invented at Google, recently came to increased attention after a variant of it won the external tuning track of the *2024 AlgoPerf: Training Algorithms* competition [Dahl et al., 2023]. While the method was originally motivated as a generalization of AdaGrad [Duchi et al., 2011] to tensor spaces, later work casts Shampoo as an approximate second-order method [Anil et al., 2020, Morwani et al., 2024]. But Shampoo—with accumulation disabled—has another, squarely first-order interpretation as duality under the spectral norm.

From	To ℓ_1	To ℓ_2	To ℓ_∞
ℓ_1	Max ℓ_1 norm over columns	Max ℓ_2 norm over columns	Max absolute entry of matrix
ℓ_2	NP-hard	Max singular value	Max ℓ_2 norm over rows
ℓ_∞	NP-hard	NP-hard	Max ℓ_1 norm over rows

Table 2.2: Common operator norms, adapted from Tropp [2004]. Optimizers that apply a consistent interpretation of activation space norms may chain together a compatible sequence of operator norms, but several options are unavailable due to NP-hardness.

To begin, at each time step t and layer L, Shampoo is given by the following updates:

$$L_t = L_{t-1} + G_t G_t^\top, \tag{2.15}$$
$$R_t = R_{t-1} + G_t^\top G_t, \tag{2.16}$$
$$W_{t+1} = W_t - \eta \cdot L_t^{-\frac{1}{4}} G_t R_t^{-\frac{1}{4}}, \tag{2.17}$$

where G_t is the gradient matrix and the two matrices L_t and R_t are called the left and right preconditioners. Shampoo derives its name from the order of operations when showering: before conditioner comes shampoo. With accumulation disabled, the update reduces to

$$W_{t+1} = W_t - \eta \cdot (G_t G_t^\top)^{-\frac{1}{4}} G_t (G_t^\top G_t)^{-\frac{1}{4}} \tag{2.18}$$
$$= W_t - \eta \cdot U_t V_t^\top, \tag{2.19}$$

where Equation (2.19) comes from substituting the reduced singular value decomposition of the gradient $G_t = U_t \Sigma_t V_t^\top$ into Equation (2.18). Notice that there is a direct parallel between Equations (2.12) and (2.13) for Adam and Equations (2.18) and (2.19) for Shampoo.

Therefore Shampoo's update $U_t V_t^\top$ is always a semi-orthogonal matrix. In fact, it is the closest semi-orthogonal matrix to G under the Frobenius norm, as proved in Proposition 4 of Bernstein and Newhouse [2024a]. This fact has given the operation a name, *orthogonalizing* the gradient. Since semi-orthogonal matrices have singular values all 1, projecting the gradient to its nearest semi-orthogonal matrix is a kind of normalization under the spectral norm. But Shampoo's update is not just a normalization $G/\|G\|_{2\to 2}$, which would suppress small singular values. It is a duality map, which increases small singular values.

Theorem 2.1.2. Spectral norm duality orthogonalizes $U\Sigma V^\top \mapsto UV^\top$

Given a matrix $G \in \mathbb{R}^{m\times n}$ with singular value decomposition $G = U\Sigma V^\top$, the spectral norm duality map is $\text{dualize}_{\|\cdot\|_{2\to 2}}(G) = UV^\top$. The solution is unique if G is full rank.

Proof. The relevant alignment measure to maximize is $\langle G, T\rangle = \operatorname{tr} G^\top T$, which multiplies the

matrices entrywise and sums. Since G has singular value decomposition $U\Sigma V^\top = \sum_i \sigma_i u_i v_i^\top$,

$$\arg\max_{\|T\|_{2\to 2}=1} \operatorname{tr} G^\top T = \arg\max_{\|T\|_{2\to 2}=1} \operatorname{tr} \sum_i \sigma_i \, v_i u_i^\top T \tag{2.20}$$

$$= \arg\max_{\|T\|_{2\to 2}=1} \sum_i \sigma_i \, u_i^\top T v_i \leqslant \sum_i \sigma_i = \operatorname{tr} \Sigma, \tag{2.21}$$

where the inequality comes from the spectral norm constraint $\|T\|_{2\to 2} = 1$. And setting $T = UV^\top = \sum_i u_i v_i^\top$ saturates the inequality, as desired. Consequently, the dual norm is $\|G\|_{2\to 2}^\dagger = \operatorname{tr} G^\top U V^\top = \operatorname{tr} \Sigma$.

For uniqueness, if G is full rank then all its singular values are positive, meaning T must select $u_i v_i^\top$ for each i to saturate the inequality. But if G is not full rank then some singular value σ_i is 0, and the corresponding subspace has no uniqueness guarantee since $-u_i v_i^\top$ would work just as well. □

So Shampoo's update projects the gradient to the nearest semi-orthogonal matrix, and this operation can be viewed as a duality map under the spectral norm. Viewing Shampoo as a (smoothed out) spectral duality map grounds the algorithm in prior literature on spectral descent [Carlson et al., 2015a, 2016, Fan, 2017].

To extend duality map updates to a neural network with multiple layers, one can dualize under the max of individual norms over the layers. The max plays a special role in duality maps, akin to an instruction to parallelize:

- A max over the absolute values of entries sets each entry to ± 1 in parallel, as in Adam.
- A max over the singular values sets each singular value to 1 in parallel, as in Shampoo.
- A max over the norms of layers dualizes each layer in parallel, according to its norm.

The final point hints at *modular duality*, a way to unify weight updates across an architecture by selecting norms for each layer and performing steepest descent using a max over norms on the indiviudal weight spaces [Bernstein and Newhouse, 2024b].

One reason to take a max over norms—as proposed by Large et al. [2024]—is that it produces updates in every layer. In contrast, Flynn [2017]'s remarkable early efforts toward duality used an ℓ_1 combination over layers, which only updates one layer at a time.

It is an open question whether the step size from steepest descent is useful in practice. Training pipelines often use learning rate schedules that may obviate the need for a global step size. However, the dual norm is efficient to compute once the duality map is computed: plug the arg max into the max.

And the odd polynomial iterations from Section 1.3 make it fast to compute the spectral norm duality map $UV^\top$ that sets all the singular values to 1. Instead of inverse fourth roots from the Shampoo update $L_t^{-1/4} G_t R_t^{-1/4}$, which can be unstable in lower precisions such as `bfloat16`, this new computational approach enables a new optimizer, called Muon.

2.2 Deriving Muon

The Muon optimizer, originally popular for setting speed records for training NanoGPT [Jordan et al., 2024b], has since been validated at scale with favorable performance compared to AdamW [Liu et al., 2025, Shah et al., 2025]. The theory behind Muon is duality, an application of the broader method of metrized deep learning. Muon falls out of three steps:

1. Select the RMS to RMS norm for every weight matrix.
2. Dualize every gradient using odd polynomial iterations.
3. Turn on momentum for smoother, full-rank gradient estimates.

Having set up duality maps, the Muon training update falls right out:

$$M_{t+1} = \beta \cdot M_t + (1 - \beta) \cdot G_t, \tag{2.22}$$

$$W_{t+1} = W_t - \eta \cdot \text{dualize}_{\|\cdot\|_{\text{RMS}\to\text{RMS}}}(M_t), \tag{2.23}$$

where t is the time step, M_t is the momentum buffer, W_t is the weight matrix, G_t is the gradient matrix, η is the learning rate, and $0 \leqslant \beta < 1$ sets the time scale for the exponential moving average. There are three distinctions between the update rule above and the default setting in the public implementation of Muon:

1. Muon uses Nesterov accelerated momentum, implemented as $\text{dualize}_{\|\cdot\|_{\text{RMS}\to\text{RMS}}}(M_t + G_t)$, because Keller Jordan found that this update performs slightly better empirically.
2. Muon sets each singular value to a random number in the range $[0.7, 1.3]$, rather than to 1. This relaxation allows using odd polynomials that inflate the small singular values faster, as depicted in Figure 2.1.
3. Muon then scales the singular values by $\max(1, \sqrt{d_{\text{out}}/d_{\text{in}}})$, which differs from the pure RMS $\to$ RMS norm factor $d_{\text{out}}/d_{\text{in}}$. The rationale may be to perserve activation norms at initialization, when weights and activations align at random. During training, however, activations and weights may learn to align, suggesting that the pure RMS $\to$ RMS scaling factor $\sqrt{d_{\text{out}}/d_{\text{in}}}$ may be preferable in the long run.

Because it is an architecture-aware optimizer, Muon is designed for linear layer weight matrices. It is not designed for embedding layers, layer normalization parameters, or bias parameters. The metrized deep learning suggests possible alternatives for these parameters:

1. Embedding could use $\text{dualize}_{\|\cdot\|_{1\to\text{RMS}}}$, since its inputs are one-hot vectors with unit ℓ_1 norm and it may be desirable for its outputs to have entries near 1, corresponding to unit RMS norm.
2. An entrywise multiplication parameter vector could use $\text{dualize}_{\|\cdot\|_\infty}$, akin to Adam, since entrywise multiplication changes the RMS norm of an activation vector by at most the maximum entry.

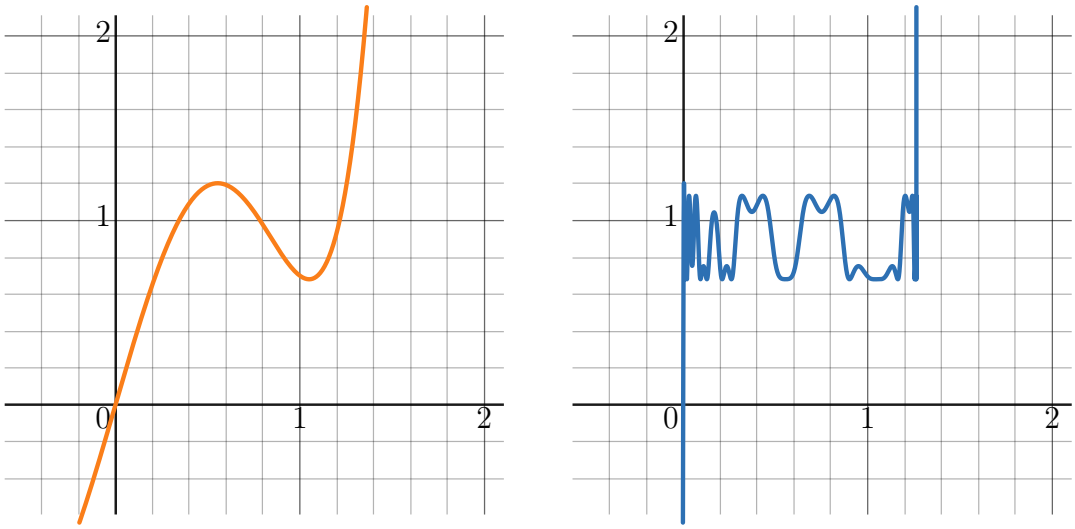

Figure 2.1: Muon approximates spectral norm duality on the gradient—which sends all singular values to 1—by iterating the polynomial $p(x) = 3.4445x - 4.7750x^3 + 2.0315x^5$ five times to rapidly inflate small singular values. The iteration noisily approximates $\text{sign}(x)$. Muon's maximum inflation factor of $485\times$ defines an implicit threshold for when it considers tiny singular values to be noise. Plots adapted from Bernstein [2025].

3. A bias parameter vector could use $\text{dualize}_{\|\cdot\|_{\text{RMS}}}$ akin to standard gradient descent, since adding a bias vector u to an activation vector v can increase RMS norm by at most $\|v + u\|_{\text{RMS}} - \|v\|_{\text{RMS}} \leqslant \|u\|_{\text{RMS}}$.

In practice, Muon uses AdamW for the non-weight-matrix parameters. Perhaps other approaches could yield better optimizers. Metrized deep learning opens up many such research questions, a sentiment captured by Pethick et al. [2025] who explore several norm choices.

2.3 Intriguing properties

Duality causes qualitatively different training behavior because it takes optimization steps that have singular values all 1, for linear weight matrices. We list several phenomena.

Fast and scalable. Figure 2.2 shows that duality-based optimizers train faster than Adam in the simple setting of MLPs on CIFAR-10 [Krizhevsky et al., 2009]. The optimal learning rate transfers across width, since the RMS norm accounts for dimensional scaling.

The weights move! This observation is due to Jeremy Bernstein. A common narrative in deep learning is that the weights cannot stray far from initialization for very wide networks [Lee et al., 2019, Jesus et al., 2021]. Duality changes this story. Figure 2.3 shows that dualized updates do move the weights—even in the Frobenius norm.

