Modular Duality in Deep Learning

Jeremy Bernstein [1] **Laker Newhouse** [1]

Abstract

An old idea in optimization theory says that since the gradient is a *dual vector* it may not be subtracted from the weights without first being mapped to the *primal space* where the weights reside. We take this idea seriously in this paper and construct such a duality map for general neural networks. Our map, which we call *modular dualization*, forms a unifying theoretical basis for training algorithms that are a) *fast* and b) *scalable*. Modular dualization involves first assigning operator norms to layers based on the semantics of each layer, and then using these layerwise norms to recursively induce a duality map on the weight space of the full neural architecture. We derive GPU-friendly algorithms for dualizing Embed, Linear and Conv2D layers—the latter two methods are based on a Newton-Schulz iteration. We conclude with small experiments demonstrating the speed, scalability and novel numerical properties of duality-based optimizers. Our methods were used in the Muon optimizer, which recently set speed records for training NanoGPT and was scaled up to a 1.5 billion parameter transformer.

1 Introduction

This paper pursues a rigorous and first-principles theoretical framework for designing neural network training algorithms. We hope that such a framework will facilitate the design of a next generation of fast and scalable optimizers that are automatically tailored to different neural architectures.

While gradient descent is the workhorse of modern machine learning, the most vanilla form of the algorithm does not, in our view, pass a basic *type check*. For a gradient update to type check, we insist that the gradient must be passed through a duality map before being multiplied by a learning rate and applied to the weights:

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

Why? The reason is that the loss function may not be equally smooth in all directions in weight space, and there is no reason for the sizes of different components of the raw gradient vector[1] to respect this heterogeneity. In other words, *the geometry of the loss function may be non-isotropic*. Insisting on a type check should force the user to become cognizant of this issue and to find a suitable duality map. A good duality map should adjust the size and direction of the gradient to respect the smoothness structure of the loss function.

Duality maps on vector spaces are commonplace in physics and applied math. Examples include the *musical isomorphism* in differential geometry (Grosse, 2022), *raising and lowering indices* in general relativity (Carroll, 2019) and the *bra-ket notation* in quantum mechanics (Sakurai & Napolitano, 2020). Duality maps are also central to several optimization theories including *mirror descent* (Nemirovsky & Yudin, 1983), *natural gradient descent* (Amari, 2016) and *steepest descent on a normed space* (Boyd & Vandenberghe, 2004). Despite the efforts of some prescient papers (Carlson et al., 2015b; Flynn, 2017), the latter kind of norm-based duality map is yet to puncture the deep learning mainstream.

We believe that duality is a key theoretical concept that will help in building performant large-scale machine learning systems. To support this belief, we show in this paper that two important and seemingly disparate methods in contemporary optimization research may be seen as approximations to a single duality map. These methods are *maximal update parameterization* (Yang & Hu, 2021, μP), which is aimed at scalable training, and *Shampoo* (Shi et al., 2023), which is targeted at fast training. We show in Section 4.1 that both methods emerge as partial approximations to a single duality map induced by the RMS–RMS operator norm.

[1]MIT CSAIL, United States. Correspondence to: Jeremy Bernstein <jbernstein@mit.edu>, Laker Newhouse <lakern@mit.edu>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

[1]In this paper, we use the term "gradient" to mean the partial derivative of the loss function. This is in accordance with the terminology used in machine learning software libraries such as PyTorch (Paszke et al., 2019) and JAX (Bradbury et al., 2018). We chose this verbiage to help make the paper easily accessible to a broad machine learning audience. However, it is worth noting that this terminology is at odds with common mathematical parlance (Blondel & Roulet, 2024) where a "gradient" is a primal vector obtained by applying some form of duality map to the partial derivative of the loss function.