Low precision stability. Unlike the inverse fourth matrix powers from the Shampoo update, odd polynomial iterations work well in `bfloat16` precision. This numerical stability further speeds up duality maps in practice and is implemented in the NanoGPT speedrun [Jordan et al., 2024a].

Several open questions remain:

- Can duality-based optimizers be efficiently sharded on to clusters with thousands of GPUs? The Dion optimizer presents a promising answer that uses an alternative to odd polynomial iterations [Ahn and Xu, 2025].

- When Shah et al. [2025] scale up Muon to batch sizes of 16M tokens, what is the role of noise? By iterating its odd polynomial for 5 steps, Muon stakes an implicit claim that below some threshold—around $485 \approx 3.445^5$—the singular values may truly be noise and will not be promoted to 1. But larger batch sizes might reduce the noise scale. If so, could increasing the singular value inflation factor—implicitly lowering the assumption of the noise scale— increase performance?

- What are the right optimization dynamics for the embedding layer? The $\ell_1 \to \ell_{\text{RMS}}$ duality map normalizes each column of the gradient, but for rare tokens, a momentum buffer will add full-size dualized updates to that column of the weight matrix until the buffer decays to exactly zero. In contrast, Adam's two buffers with $\beta_1 < \beta_2$ mitigate the impact of rare tokens. One mitigation is to cap the inflation factor for the embedding gradient columns, similar to Muon's implicit noise scale. Are there other approaches?

- Liu et al. [2025] pointed out that large models trained with Muon exhibit higher singular value entropy than when trained with AdamW. What are the implications? Might Muon interact differently with the low rank simplicity bias of deep neural networks [Huh et al., 2023]?

- If intelligence is compression, as the adage goes, then all the bits should matter equally. Yet on the simple regression task $f(x) = \sin(x) + \sin(\pi x)/1028$ with mean squared error loss, gradient descent will learn the large component first and the small component second, a bias toward higher order bits. Could Muon, which values each singular vector direction equally, learn qualitatively differently?

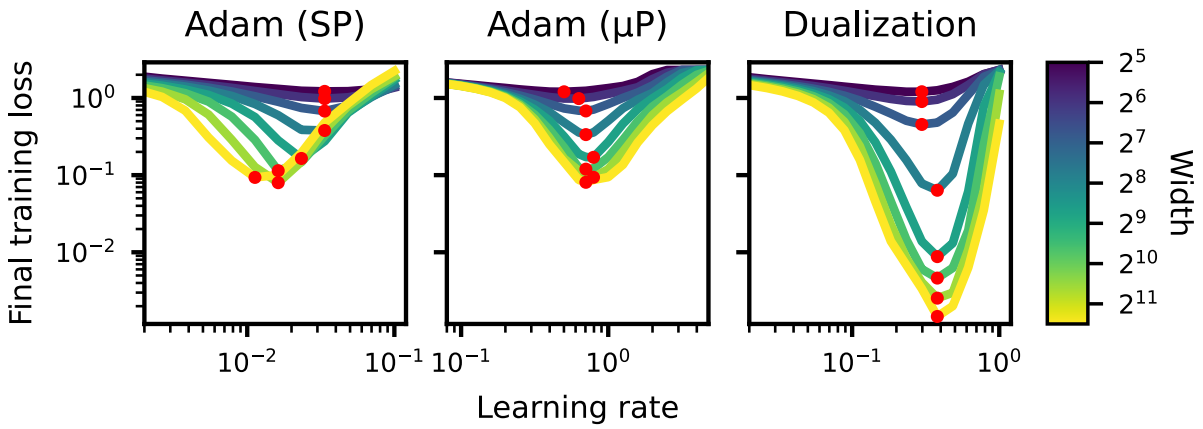

Figure 2.2: **Learning rate transfer with dualization.** To test that duality-based optimizers transfer learning rate, we train an MLP on CIFAR-10 for 20 epochs at a range of widths and learning rates. We plot the final training loss and mark the best learning rate at each width with a red dot. **Left:** In standard parameterization (SP), Adam's optimal learning rate drifts to the left. **Middle**: Maximal update parameterization [Yang and Hu, 2021, μP] mostly corrects this drift. **Right**: Duality-based optimization has a fairly stable optimal learning rate and also reaches much lower loss. More experimental details are available in Appendix A of "Modular Duality in Deep Learning" [Bernstein and Newhouse, 2024b].

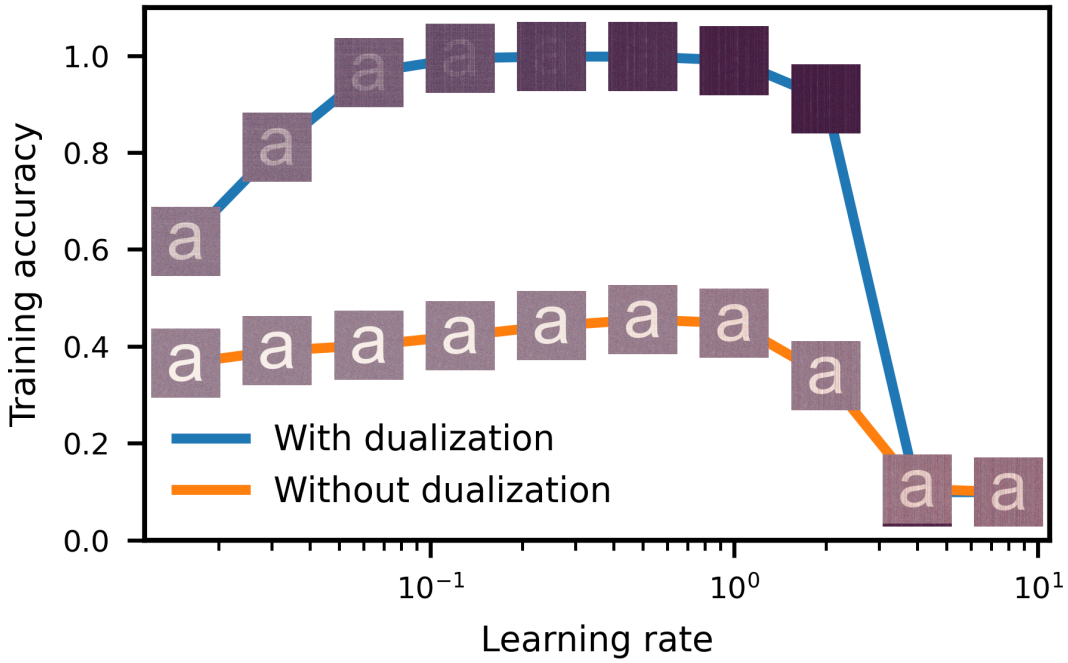

Figure 2.3: **Erasure of watermarked weights.** It is commonly held that the weights stay close to initialization in very wide networks [Lee et al., 2019, Jesus et al., 2021]. To visualize the change in weights, we "watermark" the hidden layer weights of an MLP of width 1024 at initialization by zeroing out matrix entries in the shape of the letter "a". We then train for 1000 steps on CIFAR-10, across ten learning rates. For each run, we plot the final training accuracy along with an image of the learned weight matrix. Not only does dualized gradient descent reach higher training accuracies, but it also "erases" the watermark at the highest stable learning rate, a substantial weight change. More experimental details are available in Appendix A of "Modular Duality in Deep Learning" [Bernstein and Newhouse, 2024b].

Chapter 3

Training transformers with enforced Lipschitz constants

So far the focus of the metrized deep learning method has been toward the controlling the gradients. Yet if all we control are the gradients, the weights may drift, and the model may become highly sensitive to input and weight perturbations. This sensitivity has been linked to pathologies such as vulnerability to adversarial examples, divergent training, and overfitting. Therefore, this chapter applies the metrized deep learning method to enforce norm constraints on the weights, specifically for transformers, where the closest related work, LipsFormer, only enforces constraints at initialization [Qi et al., 2023]. To explore this gap, we propose and benchmark novel, computationally-efficient tools for maintaining norm-constrained weight matrices. Applying these tools, we are able to train transformer models with Lipschitz bounds enforced throughout training. We find that optimizer dynamics matter: switching from AdamW to Muon improves standard methods—weight decay and spectral normalization—allowing models to reach equal performance with a lower Lipschitz bound. Inspired by Muon's update having a fixed spectral norm, we co-design a weight constraint method that improves the Lipschitz vs. performance tradeoff on MLPs and 2M parameter transformers. Our 4-Lipschitz transformer on Shakespeare text reaches validation accuracy 60%. Scaling to 145M parameters, our 600-Lipschitz transformer reaches 21% accuracy on internet text. However, to match the NanoGPT baseline validation accuracy of 39.4%, our Lipschitz upper bound increases to 10^{274}. Nonetheless, our Lipschitz transformers train without stability measures such as layer norm, QK norm, and logit tanh softcapping.

Contribution statement: This chapter adapts material from an upcoming paper. Jeremy Bernstein had the idea to study Lipschitz neural networks from the metrized deep learning perspective. I proposed using odd polynomial iterations to control the singular values of the weights, designed the spectral soft cap method depicted in Figure 1.3, and derived the formula to couple its decay to the learning rate. I contributed to all the experiments in this chapter. I designed the experiment pipeline and ran thousands of CIFAR-10 runs, thousands of Shakespeare transformer runs, and around fifty 145M parameter runs at the NanoGPT scale. Several collaborators joined near the end of the project: Andrii Zahorodnii investigated adversarial robustness (Figure 3.2); Preston Hess invented the spectral hammer method with Andrew Hutchison, compared methods (Figure 3.3), and wrote up several sections of the

paper; Franz Cecista contributed innumerable speedrun experiments and fixed an error in the Lipschitz constant calculation; Jeremy Bernstein and Phillip Isola advised throughout.

3.1 Introduction

Lipschitz constants for neural networks—intuitively, bounds on the sensitivity of each component and of the overall model to input perturbations—are of interest for their effect on generalization and robustness [Bartlett et al., 2017, Tsuzuku et al., 2018] and for applications such as differential privacy [Béthune et al., 2024]. Seminal work [Arjovsky et al., 2016, Cisse et al., 2017, Yoshida and Miyato, 2017, Anil et al., 2019] enforces Lipschitz constants beyond initialization for MLPs, RNNs, and GANs, but for transformers, the closest related work, LipsFormer [Qi et al., 2023], does not constrain weights while training. Without constraints, large-scale transformer training may encounter instabilities, which has been attributed to attention and output logits growing too large [Wortsman et al., 2023, Dehghani et al., 2023]. Can enforced Lipschitz constants benefit transformers, too? Specifically, we ask:

Can transformers with small, enforced Lipschitz bounds perform well?
How does the weight constraint method affect the Lipschitz vs. performance tradeoff?

Enforcing Lipschitz bounds on a transformer is challenging because transformers include components that are not globally Lipschitz, such as self-attention [Kim et al., 2021]. We build on Large et al. [2024] who, similar to LipsFormer, enable Lipschitz continuity by reparameterizing residual connections and modifying self-attention; however, the full story is elusive. LipsFormer goes further than Large et al. [2024] to eliminate layer norm [Ba et al., 2016], but the official implementation may make Lipschitz bounds impossible by setting $\epsilon = 0$ in QK norm [Henry et al., 2020]. In contrast, we remove activation normalization to explore whether training can proceed with no stability measures.

To develop a toolkit for training transformers with an enforced Lipschitz constant, in Section 3.2 we compare several methods for constraining weight norm. Surprisingly, we find that optimizer choice matters: standard methods such as weight decay [Krogh and Hertz, 1991] and spectral normalization [Yoshida and Miyato, 2017] improve a Lipschitz vs. performance tradeoff more with Muon [Jordan et al., 2024b] than with AdamW [Loshchilov and Hutter, 2019]. We see improvement under Muon for MLPs trained on CIFAR-10 [Krizhevsky et al., 2009] and corroborate it on 2M parameter transformers trained on Shakespeare text [Karpathy, 2022].

Beyond standard methods, we are inspired by a property of Muon—its weight updates have small, known spectral norm—to design a weight constraint method called *spectral soft cap*, which enforces a desired maximum spectral norm $\sigma_{\max}$ by approximating the map $\sigma \mapsto \min(\sigma_{\max}, \sigma)$ on all singular values σ in parallel by iterating odd polynomials on the weights. Section 3.2.1 proves that spectrally capping the singular values bounds weight norm when training with Muon; we provide no theoretical guarantee for AdamW because the spectral norm of its update is not controlled. For AdamW, we explore a second technique that may be better suited to low stable rank updates, although we do not provide provable guarantees. At every step, this technique finds the largest weight singular value and sets it to $\sigma_{\max}$. In

analogy with a hammer that strikes the nail that sticks out the most, we call this technique *spectral hammer*. Our experiments suggest that the most effective combination is Muon with spectral normalization or spectral soft cap, while from Adam the only technique that elicits a competitive performance vs. Lipschitz tradeoff is spectral hammer.

In Section 3.3, we scale up enforced weight constraint methods to the NanoGPT speedrun benchmark [Jordan et al., 2024a], training 145M parameter transformers to competitive performance without layer norm or QK norm. We train a 600-Lipschitz transformer to 21.2% validation accuracy, compared to the non-Lipschitz baseline of 39.4% validation accuracy. However, to reach a competitive accuracy of 38.2%, our global Lipschitz upper bound becomes astronomical at 10^{127}. While Fazlyab et al. [2019] describe ways to tighten Lipschitz bounds, inspecting the maximum activation norms reveals that our model operates far from the worst case. On a particular batch of 393K tokens, the non-Lipschitz baseline has maximum activation entry 148,480 while the 10^{127}-Lipschitz transformer has maximum activation entry 96.5. Empirically small activations in Lipschitz-constrained transformers may present an opportunity for low-precision training and inference.

Our contributions are as follows:

- We train transformers with enforced Lipschitz constraints up to 145M parameters, including a 600-Lipschitz transformer with 21% accuracy on FineWeb10B internet text and a 4-Lipschitz transformer with 60% accuracy on Shakespeare text.
- We present evidence that weight decay and spectral normalization yield greater benefits when trained with Muon compared to AdamW, matching accuracy with lower Lipschitz bound. We verify standard robustness properties hold when training with Muon.
- We introduce two weight norm constraint techniques: *spectral soft cap* and *spectral hammer*. Out of weight regularization methods for AdamW, spectral hammer elicits the most competitive Lipschitz-constrained performance. For Muon, we prove spectral soft cap bounds weight norm and find that it performs similarly or slightly better than spectral normalization.

3.2 Weight norm constraints to enforce a Lipschitz constant

A function $f(x)$ has Lipschitz constant K under a norm $\|\cdot\|$ if it satisfies $\|f(x_1) - f(x_2)\| \leqslant K \cdot \|x_1 - x_2\|$ for all inputs x_1, x_2. For neural networks, the most common operation is matrix multiplication which has ℓ_2 Lipschitz constant equal to the spectral norm of the weight matrix. Constraining the spectral norm of weight matrices is not new, with past work primarily exploring weight decay, spectral normalization, and orthogonal constraints [Krogh and Hertz, 1991, Yoshida and Miyato, 2017, Miyato et al., 2018, Gouk et al., 2020, Su, 2024]. These methods have been tested with the AdamW optimizer and have shown benefits for generalization and adversarial robustness [Bartlett et al., 2017, Tsuzuku et al., 2018]. The Muon optimizer introduces new possibilites by ensuring small, fixed-norm weight

updates. Inspired by this property, we revisit existing methods and develop new methods for constraining weights. We ask the question:

What is the best way to enforce weight norm constraints throughout training?

We compare seven methods based on how well they 1) maintain high performance, 2) enforce weight norm constraints, and 3) trade off performance with a Lipschitz bound. To summarize our conclusions, we find that the Muon optimizer achieves lower Lipschitz constants and better performance compared to AdamW. Among the constraint methods, our experiments suggest that spectral soft cap, spectral hard cap, and spectral normalization meet these criteria best.

Muon enables hard weight constraints. Unlike in AdamW, the weight update norm in Muon is bounded by the learning rate—if its orthogonalizing polynomial never exceeds 1. We follow [You, 2025] to ensure this property in our experiments. Pethick et al. [2025] noted that bounded weight update spectral norm upgrades weight decay with parameter λ to enforce a *strict* spectral norm constraint of $1/\lambda$. The reason is that an equilibrium occurs between the update step and weight decay when the weight norm w satisfies $w = w(1-\lambda\eta)+\eta$ for learning rate $\eta > 0$. See Section 3.5.1 for details. We hypothesize that this property may explain our evidence that Muon, compared to AdamW, improves the Lipschitz vs. performance tradeoff for standard methods such as weight decay.

A spectral generalization of weight decay. Weight decay can be viewed as a special case of an *odd polynomial iteration* applied to the weights. Odd polynomials are special because they act directly on the singular values: $p(U\Sigma V^\top) = Up(\Sigma)V^\top$, where $U\Sigma V^\top$ is a singular value decomposition. The odd polynomial for weight decay is $p(x) = (1-\eta\lambda)x$, where η is the learning rate and λ is the weight decay. Cisse et al. [2017] explored an orthogonalizing polynomial $p(x) = (1+\beta)x - \beta x^3$, but Miyato et al. [2018] note that pressuring all singular values toward one limits information in the spectrum. Their method, *spectral normalization*, enforces norm constraints while allowing singular values less than 1 [Gouk et al., 2020], but normalization accomplishes the constraint by scaling down the entire spectrum. This global effect motivates a more targeted approach : penalizing only the singular values that are too large, leaving smaller ones untouched. For a desired maximum norm $\sigma_{\max} \geqslant 0$, an idealized penalty would apply $\min(\sigma_{\max}, \sigma)$ to the singular values, but exactly computing the SVD is slow. Odd polynomial iterations serve as a fast and effective approximation. We contribute a family of such approximations called *spectral soft cap* that contains weight decay as a special case. The derivation and discussion is in Section 3.5.1.

3.2.1 Methods for controlling weight norm

We are interested in controlling the RMS $\to$ RMS operator norm—a rescaled spectral norm—which has emerged as natural for deep learning [Yang et al., 2023, Bernstein and Newhouse, 2024b]. Unit RMS $\to$ RMS norm is equivalent to a spectral norm of $\sqrt{d_{\mathsf{out}}/d_{\mathsf{in}}}$ for a weight matrix $W \in \mathbb{R}^{d_{\mathsf{out}} \times d_{\mathsf{in}}}$. In what follows, we denote the principal singular vector subspace with singular value $\sigma_1 \geqslant 0$ by $\sigma_1 u_1 v_1^\top$, computed via power iteration. We briefly review some known methods to constrain weight norm, then introduce two new methods called spectral capping and spectral hammer.

Weight decay, or Frobenius norm regularization, maps $W \mapsto (1 - \lambda\eta)W$ where $\lambda > 0$ is the decay parameter and $\eta > 0$ is the learning rate, guaranteeing a norm bound in conjunction with Muon.

Spectral weight decay, or spectral norm regularization, targets only the top singular value, mapping $W \mapsto W - \lambda\sigma_1 u_1 v_1^\top$ where $\lambda > 0$ is the decay parameter [Yoshida and Miyato, 2017, Su, 2024].

Spectral normalization, originally introduced in GAN training [Miyato et al., 2018], guarantees a spectral norm bound by mapping $W \mapsto \frac{W}{\sigma_1}$.

Stiefel manifold projection pressures all singular values toward 1 using an odd polynomial iteration $W \mapsto p(W)$, leaving open the choice of polynomial p. We follow You [2025] whose polynomial converges very rapidly rather than the polynomial from [Cisse et al., 2017]. Although Stiefel manifold projections usually refers to projections for rectangular matrices, with a slight abuse of notation we use it to describe this operation on both square and rectangular matrices.

We extend these ideas with two new methods:

Spectral hammer is similar to spectral weight decay, but sets the top singular value to $\sigma_{\max}$ by mapping $W \mapsto W + (\sigma_{\max} - \sigma_1)u_1 v_1^\top$—so to speak, a hammer that strikes the nail that sticks out the most [Hess and Hutchison, 2023]. Spectral hammer does not guarantee the spectral norm stays below $\sigma_{\max} > 0$ because multiple singular vectors may increase per update. Spectral hammer is better suited to low stable rank weight updates as are common in Adam [Zhao et al., 2024]. Muon's update is always high stable rank.

Spectral capping is co-designed for Muon's high stable rank update, smoothly approximating the map $\sigma \mapsto \min(\sigma_{\max}, \sigma)$ for all singular values in parallel. Rather than rely on costly SVDs, it uses an odd polynomial approximation. The primary variant we experiment with is called *spectral soft cap* because it applies a loose approximation $p_2(p_1(x))$, where $p_1(x) = x - \alpha x^3$ and $p_2(x) = x + \alpha x^3$ with strength parameter $\alpha \geqslant 0$, as depicted in Figure 1.3 in Section 1.3. To incorporate weight decay as a special case, we may first apply $p_0(x) = (1 - \lambda\eta)x$. This composition is designed to decay a singular value very little when $\sigma \ll \sigma_{\max}$, but when $\sigma = \sigma_{\max}$ to decay it as strongly as necessary to counteract Muon's known update norm, strictly enforcing a weight norm bound. When the learning rate is scheduled, finding the minimal $\alpha \geqslant 0$ that bounds the weight norm avoids accumulating error.

Theorem 3.2.1. Spectral soft cap bounds spectral norm.

Given a desired maximum spectral norm $\sigma_{\max} \geqslant 0$, learning rate $\eta \geqslant 0$, weight decay $\lambda \geqslant 0$, and weight matrix with bounded norm $\|W\|_* \leqslant \sigma_{\max}$, there is a minimal $\alpha \geqslant 0$ such that Muon's update step followed by spectral soft cap preserves $\|W\|_* \leqslant \sigma_{\max}$. Calculating α involves solving the roots of a quartic polynomial.

Section 3.5.1 gives the proof. A second variant of spectral capping is *spectral hard cap*, which uses four quintic polynomials that closely approximate $\min(\sigma_{\max}, \sigma)$, as depicted in Figure 3.4 in Section 3.5.1. Because this approximation is fixed, errors can compound later in training when the learning rate is scheduled to 0.

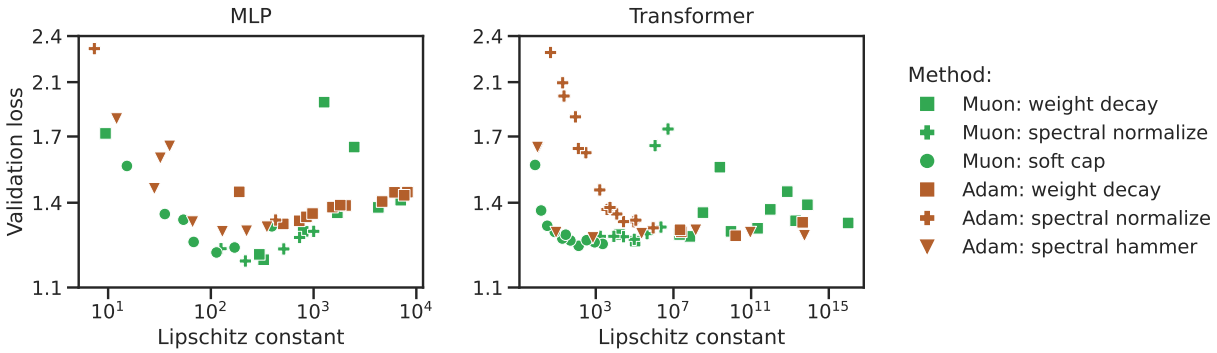

Figure 3.1: **Using Muon instead of AdamW improves the Lipschitz vs. performance tradeoff for standard weight regularization techniques.** We train 1600 MLPs on CIFAR-10 (left) and 800 transformers on Shakespeare text (right), varying the optimizer and weight constraint method. Weight decay and spectral normalization reach better loss with lower Lipschitz constant when using Muon. Two weight constraint methods that we design—spectral soft cap and spectral hammer—are also promising. See Section 3.5.4 for experimental details.

Together, these methods cover a range of tradeoffs between strict norm enforcement, preserving the spectrum, and computational efficiency. There may be other, better approaches, and we think exploring alternatives is an exciting direction. Note that spectral soft cap and spectral hard cap are designed to be compatible with Muon and therefore were not applied to tests with AdamW.