1.1 Summary of contributions

First contribution. We describe a procedure for constructing duality maps for general neural architectures. The procedure, called *modular dualization*, works in three steps:

- **Step 1:** Operator norms are assigned to individual layers based on the input-output semantics of each layer;
- **Step 2:** Based on these operator norms, duality maps are constructed for individual layers;
- **Step 3:** Given the layerwise duality maps and the structure of the neural architecture, a single duality map is recursively induced on the full weight space of the architecture.

Second contribution. We instantiate modular dualization for a rich family of neural architectures—including convolutional networks and transformers—by writing down duality maps for Linear, Embed and Conv2D layers. We also provide GPU-friendly algorithms for computing these duality maps efficiently. One of these methods has already been applied in the Muon optimizer (Jordan et al., 2024b).

Third contribution. We run two experiments. In the first, we find that duality-based optimizers are fast and scalable across width—see Figure 1. In the second, we see that duality-based training exhibits novel numerical properties: the weights move substantially further from their initial values than for non-dualized training—see Figure 2.

2 Related Work

This paper constructs a duality map for general neural architectures. Our approach is based on assigning operator norms to individual network layers and using these layerwise norms to recursively induce a duality map on the full neural architecture. The most closely related prior work is a series of papers on *spectral descent* (Carlson et al., 2015a;b; 2016) and a paper on *duality structure gradient descent* (Flynn, 2017).

Spectral descent has been applied to restricted Boltzmann machines (Carlson et al., 2015a) and discrete graphical models (Carlson et al., 2016), but let us focus on the more closely related paper on spectral descent for deep learning (Carlson et al., 2015b). In that paper, the authors propose assigning the Schatten-∞ norm (a.k.a. spectral norm) to individual linear layers. This assignment is based on the observation that neural networks admit natural majorization bounds in the Schatten-∞ norm. The authors call the corresponding duality map for linear layers the "#-operator"—a name presumably inspired by the musical isomorphism (Grosse, 2022). The authors propose a cheap approximation to the #-operator based on sketching (Martinsson & Tropp, 2020), and they also propose a way to mix RMSprop-style preconditioning information (Tieleman & Hinton, 2012) into the weight updates. In contrast to our work, the authors only derive duality maps for single linear layers, and these maps are then heuristically extended to all-layer updates. Nonetheless, the authors achieve substantial wall clock speedups using variants of spectral descent to train small networks.

Now, let us turn our attention to duality structure gradient descent (Flynn, 2017), which constructs a duality map on the full weight space of the neural architecture based on identifying a *Finsler structure* (Deimling, 1985) inherent to neural networks. Similar to modular dualization, Flynn (2017)'s duality map works by assigning duality maps to each layer and then inducing a duality map on the full weight space. The substantial difference to our approach is that Flynn (2017) leverages a weighted sum (L_1 combination) of layerwise norms to construct his full duality map. This leads to optimization methods that only update a single layer at each iteration, and the methods need to be heuristically extended to achieve all-layer updates. In contrast, we leverage the modular norm (Large et al., 2024), which takes a weighted max (L_∞ combination) of layerwise norms. In turn, our duality map leads directly to more conventional all-layer optimizers.

Another important difference between our work on modular duality and prior work on duality structure gradient descent is that we fully "modularize" our theory—meaning that our construction is explicitly recursive—and as such it is easy to code up into a software package. In this regard, we are inspired by a line of work that attempts to build optimization algorithms that automatically adapt to the structure of general computation graphs. The earliest work we know of in this category is the PhD thesis of Grant (2004) on disciplined convex programming, which aims to infer the convexity properties of general functions by breaking them up into subexpressions and applying composition theorems from convex analysis. More recent progress in this vein includes work on universal majorization-minimization algorithms (Streeter & Dillon, 2022; Streeter, 2023) and related papers on automatic majorization (Tran et al., 2015; Bernstein et al., 2023).

3 Theoretical Preliminaries

In this section, we introduce duality maps, a means of constructing duality maps based on norms, and finally a norm called the *modular norm* that is well-suited to describe the geometry of general neural architectures.