3.2.2 AdamW and Muon: comparing weight constraint methods

In Figure 3.1, we run a sweep to map the tradeoff frontier between validation loss and Lipschitz constant across our methods. The Lipschitz constants are calculated with respect to the RMS → RMS operator norm (Section 3.2.1). For a ReLU MLP, the Lipschitz constant is the product of the RMS → RMS norms of its weights. For a transformer, the Lipschitz constant is calculated as described in Section 3.3.2. Muon consistently achieves both lower validation loss *and* lower Lipschitz constants than AdamW, a trend that holds for MLPs on CIFAR-10 and transformers on Shakespeare text. This result motivates our choice to adopt Muon for larger-scale experiments. Spectral normalization and spectral soft cap appear to make the most efficient use of a Lipschitz budget. Spectral hammer–which is designed for AdamW's low stable rank weight updates–shows competitive performance but does not enforce a Lipschitz constant, limiting its reliability for settings where constraint enforcement is critical. The sweep includes 2400 training runs, where for each method we display only the best validation loss per bin of Lipschitz constant.

3.2.3 Adversarial robustness of Lipschitz networks

Prior work [Cisse et al., 2017, Huang et al., 2021] suggests that a neural network's adversarial robustness is related to its Lipschitz constant, making Lipschitz control a potential path for

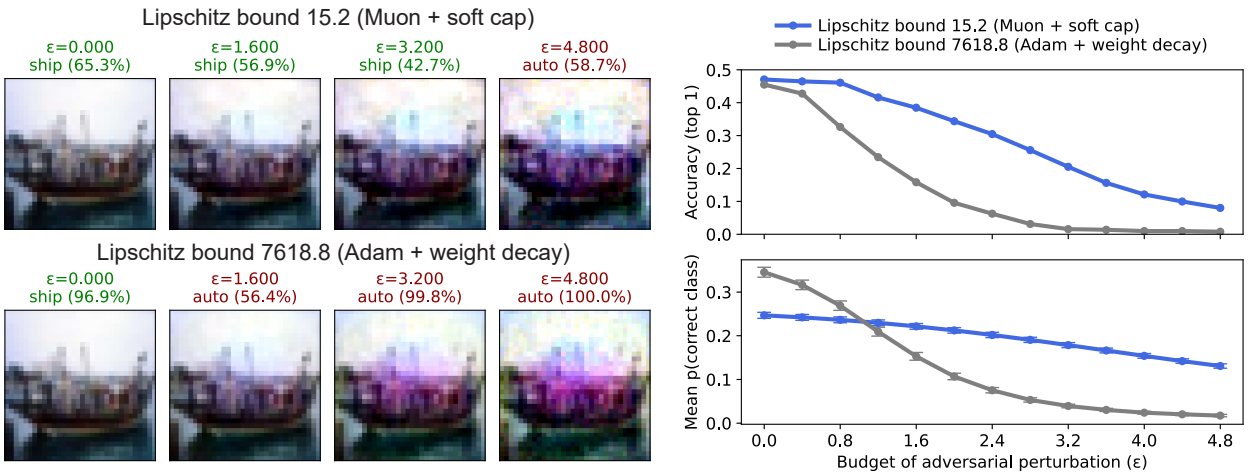

Figure 3.2: **Networks trained with Muon and spectral soft cap have lower Lipschitz bounds and are more adversarially robust.** Lower Lipschitz constants have been linked to greater adversarial robustness [Cisse et al., 2017, Huang et al., 2021]. To assess this effect in our models, we train a CIFAR-10 MLP with a Lipschitz constant of 15.2 (Muon + spectral soft cap), which matches the 45% clean accuracy of a baseline model (AdamW + weight decay) that has a a higher Lipschitz constant of 7618.8. Left: Example adversarial attacks with different ℓ_2 budget $\epsilon \geqslant 0$ for the perturbation. Top right: We quantify adversarial robustness across 2000 test images by the top-1 accuracy as a function of ϵ. The Lipschitz-constrained network trained with Muon and spectral soft cap maintains a higher accuracy for larger values of ϵ. Bottom right: the mean probability of the correct class in the Lipschitz-constrained network starts lower than that of the baseline model, but degrades slowly under increasing ϵ. By contrast, the baseline model peaks higher for $\epsilon = 0$, but drops off sharply.

developing models with high certified accuracy. We confirm this relationship holds for MLPs trained with Muon and spectral soft cap. Figure 3.2 compares the accuracy of two trained MLPs under various budgets of adversarial perturbation ϵ. The pair depicted has matching accuracy $\approx 45\%$ but different Lipschitz constants: 15.2 (Muon + spectral soft cap) vs. 7618.8 (AdamW + weight decay).

While both models achieve similar accuracy without perturbation, the Lipschitz-constrained MLP (quantified in the RMS $\to$ RMS operator norm, Section 3.2.1) trained with Muon exhibits a smoother dropoff in accuracy and confidence across the 2000 CIFAR-10 test set images as the adversarial attack increases its l_2 perturbation. Larger values of ϵ are required to fool the Lipschitz-constrained network (Figure 3.2, left).

3.2.4 Comparing weight constraint methods within Muon

In Figure 3.3, we use Muon alongside all seven weight constraint methods to identify which approaches best meet three goals: 1) enabling us to define a Lipschitz constant before training, 2) enforcing that constant throughout training, and 3) matching or exceeding the performance of standard weight decay. We tested each weight constraint method on a 3-layer MLP with hidden dimension 256, trained with Muon on CIFAR-10 classification. Full ex-

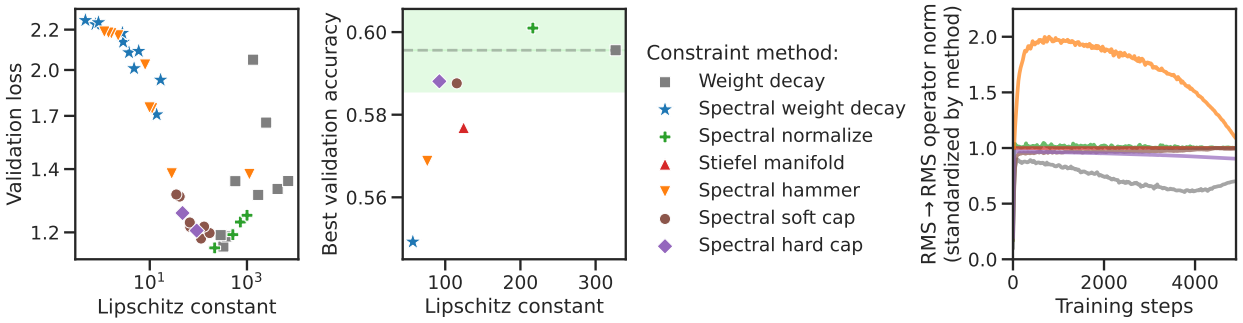

Figure 3.3: **Left: Weight constraint methods designed for Muon lie on the frontier of the Lipschitz vs. loss tradeoff.** Each point shows the lowest validation loss achieved at a given Lipschitz constant across all MLP runs on CIFAR-10. In the low loss regime, staying on the tradeoff frontier requires either spectral normalization or spectrally capping the singular values. **Middle: Spectral normalization and spectral capping match baseline with lower Lipschitz constant on CIFAR-10.** Each method comes within 1% accuracy (shaded green region) and has lower Lipschitz constant. **Right: RMS → RMS operator norm of hidden layers over training for the best networks.** The norm is standardized so that the weight constraints all target 1. Spectral normalization and Stiefel manifold projection strictly meet this target. Weight decay, spectral soft cap, and spectral hard cap stay below the target, while spectral hammer fails to remain constrained.

perimental details are in Section 3.5.4.

In Figure 3.3 (left panel), we see that spectral normalization, spectral soft cap, and spectral hard cap define the frontier of the Lipschitz vs. validation loss tradeoff. In Figure 3.3 (middle), we select the parameter settings with the highest validation accuracy for further analysis. Spectral normalization, spectral soft cap, and spectral hard cap are the only methods to reach within 1% validation accuracy of the baseline (Muon with weight decay). Other methods reached within 4% accuracy.

Figure 3.3 (right panel) visualizes how the norm of the hidden weight matrix evolves during training. All methods but spectral hammer retain the weight norm at or under its target. In contrast, spectral hammer exceeds its target but begins to converge near the end, indicating that it could be a promising technique under Muon, too, on longer training runs that also schedule learning rate to 0. However, within our 50-epoch window, it fails to reliably control the Lipschitz constant. Spectral weight decay does not enforce predefined Lipschitz constants, so it is not plotted.

We select spectral soft cap, spectral hard cap, and spectral normalization for our transformer training experiments. While all weight constraint methods could potentially benefit from combining with standard weight decay, we want to isolate the effect of each one.

3.3 Transformers with enforced weight constraints

To develop a toolkit for training Lipschitz-constrained transformers, we begin in Section 3.3.1 with a discussion of how to handle residual connections and self-attention. In Section 3.3.2,

we explain how to calculate a Lipschitz bound for a transformer. In Section 3.3.3, we find that spectral normalization and spectral soft cap perform well on 2M parameter transformers trained on Shakespeare text. In Section 3.3.4, we scale up to a transformer with 145M parameters trained on FineWeb10B internet text [Penedo et al., 2024]. To combat the undertuned baseline problem, we begin from the competitive benchmark of NanoGPT speedrunning [Jordan et al., 2024a].

3.3.1 Breaking the multiplication barrier?

A major triumph of Large et al. [2024] is to create transformers with a depth-independent Lipschitz constant. However, their approach relies on activations having unit RMS norm. Because we will later relax this constraint, our transformers will typically *not* have a depth-independent Lipschitz bound. Nonetheless, we use their two architectural suggestions.

Reparameterizing residual connections. The classic residual connection from He et al. [2015] defines the update $x + \mathsf{block}(x)$, but even a 1-Lipschitz block can exponentially increase the Lipschitz constant: $x + \mathsf{identity}(x)$ doubles at every layer. Large et al. [2024] use a convex combination

$$\frac{N-1}{N} \cdot x + \frac{1}{N} \cdot \mathsf{block}(x) \tag{3.1}$$

to break this multiplication barrier, where N is the number of layers. The residual connection will be 1-Lipschitz if the block is 1-Lipschitz. See Proposition 4 of Large et al. [2024]. Our experiments use this convex parameterization. However, the bound breaks down if activation norms exceed 1. We are unable to attain high performance without relaxing the 1-Lipschitz constraint. Therefore we do not fully break the multiplication barrier: deeper networks can accrue astronomical Lipschitz bounds.

Attention with $1/d$ scaling. The original multihead attention of Vaswani et al. [2017] has no global Lipschitz bound [Kim et al., 2021]. In a footnote, Vaswani et al. [2017] indicate that they chose $1/\sqrt{d}$ scaling because two random vectors u, v with mean 0 and variance 1 will have a dot product $u \cdot v$ with mean 0 and variance the dimension d. But key and query vectors may align more than at random. Perfect alignment would suggest $1/d$ scaling. Large et al. [2024] prove that $1/d$ scaling in the softmax, together with a factor of $\frac{1}{3}$, makes functional attention,

$$\frac{1}{3} \times \mathrm{softmax}\left(\frac{QK^\top}{d}\right) V, \tag{3.2}$$

1-Lipschitz if the input norms are 1. The Lipschitz constant is with respect to the $\|\cdot\|_{\mathrm{RMS}\infty}$ norm, the max RMS norm of any token. See Proposition 7 of Large et al. [2024]. Finally, while Large et al. [2024] retains layer normalization, we remove it so that every operation is Lipschitz continuous.

3.3.2 Calculating the Lipschitz constant of a transformer

To test whether transformers with small Lipschitz constant can perform well, we would like to know what weight norm to enforce to end up with a desired Lipschitz bound. This

section sketches an algorithm for bounding the Lipschitz constant of a transformer given its weight norms; the full algorithm is in Section 3.5.2. Our bound tightens Theorem 2 from LipsFormer [Qi et al., 2023] by accounting for each weight norm individually, rather than working with the max weight norm across residual blocks. Fazlyab et al. [2019] suggest ways to tighten a global bound like ours, which we leave to future work. To bound self-attention, we extend Proposition 7 from the modular norm paper [Large et al., 2024] to when the input is no longer unit norm, an essential concession to reach different regions of the Lipschitz vs. performance tradeoff.

Our Lipschitz bounds are with respect to the max RMS norm over token positions, denoted $\|\cdot\|_{\infty\text{RMS}}$.

Step 1: Bound activation norms. Our Lipschitz bound for rescaled dot-product attention relies on the activation norms remaining bounded. Therefore, the algorithm begins by computing a per-layer bound for the maximum $\|\cdot\|_{\infty\text{RMS}}$ norm of activations. This bound comes from combining residual connections with per-block maximum activation increases, found for an MLP by multiplying its two weight norms and for attention by multiplying its W_V and W_O weight norms. Our MLPs can slightly decay activation norm because we scale GeLU down by its maximum derivative, GeLU/1.1289.

Step 2: Bound the Lipschitz constant. Suppose the Lipschitz constant prior to reaching a particular layer is L. The Lipschitz constant after a residual connection is at most $(1-\alpha)L + \alpha \cdot L \cdot L_{\text{block}}$. To compute a block's Lipschitz constant L_{block}, for MLPs we calculate $\|W_{\text{in}}\|_{\text{RMS}\to\text{RMS}} \cdot \|W_{\text{out}}\|_{\text{RMS}\to\text{RMS}}/1.1289$ due our rescaled GeLU, and for attention we use the formula from Section 3.5.2.

3.3.3 Shakespeare Transformer

Before scaling to NanoGPT, we explore the Lipschitz vs. performance tradeoff for transformers at a smaller scale. To narrow our aim, we want a transformer that has small Lipschitz constant *at every layer*. Béthune et al. [2022] comments that any L-Lipschitz classifier can be made 1-Lipschitz by dividing the logits by L, but we do not downscale our final logits, although scaling temperature during training could have beneficial effects [Agarwala et al., 2023]. All experimental details are available in Section 3.5.4.

We can train a competitive 4-Lipschitz Shakespeare transformer. Our 4-Lipschitz transformer reaches validation loss $1.29 < 1.47$ from the baseline in Karpathy [2022], although the baseline may not be tuned carefully. Our model has dimension 256, depth 3, and was trained for 2000 steps with Muon; the baseline has dimension 384, depth 6, and was trained for 5000 steps with AdamW. Ours uses no layer normalization. To achieve this performance requires relaxing the maximum weight norm $\sigma_{\max}$ to around 2. The best validation loss from our sweep was 1.20 with a 131-Lipschitz transformer. While gains may be attributed to hyperparameters or optimizer choice, we nonetheless end up with one construction of our aim: a performant transformer with a small, enforced Lipschitz bound.

3.3.4 Scaling to NanoGPT

We validate our method by training a transformer with 145M parameters on the NanoGPT speedrun benchmark [Jordan et al., 2024a], which is built on top of [Karpathy, 2022]'s re-

Transformer Architecture	Lipschitz Bound	Number of Steps	Weight Constraint	Validation Accuracy (↑)	Validation Loss (↓)	Activation Max Entry
Baseline (speedrun)	∞	1,770	none	**0.394**	**3.280**	148,480
Baseline (Karpathy)	∞	17,000	none	-	**3.280**	-
LipsFormer	10^{130}	1,770	none	0.301	4.130	61.1
Ours ($\sigma_{\max} = 1$)	600	1,770	spectral normalize	0.212	5.047	14
Ours ($\sigma_{\max} = 8$)	10^{118}	1,770	spectral soft cap	**0.365**	**3.569**	54.25
Ours ($\sigma_{\max} = 8$)	10^{56}	1,770	spectral hard cap	0.354	3.670	107
Ours ($\sigma_{\max} = 8$)	10^{130}	1,770	spectral normalize	0.360	3.607	100
Ours ($\sigma_{\max} = 16$)	10^{274}	7,080	spectral normalize	**0.395**	**3.280**	160

Table 3.1: Transformers with enforced Lipschitz constraints can match performance on NanoGPT. The NanoGPT speedrun is a competitively tuned benchmark building on Karpathy's original replication of GPT-2 [Karpathy, 2022, Jordan et al., 2024a]. With the speedrun baseline as a starting point, we substitute Lipschitz transformer components and constrain weight norms to not exceed a given $\sigma_{\max} \geqslant 0$. Unlike LipsFormer [Qi et al., 2023], our weight constraint methods enforce a Lipschitz constant chosen prior to training. To demonstrate, we train a 600-Lipschitz transformer to 21.2% accuracy. However, reaching accuracy on par with the baseline increases the Lipschitz bound to 10^{274}, computed as in Section 3.3.2. Our Lipschitz bounds may be loose, as suggested by the small maximum activation that we observe across a batch of 393K tokens. Per-run loss variance is 0.0008.

production of GPT-2. The baseline is competitively optimized to reach validation loss 3.28 in the shortest wallclock time. The latest record as of February 1, 2025 requires only 1770 training steps, or 3 minutes of training on an 8xH100, and achieves a validation accuracy of 39.4%. We implement our methods on top of this baseline while keeping all other training methods fixed. We report the validation loss and the validation accuracy as primary comparison metrics.

To implement our method, we first remove speedrun-specific optimizations such as skip connections and learnable scale parameters. Next we remove the layer norms used for pre-normalization, the logit tanh softcap, and the QK norms in the attention layers. We also replace the ReLU^2 activations with $\mathsf{GeLU}/1.1289$—making the activation function Lipschitz continuous. We reparametrize the residual connections and attention layer as in Equations (3.1) and (3.2). At initialization, we project the linear weights to be semi-orthogonal and normalize the embeddings to have RMS norm 1. Finally, we explicitly extend beyond the closest related work, LipsFormer [Qi et al., 2023], by enforcing weight norm constraints throughout training: we cap the RMS norm of embeddings to 1 and apply one of the weight constraint methods from Section 3.2.1 to all other weights after every training step.

Table 3.1 summarizes our 145M parameter scale transformer results. In contrast to LipsFormer, our method guarantees a Lipschitz upper bound specified prior to training. An important lever is the maximum RMS $\to$ RMS norm $\sigma_{\max} \geqslant 0$ we allow for the linear layers. Smaller $\sigma_{\max}$ correspond to smaller Lipschitz bounds but may lower performance, and vice versa. Two other variables that affect the Lipschitz constant are the attention logit scale and final logit scale. Using spectral normalization with $\sigma_{\max} = 1$ and a final logit scale of 8 results

in a 600-Lipschitz transformer that attains validation loss 5.047 and accuracy 21.2%. No activation in this model exceeds RMS norm 1, which could enhance stability during training. Setting $\sigma_{\max} = 16$ enables reaching parity with the speedrun performance but increases the Lipschitz bound to 10^{276}. This setting trains for $4\times$ as many steps as the current speedrun record (but $2.4\times$ fewer than Karpathy's baseline).

3.4 Discussion

Despite high Lipschitz upper bounds, our transformers on NanoGPT exhibit low maximum activation entries (50-110) compared to the baseline (148K). Perhaps as a result, our models train stably without standard measures including layer norm, QK norm, or tanh logit soft-capping. In future work, we are interested to test whether these low maximum activations hold potential for low-precision training and inference. We also wonder whether training would remain stable at larger scales.

For MLPs and small transformers, we find that using Muon improves the Lipschitz vs. performance tradeoff. Out of weight constraint methods we test, *spectral normalization*, *spectral soft cap*, and *spectral hard cap* compare favorably to standard weight decay. Perhaps surprisingly, on both CIFAR-10 and Shakespeare data, we achieve our best loss with Lipschitz-enforced models, potentially representing a training speed benefit.

Our work has several limitations. We did not find a principled way to select weight norm, final logit scale, and attention logit scale hyperparameters, instead relying on sweeps. Our Lipschitz bound also increases rapidly as depth increases, unless we constrain weights to unit norm. A different architecture, or insight beyond a global Lipschitz bound, could make progress on this problem.

In conclusion, this paper develops a method for training transformers with an enforced Lipschitz constant throughout training, extending earlier efforts focused on different architectures or only constraints at initialization. Lipschitz-certified transformers may be of interest for domains such as privacy, control, adversarial robustness, and low-precision training. Although training speed benefits fade in our NanoGPT speedrun experiments, we wonder whether at this scale Lipschitz-enforced training can be made faster than standard training.

3.5 Proofs and experimental details

3.5.1 Coupling spectral cap to learning rate

This section proves Section 3.2.1: spectral soft cap bounds weight norm. We will derive a strength parameter that couples to the learning rate, because otherwise using odd polynomial approximation—rather than the ideal map $\min(\sigma_{\max}, \sigma)$—accumulates errors when the learning rate falls below the approximation gap. We will prove the theorem using an equilbrium analysis: solving for the fixed point of a contractive map.

To warm up, there is a special case in which weight decay provably bounds the weight norm during training. It happens when the norm of the update is bounded by the learning rate, $\|\Delta W\| \leqslant \eta$. To see why, suppose weight decay is applied at every step of training

and is linearly coupled to the learning rate as $\lambda\eta$ for some constant $\lambda > 0$. Subadditivity of norms guarantees $\|W + \Delta W\| \leqslant \|W\| + \|\Delta W\| \leqslant \|W\| + \eta$. The weights cannot increase further when the decay and the learning rate are in equilbrium: $\|W\| \cdot (1 - \lambda\eta) + \eta = \|W\|$. The equilibrium occurs at $\|W\| = 1/\lambda$. Therefore $1/\lambda$ is the maximum weight norm possible under standard weight decay, if the update norm is bounded above by η. Pethick et al. [2025] noted the same phenomenon.

To prove Section 3.2.1, we conduct a general equilibrium analysis to consider three effects: weight decay, optimizer step, and weight projection. The net effect of the three effects should decrease every singular value $\sigma \geqslant \sigma_{\max}$. Let the weight decay λ be coupled to the learning rate η. Suppose the weight update is constrained to have norm $\|\Delta W\| \leqslant \eta$. Let $p(x)$ be an odd polynomial. Equilbrium occurs when

$$p(x \cdot (1 - \lambda\eta) + \eta) \leqslant x. \tag{3.3}$$

In words, apply weight decay, apply an optimizer step that will not increase singular values by more than η, and then apply the odd polynomial. If the singular value does not increase for all $x \in [0, \sigma_{\max}]$, then the weight norm will never exceed $\sigma_{\max}$.

Recall that $p_1(x) = x - \alpha x^3$ and $p_2(x) = x + \alpha x^3$.

When $p(x) = p_2(p_1(x))$, the equilibrium condition Equation (3.3) becomes

$$f(\alpha) = \left(k - \alpha k^3\right) + \alpha \left(k - \alpha k^3\right)^3 - \sigma_{\max} \leqslant 0, \tag{3.4}$$

$$\text{where} \quad k = \sigma_{\max} \cdot (1 - \lambda\eta) + \eta. \tag{3.5}$$

We can consider only $x = \sigma_{\max}$ because, in this case, larger x will decrease if a smaller x decreases. Here α is the free variable. Finding the smallest $\alpha > 0$ amounts to solving the quartic polynomial

$$-k^9\alpha^4 + 3k^7\alpha^3 - 3k^5\alpha^2 + k - \sigma_{\max} = 0. \tag{3.6}$$

Any numerical solver can approximate α. This is the α that makes $\sigma_{\max}$ a fixed point under the overall training step. Connecting to the earlier equilibrium analysis, the special case $\alpha = 0$ is possible when already $k = \sigma_{\max}$, or $\sigma_{\max} = 1/\lambda$. While the coupling for spectral soft capping is not linear as in common implementations of AdamW, it succeeds at making tight weight norm bounds compatible with learning rate schedules that may tend to 0. Initializing the weights near $\sigma_{\max}$, rather than strictly less than it, suffices for the bound to hold throughout training in practice.

One limitation of automatic coupling is that it may be stronger than necessary, because it assumes updates align perfectly with the weights in the worst case. If the learning rate is scheduled to 0, gradients may align less with the existing weights especially at the end, which can cause the weight norm to contract slightly.

3.5.2 Proving an upper bound on the Lipschitz constant of a transformer

We elaborate on the algorithm sketched in Section 3.3.2 and prove a Lipschitz bound on attention. Our Lipschitz bounds are with respect to the max RMS norm over token positions, denoted $\|\cdot\|_{\infty\text{RMS}}$.

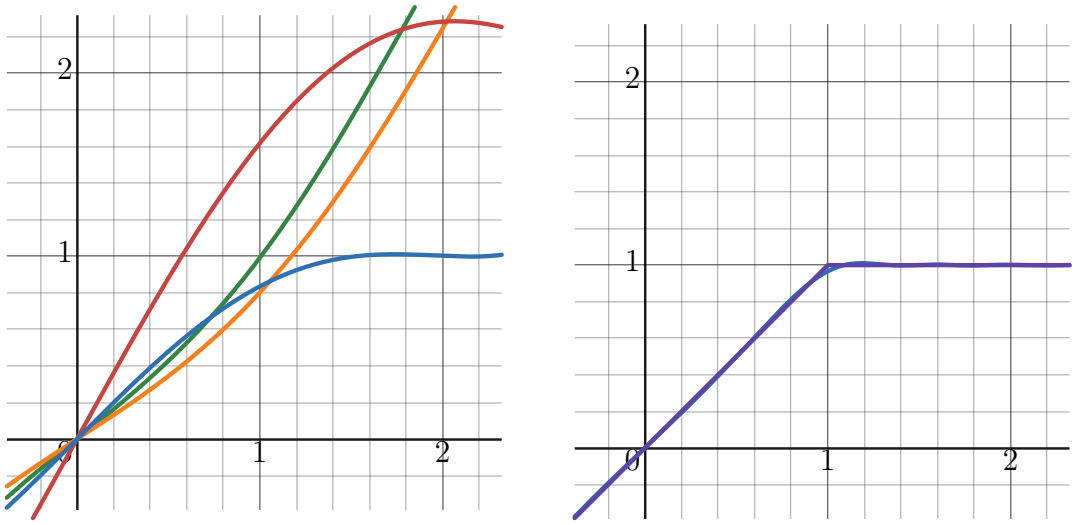

Figure 3.4: **Spectral hard cap** is a weight constraint method that applies the above odd polynomial to a weight matrix, which applies it to all singular values in parallel. Left: the four quintic polynomials. Right: the composition of the four quintic polynomials is designed to approximate $\min(1, x)$ on the range $x \in [0, 3]$.

Recall the two primary ways Lipschitz constants L_f and L_g of two functions f and g interact:

- **Adding:** $f + g$ has Lipschitz constant at most $L_f + L_g$.
- **Composing:** $f \circ g$ has Lipschitz constant at most $L_f \cdot L_g$.

Step 1: Residual connections. Suppose that, before reaching a certain residual connection, a transformer maps input data x to $f(x)$ with Lipschitz constant L. Suppose the transformer has $2N$ residual connections. Let $\alpha = \frac{1}{2N}$. The residual connection acts on $f(x)$ as

$$[(1-\alpha) \cdot \mathsf{identity} + \alpha \cdot \mathsf{block}]\,(f(x)). \tag{3.7}$$

After the residual connection, the Lipschitz constant composes and adds to become at most

$$(1-\alpha) \cdot L + \alpha \cdot L \cdot L_{\mathsf{block}}. \tag{3.8}$$

Applying this formula sequentially upper bounds the Lipschitz constant of a transformer layer by layer. We now determine L_{block} for an MLP and attention block in terms of their weight norms.

Step 2: MLP. Our MLP composes $W_{\mathsf{out}} \circ (\mathsf{GeLU}/1.1289) \circ W_{\mathsf{in}}$. The Lipschitz constants of the two weight matrices are their norms $\|W_{\mathsf{out}}\|_{\mathrm{RMS}\to\mathrm{RMS}}$ and $\|W_{\mathsf{in}}\|_{\mathrm{RMS}\to\mathrm{RMS}}$, while $\mathsf{GeLU}/1.1289$ has Lipschitz constant 1 because we divide by the maximum derivative of GeLU. Overall, the Lipschitz bound for an MLP block is $L_{\mathsf{MLP}} \leqslant \|W_{\mathsf{out}}\|_{\mathrm{RMS}\to\mathrm{RMS}}\|W_{\mathsf{in}}\|_{\mathrm{RMS}\to\mathrm{RMS}}/1.1289$.

Step 3: Attention. Let ℓ denote the token dimension. Let the queries, keys, and values be denoted by $(q, k, v) \in \mathbb{R}^{\ell\times d_Q} \times \mathbb{R}^{\ell\times d_Q} \times \mathbb{R}^{\ell\times d_V}$. Our attention block composes

$$\tfrac{1}{3} W_{\mathsf{out}} \circ F, \tag{3.9}$$

where function attention is denoted by $F = \text{softmax}(\frac{1}{d_Q} qk^\top + M)v$ for some mask M. As a consequence of the following theorem, if every attention input is unit norm, then functional attention is 1-Lipschitz. This property is what motivates scaling attention by $\frac{1}{d_Q}$ rather than $\frac{1}{\sqrt{d_Q}}$ inside the softmax. It also motivates scaling the attention output by $\frac{1}{3}$ to make it unit sensitivity. Functional attention is no longer 1-Lipschitz if its inputs are not unit norm. Recall that the shorthand notation $\|x\|_{\infty\text{RMS}}$ is the max RMS norm of a d-dimensional activation over l tokens, $x \in \mathbb{R}^{\ell \times d}$.

Theorem 3.5.1. Lipschitz bound on functional attention

Let $\diamond$ denote tensor contraction. Given any perturbations $\Delta q, \Delta k, \Delta v$ to the queries, keys, and values, functional attention satisfies

$$\|\nabla F(q,k,v) \diamond (\Delta q, \Delta k, \Delta v)\| \leqslant \|\Delta v\| + \|v\|(\|\Delta q\|\|k\| + \|q\|\|\Delta k\|), \tag{3.10}$$

where the norm is $\|\cdot\|_{\infty\text{RMS}} : \mathbb{R}^{\ell \times d} \to \mathbb{R}$, the max-over-tokens RMS norm of the embedding vector.

Proof. The argument mirrors the proof of Proposition 7 from the modular norm paper [Large et al., 2024]. We write the attention matrix as $A = \text{softmax}(\frac{1}{d_Q} qk^\top + M)$. Its derivative is $\Delta A = \nabla_{(q,k)} \text{softmax}(\frac{1}{d_Q} qk^\top + M) \diamond (\Delta q, \Delta k)$. The derivative of F splits into two terms,

$$\nabla F(q,k,v) \diamond (\Delta q, \Delta k, \Delta v) = A(\Delta v) + (\Delta A)v. \tag{3.11}$$

We call the maximum entry of A or ΔA its L^∞ operator norm, which comes into play by observing that $\|Ax\|_{\infty\text{RMS}} \leqslant \|A\|_{\infty-\mathsf{op}} \|x\|_{\infty\text{RMS}}$. For the first term, note that $\|A\|_{\infty-\mathsf{op}} \leqslant 1$ because softmax never exceeds 1. For the second term, Large et al. [2024] in Equation E.58 show that

$$\|\Delta A\|_{\infty-\mathsf{op}} \leqslant \|\Delta q\|_{\infty\text{RMS}} \|k\|_{\infty\text{RMS}} + \|q\|_{\infty\text{RMS}} \|\Delta k\|_{\infty\text{RMS}}. \tag{3.12}$$

The result follows by applying these bounds to $\|A\|_{\infty-\mathsf{op}} \|\Delta v\|_{\infty\text{RMS}} + \|\Delta A\|_{\infty-\mathsf{op}} \|v\|_{\infty\text{RMS}}$. □

Step 4. Activation norm bounds. To apply the theorem, we now bound the input norm to attention. To do so we will track the maximum RMS norm of activations everywhere in the network. We do not use layer norm and therefore cannot reset activation norms to 1. Let $x_0, \ldots, x_{2N}$ denote all the activations, from the initial embedding x_0 through to the N alternating attention and MLP blocks acting via residual connections. Suppose the embedding layer maps tokens to have RMS norm at most 1, or $\|x_0\|_{\infty\text{RMS}} \leqslant 1$. Attention and MLP increase the norm as follows:

- Attention computes $W_{\mathsf{out}} \circ (V, A)$ for some attention matrix A, where (V, A) is shorthand for functional attention. By definition V cannot increase the RMS norm of the embedding x_i at any token by more than its RMS $\to$ RMS operator norm, meaning $\|Vx_i\|_{\infty\text{RMS}} \leqslant \|V\|_{\text{RMS}\to\text{RMS}} \|x_i\|_{\infty\text{RMS}}$. The same bound applies to $(V, A)x_i$ by subadditivity of norms, since entries of the attention matrix A sum to 1 in the token dimension.

Therefore attention can increase the activation norm by

$$\|(W_{\mathsf{out}} \circ (V, A))x_i\|_{\infty\mathrm{RMS}} \leqslant \|W_{\mathsf{out}}\|_{\mathrm{RMS}\to\mathrm{RMS}}\|V\|_{\mathrm{RMS}\to\mathrm{RMS}}\|x_i\|_{\infty\mathrm{RMS}}. \tag{3.13}$$

In words, multiply the weight norms of W_{out} and V to get the maximum increase.

- The MLP computes $W_{\mathsf{out}} \circ (\mathsf{GeLU}/1.1289) \circ W_{\mathsf{in}}$. Therefore the MLP can increase activation norm by $\|W_{\mathsf{out}}\|_{\mathrm{RMS}\to\mathrm{RMS}}\|W_{\mathsf{in}}\|_{\mathrm{RMS}\to\mathrm{RMS}}/1.1289$, since $|\mathsf{GeLU}(x)| \leqslant |x|$ for all $x \in \mathbb{R}$.

- The residual connection acts like

$$\|(1-\alpha)\cdot x_i + \alpha \cdot \mathsf{block}(x_i)\|_{\infty\mathrm{RMS}} \leqslant (1-\alpha)\|x_i\|_{\infty\mathrm{RMS}} + \alpha\|\mathsf{block}(x_i)\|_{\infty\mathrm{RMS}}. \tag{3.14}$$

Algorithm to compute Lipschitz constant. Therefore, given the weight norms of all matrices in a transformer, we use the preceding results to compute its Lipschitz constant in two steps. First, we upper bound the activation norm everywhere in the network using Step 4. Second, we upper bound the Lipschitz constant using Steps 1-3. The Lipschitz bound after the final layer is what we refer to as the transformer's Lipschitz upper bound.

3.5.3 Implementing LipsFormer and bounding its Lipschitz constant

To turn our enforced norm training into LipsFormer [Qi et al., 2023], we make the following changes:

1. Remove spectral soft cap and embed projections.
2. Use CenterNorm: mean subtraction with learnable entrywise scale and bias.
3. Use scaled-head cosine attention with $\epsilon = 10^{-6}$, $\tau = 12$, $\nu = 1$. Notably, the official implementation of LipsFormer uses $\epsilon = 0$. According to their Theorem 1, this choice may make a finite Lipschitz bound impossible. We set $\epsilon > 0$ to fix the issue.
4. Heuristically scale down attention output by $1/n_{\text{heads}}$ to match their implementation.
5. Insert residual connections with learnable strength α, initialized to $1/n_{\text{residual\_connections}}$.
6. Xavier normal initialize linear layers, then apply spectral normalization $W \mapsto W/\|W\|_*$.
7. Include drop path: every residual connection is skipped with $p = 0.5$ and, if taken, is scaled up by $1/(1-p)$, matching their official implementation which uses `nn.Dropout`.
8. Use weight decay 0.1, matching their implementation (not applied to scalar parameters).

9. Use the Muon optimizer to give LipsFormer the fairest comparison, copying hyperparameters from our run. We tested training with AdamW for all parameters, an exact replication, but found performance degraded sigificantly: after 1770 steps, validation loss was 4.86 (compared to 3.61) and validation accuracy was 0.227 (compared to 0.301).

10. For non-weight-matrix parameters, use Adam hyperparameters $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ to match their implementation.

11. Use cosine learning rate schedule with decay to 0 to match their implementation.

Bounding the Lipschitz constant of LipsFormer. In Table 3.1, we report that our trained implementation of LipsFormer has a Lipschitz upper bound of 10^{130}. To calculate this value, we use the final weight norms of the MLP and attention blocks to bound the Lipschitz constant of each residual block, relying on LipsFormer's Theorem 1:

$$\mathsf{Lip}(\mathrm{SCSA})_2 \leqslant 2N(N-1)\nu\tau\epsilon^{-\frac{1}{2}}\|W^K\|_2 + 2(N-1)\nu\tau\epsilon^{-\frac{1}{2}}\|W^Q\|_2 + 2N\nu\epsilon^{-\frac{1}{2}}\|W^V\|_2.$$

Using $N = 128$ (head dimension), $\tau = 12$, $\nu = 1$, and empirical weight norms, we calculate the Lipschitz constant for every layer. We use the maximum entry of the learned residual strength α, which is an entrywise multiplication, to convert the layerwise bounds into a final bound

$$\mathsf{Lip}(F) \leqslant \prod_{s=1}^{S}\prod_{m=1}^{S}(1 + \alpha_{s,m}\mathsf{Lip}(f_{s,m})),$$

which we take from their Equation 19. Alpha has typical maximum entries around 0.5 for attention connections and 0.15 for MLP connections. With $\epsilon = 10^{-6}$, we compute a final Lipschitz bound of 1.97×10^{129}.

3.5.4 Experimental details

This section gives experimental details for all results in the paper. The three categories of experiments we run are MLP training, Shakespeare transformer training, and NanoGPT speedrun training.

Datasets.

- For MLP training we use the CIFAR-10 dataset Krizhevsky et al. [2009] with the standard train and test splits and no data augmentation. We do not shuffle the order of batches.

- For Shakespeare transformer training we use Karpathy's 1M character-level dataset with standard training and validation splits [Karpathy, 2022]. We shuffle the order of batches.

- For NanoGPT speedrun transformer training we use the FineWeb10B dataset [Penedo et al., 2024] loaded in the standard order. We use the same validation split as the modded NanoGPT speedrun benchmark [Jordan et al., 2024a].

Compute requirement. All our experiments can run on a V100, A100, or H100 GPU in less than 5 minutes, except the NanoGPT speedrun transformer which requires 8xH100 and runs in 5-10 minutes.

Modula library. For MLP and Shakespeare experiments, we use JAX [Bradbury et al., 2018] on top of the Modula library [Large et al., 2024, Bernstein, 2025]. We implement our own model components. Our AdamW implementation does not include bias correction, although the discrepancy decays rapidly after aronud 20 steps because we use $\beta_1 = 0.9, \beta_2 = 0.95$ in all experiments except one, not reported, in which we determine that this is a good setting for the momentum EMAs.

MLP experiments. All MLPs we train are width 256 and depth 3 (i.e., one hidden layer) with ReLU activations and no bias on data from CIFAR-10. We use batch size 512 and a linear learning rate schedule that decays to 0 in all experiments. Modula's mass calculation causes the effective learning rate to be scaled by 1/3. We train for 50 epochs except in one case, when we train for 20 epochs for the models in Figure 3.2. We zero-initialize the final layer. We train all models in float32 precision and run the weight constrain methods in float32 precision. We experimented with lower precision and found comparable metrics across the board for bfloat16 training. We set seed 0 and store all hyperparameters and log information to enhance reproducibility.

Shakespeare experiments. All transformers we train for Shakespeare are width 256 with 3 blocks (attention + MLP), no bias, and four attention heads. The out projection in each attention and MLP block is initialized to zero. We use sequence length 256 and batch size 64 to match the baseline from [Karpathy, 2022], except we train for 2000 steps while Karpathy trains for 5000 steps. We set Modula's blocks mass parameter to 32 to cause 95% of the feature learning to occur in the transformer blocks. We determined this ratio by sweeping the blocks mass, which controls the ratio of learning rate between the two embedding layers and the transformer blocks. Training with Muon means applying Muon to all linear layer weight matrices (including the final logit head) but normalizing the columns of embedding gradient, as suggested by the $\ell_1 \to$ RMS duality map [Bernstein and Newhouse, 2024b]. We were concerned that rare tokens may cause the momentum buffer to dualize columns to full strength updates for hundreds of steps until the column decays to exactly zero, so we tested whether capping the maximum inflation factor for the embedding column normalization could help. We tested maximum factors in the set $\{1, 4, 16, \ldots, 65536\}$ across 8 seeds and found no significant difference. We choose to maximally multiply each column by 16 during the dualization step. Finally, we found that to train to the validation losses reported we had to use a trick: we decayed the learning rate by a factor of $1/2$ per residual layer, causing later layers to train more than earlier layers. This change is implemented by setting the sensitivity of the Mul module in Modula to 1. We do not know why this trick is necessary.

Figure 3.1 sweeps over the following hyperparameters:

- MLPs on CIFAR-10: we test the following combinations of optimzer and weight constraint method: weight decay, spectral weight decay, spectral hammer, spectral normalization, and Stiefel manifold projection for both AdamW and Muon; and spectral soft cap and spectral hard cap only for Muon. For AdamW, we vary the weight decay and spectral weight decay parameters with 10 points in log-space from 10^{-2}

to 10^0. For Muon we vary the weight decay parameter with 10 points in log-space from 10^{-3} to 10^0 and the spectral weight decay parameter with 10 points in log-space from 10^{-2} to 10^0. For AdamW with spectral normalization, Stiefel manifold projection, and spectral hammer, we vary the maximum weight norm in the set $\sigma_{\max} \in \{2, 3, 4, 5, 6, 7, 8\}$. For Muon with spectral normalization, Stiefel manifold projection, spectral soft cap, and spectral hard cap, we vary the maximum weight norm in the set $\sigma_{\max} \in \{4, 5, 6, 7, 8, 9, 10\}$. For Muon with spectral hammer we vary the maximum weight norm in the set $\sigma_{\max} \in \{1, 2, 3, \ldots, 10\}$. For AdamW with all methods we sweep 16 learning rates in log-space between 10^{-5} and $10^{-0.5}$. For Muon we sweep 16 learning rates in log-space between 10^{-2} and 10^1 for all methods except for spectral hammer, where we use these learning rates for $\sigma_{\max} \in \{4, 5, 6, 7, 8, 9, 10\}$, and use 16 learning rates in log-space between 10^{-3} and 10^0 for $\sigma_{\max} \in \{1, 2, 3\}$. Overall, this sweep results in 1,610 total combinations, 682 with AdamW and 928 with Muon.

- Transformers on Shakespeare: we test the following combinations of optimizer and weight constraint method: (AdamW, weight decay), (AdamW, spectral normalize), (AdamW, spectral hammer), (Muon, weight decay), (Muon, spectral normalize), (Muon, spectral soft cap). For spectral normalize, spectral hammer, and spectral soft cap, we vary the maximum weight norm in the set $\sigma_{\max} \in \{1.0, 1.2, \ldots, 2.8, 3.0\}$. For the baseline, we vary weight decay in the set $\lambda \in \{2/3, 0.5, 0.4, 0.3, 0.2, 0.1, 0.05, 0.03, 0.01, 0\}$. For AdamW we sweep 16 learning rates between $10^{-4.5}$ and $10^{-1.5}$. For Muon, we sweep 12 learning rates between $10^{-1.5}$ and $10^{1.5}$. We ran tests before to find ranges that cover the optimal learning rate.

Figure 3.2 reports adversarial examples and dataset-wide statistics from two models trained for 20 epochs. The AdamW model is trained with learning rate 8.1×10^{-3} and weight decay $\lambda = 0.1$. The Muon model is trained with learning rate 2.3×10^{-1} and weight decay $\lambda = 0$, using the spectral soft cap method with a weight constraint of $\sigma_{\max} = 3$.

The left panel of Figure 3.3 visualizes the same data from the experiment for Figure 3.1, but focuses only on MLPs trained with Muon on CIFAR-10. The middle and right panels use the Muon optimizer, with the following tuples of weight constraint method, maximum singular value, weight decay, spectral weight decay, and learning rate:

- Weight decay, N/A, 0.1, 0, 1.585
- Spectral weight decay, N/A, 0, 0.05, 0.157
- Spectral normalization, 6, 0, 0, 1.0
- Stiefel manifold projection, 5, 0, 0, 1.0
- Spectral soft cap, 6, 0, 0, 0.398
- Spectral hard cap, 5, 0, 0, 0.631

NanoGPT experiments. Following the Modded-NanoGPT speedrun standard [Jordan et al., 2024a], our training runs print log files with the full source code required to reproduce the results. We briefly summarize the changes we made to convert the NanoGPT speedrun record (as of May 2025) into our method:

- Every step, RMS normalize the embedding columns.
- Initialize all linear layer weight matrices to be orthogonal.
- Reparameterize residual connections according to Equation (3.1): $\frac{L-1}{L}x + \frac{1}{L}\mathsf{block}(x)$ residual connections, where $L = 24$ is the number of residual connections.
- Reparameterize attention according to Equation (3.2): $\frac{1}{3}$ overall scale on the attention output and $1/d_{\text{head}}$ scale inside the softmax.
- Every step, apply spectral soft cap (or spectral normalize) to every linear layer weight matrix based on a prespecified maximum desired weight norm $\sigma_{\max}$.
- Use different orthogonalization coefficients that at most inflate a singular value to 1.14502. Therefore, the maximum update norm we pass to the strength parameter solver for learning rate coupling in spectral soft cap is $\eta \cdot 1.14502 \cdot 1.05$ with an extra factor of 1.05 to be safe around numerical precision errors. The iteration is derived by modifying the method in [Cesista et al., 2025].
- Remove U-net structure.
- Use $\mathsf{GeLU}/1.1289$ instead of ReLU^2.
- Switch dimensional scaling to be $\sqrt{\mathsf{fan\_out}/\mathsf{fan\_in}}$ instead of $\max(1, \sqrt{\mathsf{fan\_out}/\mathsf{fan\_in}})$.
- Remove RMS normalization: the model is now Lipschitz continuous.
- Add back the 7th attention layer (which was removed in the speedrun).
- Run weight projections in `bfloat16` (which we found to slightly improve performance). Spectral normalization uses 2 iterations, meaning that weight norms can exceed the specified maximum $\sigma_{\max}$ due to approximation error; in practice weights with norms enforced by spectral normalization exceed the specified maximum by around 10%.

Chapter 4

Conclusion

Metrized deep learning suggests assigning a norm to every part of a neural network. We hope this thesis serves as a useful guide to the method. While we present early applications, such as deriving Muon, there remain many paths to explore. In summary:

Enforcing norms on the *gradients*, duality maps arise as a balance between decreasing loss the most and disturbing the model the least. Popular optimizers such as SGD, Adam, and Shampoo can be viewed as resolving this balance under different choices of norm. Duality-based optimizers like Muon have shown convincing evidence of speeding up training at scale.

Enforcing norms on the *weights*, a Lipschitz bound can be maintained throughout training. Standard methods—weight decay and spectral normalization—appear to improve the Lipschitz vs. performance tradeoff when trained with Muon. A constraint method co-designed for Muon, called spectral cap, performs as well or slightly better than spectral normalization. A 4-Lipschitz, 2M-parameter transformer on Shakespeare text reaches validation accuracy 60%. Scaling to 145M parameters on NanoGPT, a 600-Lipschitz transformer reaches 21% accuracy on internet text. However, attaining baseline accuracy resorts to an astronomical global Lipschitz bound of 10^{278}. Maximum activation entries remain small in practice, with potential benefits for low precision training and inference.

References

Atish Agarwala, Samuel Stern Schoenholz, Jeffrey Pennington, and Yann Dauphin. Temperature check: Theory and practice for training models with softmax-cross-entropy losses. *Transactions on Machine Learning Research*, 2023.

Arash Ahmadian, Saurabh Dash, Hongyu Chen, Bharat Venkitesh, Stephen Gou, Phil Blunsom, Ahmet Üstün, and Sara Hooker. Intriguing properties of quantization at scale. *arXiv:2305.19268*, 2023.

Kwangjun Ahn and Byron Xu. Dion: A communication-efficient optimizer for large models. *arXiv:2504.05295*, 2025.

Shun-ichi Amari. *Information Geometry and Its Applications*. Springer, 2016.

Cem Anil, James Lucas, and Roger B. Grosse. Sorting out Lipschitz function approximation. In *International Conference on Machine Learning*, 2019.

Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. Scalable second order optimization for deep learning. *arXiv:2002.09018*, 2020.

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. *arXiv:1511.06464*, 2016.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv:1607.06450*, 2016.

Lukas Balles and Philipp Hennig. Dissecting Adam: The sign, magnitude and variance of stochastic gradients. In *International Conference on Machine Learning*, 2018.

Peter Bartlett, Dylan J. Foster, and Matus Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Neural Information Processing Systems*, 2017.

Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *arXiv:1206.5533*, 2012.

Jeremy Bernstein. Optimisation & generalisation in networks of neurons. *arXiv:2210.10101*, 2022.

Jeremy Bernstein. The modula docs, 2025. URL https://docs.modula.systems/.

Jeremy Bernstein and Laker Newhouse. Old optimizer, new norm: An anthology. In *Workshop on Optimization for Machine Learning*, 2024a.

Jeremy Bernstein and Laker Newhouse. Modular duality in deep learning. *arXiv:2410.21265*, 2024b.

Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signSGD: Compressed optimisation for non-convex problems. In *International Conference on Machine Learning*, 2018.

Jeremy Bernstein, Arash Vahdat, Yisong Yue, and Ming-Yu Liu. On the distance between two neural networks and the stability of learning. *arXiv:2002.03432*, 2021.

Jeremy Bernstein, Chris Mingard, Kevin Huang, Navid Azizan, and Yisong Yue. Automatic gradient descent: Deep learning without hyperparameters. *arXiv:2304.05187*, 2023.

Louis Béthune, Thibaut Boissin, Mathieu Serrurier, Franck Mamalet, Corentin Friedrich, and Alberto Gonzalez Sanz. Pay attention to your loss: Understanding misconceptions about Lipschitz neural networks. In *Neural Information Processing Systems*, 2022.

Louis Béthune, Thomas Massena, Thibaut Boissin, Yannick Prudent, Corentin Friedrich, Franck Mamalet, Aurelien Bellet, Mathieu Serrurier, and David Vigouroux. DP-SGD without clipping: The Lipschitz neural network way. In *International Conference on Learning Representations*, 2024.

Åke Björck and C. Bowie. An iterative algorithm for computing the best estimate of an orthogonal matrix. *SIAM Journal on Numerical Analysis*, 1971.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004a.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004b.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

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. Language models are few-shot learners. *arXiv:2005.14165*, 2020.

David Carlson, Volkan Cevher, and Lawrence Carin. Stochastic spectral descent for restricted Boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, 2015a.

David Carlson, Ya-Ping Hsieh, Edo Collins, Lawrence Carin, and Volkan Cevher. Stochastic spectral descent for discrete graphical models. *Selected Topics in Signal Processing*, 2016.

David E. Carlson, Edo Collins, Ya-Ping Hsieh, Lawrence Carin, and Volkan Cevher. Preconditioned spectral descent for deep learning. In *Neural Information Processing Systems*, 2015b.

Franz Louis Cesista, YouJiacheng, and Keller Jordan. Squeezing 1–2% efficiency gains out of Muon by optimizing the Newton-Schulz coefficients, 2025. URL http://leloykun.github.io/ponder/muon-opt-coeffs/.

Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. *arXiv:1704.08847*, 2017.

George E. Dahl, Frank Schneider, Zachary Nado, Naman Agarwal, Chandramouli Shama Sastry, Philipp Hennig, Sourabh Medapati, Runa Eschenhagen, Priya Kasimbeg, Daniel Suo, Juhan Bae, Justin Gilmer, Abel L. Peirson, Bilal Khan, Rohan Anil, Mike Rabbat, Shankar Krishnan, Daniel Snider, Ehsan Amid, Kongtao Chen, Chris J. Maddison, Rakshith Vasudev, Michal Badura, Ankush Garg, and Peter Mattson. Benchmarking neural network training algorithms. *arXiv:2306.07179*, 2023.

DeepSeek-AI. Deepseek-v3 technical report. *arXiv:2412.19437*, 2025.

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, Rodolphe Jenatton, Lucas Beyer, Michael Tschannen, Anurag Arnab, Xiao Wang, Carlos Riquelme, Matthias Minderer, Joan Puigcerver, Utku Evci, Manoj Kumar, Sjoerd van Steenkiste, Gamaleldin F. Elsayed, Aravindh Mahendran, Fisher Yu, Avital Oliver, Fantine Huot, Jasmijn Bastings, Mark Patrick Collier, Alexey Gritsenko, Vighnesh Birodkar, Cristina Vasconcelos, Yi Tay, Thomas Mensink, Alexander Kolesnikov, Filip Pavetić, Dustin Tran, Thomas Kipf, Mario Lučić, Xiaohua Zhai, Daniel Keysers, Jeremiah Harmsen, and Neil Houlsby. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning*, 2023.

John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal Machine Learning Research*, 2011.

Kai Fan. Unifying the stochastic spectral descent for restricted Boltzmann machines with Bernoulli or Gaussian inputs. *arXiv:1703.09766*, 2017.

Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George J. Pappas. Efficient and accurate estimation of Lipschitz constants for deep neural networks. In *Neural Information Processing Systems*, 2019.

Thomas Flynn. The duality structure gradient descent algorithm: Analysis and applications to neural networks. *arXiv:1708.00523*, 2017.

Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J. Cree. Regularisation of neural networks by enforcing lipschitz continuity. *arXiv:1804.04368*, 2020.

Vineet Gupta, Tomer Koren, and Yoram Singer. A unified approach to adaptive regularization in online and stochastic optimization. Technical report, Google Brain, 2017.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv:1606.08415*, 2023.

Alex Henry, Prudhvi Raj Dachapally, Shubham Pawar, and Yuxuan Chen. Query-key normalization for transformers. In *Empirical Methods in Natural Language Processing*, 2020.

Preston Hess and Andrew Hutchison. Exploring frobenius and spectral normalization in mlps and residual networks. https://deep-learning-mit.github.io/staging/blog/2023/WeightDecaySpecNormEffects/, December 2023. MIT BCS and EECS, MIT EECS.

Nicholas J. Higham. *Functions of Matrices*. Society for Industrial and Applied Mathematics, 2008.

Yujia Huang, Huan Zhang, Yuanyuan Shi, J. Zico Kolter, and Anima Anandkumar. Training certifiably robust neural networks with efficient local Lipschitz bounds. In *Neural Information Processing Systems*, 2021.

Minyoung Huh, Hossein Mobahi, Richard Zhang, Brian Cheung, Pulkit Agrawal, and Phillip Isola. The low-rank simplicity bias in deep networks. *arXiv:2103.10427*, 2023.

Ricardo J. Jesus, Mário L. Antunes, Rui A. da Costa, Sergey N. Dorogovtsev, José F. F. Mendes, and Rui L. Aguiar. Effect of initial configuration of weights on training and function of artificial neural networks. *Mathematics*, 2021.

Keller Jordan, Jeremy Bernstein, Brendan Rappazzo, @fernbear.bsky.social, Boza Vlado, You Jiacheng, Franz Cesista, Braden Koszarsky, and @Grad62304977. modded-nanogpt: Speedrunning the nanogpt baseline, 2024a. URL https://github.com/KellerJordan/modded-nanogpt.

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

Andrej Karpathy. nanoGPT. https://github.com/karpathy/nanoGPT, 2022. MIT License.

Hyunjik Kim, George Papamakarios, and Andriy Mnih. The lipschitz constant of self-attention. *arXiv:2006.04710*, 2021.

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

Zdislav Kovarik. Some iterative methods for improving orthonormality. *SIAM Journal on Numerical Analysis*, 1970.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. CIFAR-100 and CIFAR-10 (Canadian Institute for Advanced Research). http://www.cs.toronto.edu/~kriz/cifar.html, 2009. MIT License.

Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems*, volume 4, 1991.

Tanishq Kumar, Zachary Ankner, Benjamin F. Spector, Blake Bordelon, Niklas Muennighoff, Mansheej Paul, Cengiz Pehlevan, Christopher Ré, and Aditi Raghunathan. Scaling laws for precision. *arXiv:2411.04330*, 2024.

Kenneth Lange. *MM Optimization Algorithms*. Society for Industrial and Applied Mathematics, 2016. doi:10.1137/1.9781611974409.

Tim Large, Yang Liu, Minyoung Huh, Hyojin Bahng, Phillip Isola, and Jeremy Bernstein. Scalable optimization in the modular norm. In *Neural Information Processing Systems*, 2024.

Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Neural Information Processing Systems*, 2019.

Tianyang Lin. Flash-muon: An efficient implementation of muon optimizer, 2025. URL https://github.com/nil0x9/flash-muon.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, Yanru Chen, Huabin Zheng, Yibo Liu, Shaowei Liu, Bohong Yin, Weiran He, Han Zhu, Yuzhi Wang, Jianzhou Wang, Mengnan Dong, Zheng Zhang, Yongsheng Kang, Hao Zhang, Xinran Xu, Yutao Zhang, Yuxin Wu, Xinyu Zhou, and Zhilin Yang. Muon is scalable for llm training. *arXiv:2502.16982*, 2025.

Yang Liu, Jeremy Bernstein, Markus Meister, and Yisong Yue. Learning by turning: Neural architecture aware optimisation. *arXiv:2102.07227*, 2021.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv:1711.05101*, 2019.

Per-Gunnar Martinsson and Joel A. Tropp. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numerica*, 2020.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv:1802.05957*, 2018.

Depen Morwani, Itai Shapira, Nikhil Vyas, Eran Malach, Sham Kakade, and Lucas Janson. A new perspective on Shampoo's preconditioner. *arXiv:2406.17748*, 2024.

Arkady S. Nemirovsky and David B. Yudin. *Problem complexity and method efficiency in optimization*. Wiley, 1983.

Yurii Nesterov and Boris T. Polyak. Cubic regularization of Newton method and its global performance. *Mathematical Programming*, 108(1):177–205, 2006. doi:10.1007/s10107-006-0706-8.

Laker Newhouse, Dakota Goldberg, and Ricardo Ruiz. Faster symmetric matrix multiplication with thunderkittens. https://www.lakernewhouse.com/assets/writing/faster-symmul-with-thunderkittens.pdf, December 2024.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, New York, NY, 2 edition, 2006. ISBN 978-0-387-30303-1. doi:10.1007/978-0-387-40065-5.

Kiho Park, Yo Joong Choe, and Victor Veitch. The linear representation hypothesis and the geometry of large language models. *arXiv:2311.03658*, 2024.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Neural Information Processing Systems: Datasets and Benchmarks Track*, 2024. ODC-By 1.0 License.

Thomas Pethick, Wanyun Xie, Kimon Antonakopoulos, Zhenyu Zhu, Antonio Silveti-Falls, and Volkan Cevher. Training deep learning models with norm-constrained lmos. *arXiv:2502.07529*, 2025.

Xianbiao Qi, Jianan Wang, Yihao Chen, Yukai Shi, and Lei Zhang. Lipsformer: Introducing lipschitz continuity to vision transformers. *arXiv:2304.09856*, 2023.

F. Riesz. Quelques applications de la théorie des fonctions de plusieurs variables complexes. *Mathematische Zeitschrift*, 1907.

Ishaan Shah, Anthony M. Polloreno, Karl Stratos, Philip Monk, Adarsh Chaluvaraju, Andrew Hojel, Andrew Ma, Anil Thomas, Ashish Tanwer, Darsh J Shah, Khoi Nguyen, Kurt Smith, Michael Callahan, Michael Pust, Mohit Parmar, Peter Rushton, Platon Mazarakis, Ritvik Kapila, Saurabh Srivastava, Somanshu Singla, Tim Romanski, Yash Vanjani, and Ashish Vaswani. Practical efficiency of muon for pretraining. *arXiv:2505.02222*, 2025.

Benjamin F. Spector, Simran Arora, Aaryan Singhal, Daniel Y. Fu, and Christopher Ré. Thunderkittens: Simple, fast, and adorable ai kernels. *arXiv:2410.20399*, 2024.

Matthew Streeter. Universal majorization-minimization algorithms. *arXiv:2308.00190*, 2023.

Jianlin Su. Thoughts from spectral norm gradient to new weight decay, Dec 2024. URL https://kexue.fm/archives/10648.

Jianlin Su. Muon sequel: Why did we choose to try muon?, Feb 2025. URL https://kexue.fm/archives/10739.

Shiqing Sun and James C. Spall. Connection of diagonal Hessian estimates to natural gradients in stochastic optimization. In *Information Sciences and Systems*, 2021.

Joel A. Tropp. *Topics in Sparse Approximation*. PhD thesis, The University of Texas at Austin, 2004.

Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. In *Neural Information Processing Systems*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems*, 2017.

Mitchell Wortsman, Peter J. Liu, Lechao Xiao, Katie Everett, Alex Alemi, Ben Adlam, John D. Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, Jeffrey Pennington, Jascha Sohl-dickstein, Kelvin Xu, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Small-scale proxies for large-scale transformer training instabilities. *arXiv:2309.14322*, 2023.

Greg Yang and Edward J. Hu. Tensor programs IV: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, 2021.

Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv:2203.03466*, 2022.

Greg Yang, James B. Simon, and Jeremy Bernstein. A spectral condition for feature learning. *arXiv:2310.17813*, 2023.

Yuichi Yoshida and Takeru Miyato. Spectral norm regularization for improving the generalizability of deep learning. *arXiv:1705.10941*, 2017.

Jiacheng You. Rapidly converging orthogonalizing Newton-Schulz iteration, 2025. URL https://x.com/YouJiacheng/status/1893704552689303901.

Pengxiang Zhao, Ping Li, Yingjie Gu, Yi Zheng, Stephan Ludger Kölker, Zhefeng Wang, and Xiaoming Yuan. Adapprox: Adaptive approximation in Adam optimization via randomized low-rank matrices. *arXiv:2403.14958*, 2024.