3.1 Duality Maps

Given a vector space $\mathcal{V}$, we say that a function $f : \mathcal{V} \to \mathbb{R}$ is a *linear functional* on $\mathcal{V}$ if f is linear. We define the

dual space $\mathcal{V}^*$ to be the set of linear functionals on $\mathcal{V}$. The dual space is itself a vector space provided that addition is defined pointwise $(f+g)(x) := f(x) + g(x)$ and scalar multiplication is defined pointwise $(\alpha f)(x) := \alpha f(x)$ for any scalar α. By *duality map* we mean any function that sends a member of the dual vector space $\mathcal{V}^*$ to the primal vector space $\mathcal{V}$. The function need not be an involution.

Let $\mathcal{L} : \mathcal{W} \to \mathbb{R}$ denote the loss of a differentiable machine learning model with weight space $\mathcal{W} = \mathbb{R}^n$. The Taylor expansion of the loss at weight setting $\boldsymbol{w} \in \mathcal{W}$ is given by:

$$\mathcal{L}(\boldsymbol{w} + \Delta\boldsymbol{w}) = \mathcal{L}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})^\top \Delta\boldsymbol{w} + \cdots . \quad (1)$$

Observe that, in the first-order term, the gradient $\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})$ is acting as a linear functional: it is pairing with the weight vector $\Delta\boldsymbol{w} \in \mathcal{W}$ in a linear way to produce a real number. As such, we shall say that the gradient belongs to the dual weight space: $\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}) \in \mathcal{W}^*$. We shall forbid ourselves from directly subtracting a member of the dual weight space $\mathcal{W}^*$ from the weight space $\mathcal{W}$. If we would like to conduct a gradient descent update, then we had better find a duality map to send the gradient back to the primal space $\mathcal{W}$.

This restriction may seem absurd! After all, here the weight space $\mathcal{W}$ and its dual $\mathcal{W}^*$ are both just $\mathbb{R}^n$. However, insisting upon this type check serves to remind us that the curvature of the loss function may be highly heterogeneous. The next section will show one way to construct duality maps to account for this.

3.2 Steepest Descent on a Normed Space

Suppose that we have found a *norm* $\|\cdot\| : \mathcal{W} \to \mathbb{R}$ and a *sharpness parameter* $\lambda > 0$ that serve as a good model of the higher-order terms in the Taylor expansion of the loss function given in Equation (1):

$$\mathcal{L}(\boldsymbol{w} + \Delta\boldsymbol{w}) \lessapprox \mathcal{L}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})^\top \Delta\boldsymbol{w} + \frac{\lambda}{2} \cdot \|\Delta\boldsymbol{w}\|^2. \quad (2)$$

In other words, the norm provides a good characterization of the heterogeneity in curvature of the loss function. Then it makes sense to solve for a weight update $\Delta\boldsymbol{w}$ by minimizing the right-hand side of Equation (2). We will show that the minimizer can be expressed in terms of a *dual norm* and a *duality map*:

Definition 1 (Dual norm)**.** *Given a norm $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$, the dual norm $\|\cdot\|^\dagger$ of a vector $\boldsymbol{g} \in \mathbb{R}^n$ is given by:*

$$\|\boldsymbol{g}\|^\dagger := \max_{\boldsymbol{t} \in \mathbb{R}^n : \|\boldsymbol{t}\| = 1} \boldsymbol{g}^\top \boldsymbol{t}.$$

Definition 2 (Duality map based on a norm)**.** *Given a norm $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$, we consider the duality map:*

$$\operatorname{dualize}_{\|\cdot\|} \boldsymbol{g} := \operatorname*{arg\,max}_{\boldsymbol{t} \in \mathbb{R}^n : \|\boldsymbol{t}\| = 1} \boldsymbol{g}^\top \boldsymbol{t},$$

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

Given these definitions, minimizing the expression in the right-hand side of Equation (2) can be done using the following standard proposition, for which Bernstein & Newhouse (2024) provide a proof:

Proposition 1 (Steepest descent under a norm)**.** *For any vector $\boldsymbol{g} \in \mathbb{R}^n$ thought of as "the gradient", any $\lambda \geqslant 0$ thought of as "the sharpness", and any norm $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ with dual norm $\|\cdot\|^\dagger$ and duality map* $\operatorname{dualize}_{\|\cdot\|}$*:*

$$\operatorname*{arg\,min}_{\Delta\boldsymbol{w} \in \mathbb{R}^n} \left[\boldsymbol{g}^\top \Delta\boldsymbol{w} + \frac{\lambda}{2} \|\Delta\boldsymbol{w}\|^2 \right] = -\frac{\|\boldsymbol{g}\|^\dagger}{\lambda} \times \operatorname{dualize}_{\|\cdot\|} \boldsymbol{g}.$$

In words: to find the minimizer of a linear term penalized by a squared norm, we need only evaluate the dual norm and a duality map. In this paper, we focus on constructing a duality map for the *modular norm*, which is a norm tailored to the weight space of general neural architectures. But first, we shall cover duality maps for more standard norms.

3.3 Basic Norms and Duality Maps

Many basic norms and duality maps are already covered in prior work (Carlson et al., 2016; 2015a;b; Flynn, 2017). For some warmup examples, the following duality maps for vector norms are standard:

Example 1 (Duality map for the Euclidean norm)**.** *For a nonzero vector $\boldsymbol{g} \in \mathbb{R}^d$, we have* $\operatorname{dualize}_{\|\cdot\|_2} \boldsymbol{g} = \boldsymbol{g}/\|\boldsymbol{g}\|_2$*. For the zero vector, we take* $\operatorname{dualize}_{\|\cdot\|_2} \mathbf{0} = \mathbf{0}$*.*

Example 2 (Duality map for the infinity norm)**.** *For a vector $\boldsymbol{g} \in \mathbb{R}^d$, we have* $\operatorname{dualize}_{\|\cdot\|_\infty} \boldsymbol{g} = \operatorname{sign}(\boldsymbol{g})$*, where the sign function is applied entrywise and we take* $\operatorname{sign}(0) = 0$*.*

In neural networks, the weight spaces of individual layers tend to have matrix structure. And layers with the same shape weight matrix may have semantically different input and output spaces—think *embedding* versus *linear* layers in a transformer. As such, we will need duality maps for different *induced operator norms*:

Definition 3 (Induced operator norm)**.** *Given a matrix $\boldsymbol{M} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ and two normed vector spaces $(\mathbb{R}^{d_{\text{in}}}, \|\cdot\|_\alpha)$ and $(\mathbb{R}^{d_{\text{out}}}, \|\cdot\|_\beta)$, the "$\alpha$ to β" induced operator norm is:*

$$\|\boldsymbol{M}\|_{\alpha \to \beta} = \max_{\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}}} \frac{\|\boldsymbol{M}\boldsymbol{x}\|_\beta}{\|\boldsymbol{x}\|_\alpha}.$$

For tensors, we define the duality map via

$$\operatorname{dualize}_{\|\cdot\|} \boldsymbol{G} := \operatorname*{arg\,max}_{\|\boldsymbol{T}\| = 1} \operatorname{flatten}(\boldsymbol{G})^\top \operatorname{flatten}(\boldsymbol{T}).$$

For linear layers, we will need the duality map for the RMS $\to$ RMS induced operator norm. This ends up as

a rescaled version of the spectral norm duality map from prior work (Carlson et al., 2015b; Flynn, 2017).

Example 3 (Duality map for the RMS $\to$ RMS operator norm)**.** *For a vector $\boldsymbol{v} \in \mathbb{R}^d$, we define the RMS norm to be the normalized Euclidean norm: $\|\boldsymbol{v}\|_{\text{RMS}} = \|\boldsymbol{v}\|_2/\sqrt{d}$. Given a matrix $\boldsymbol{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, the* RMS $\to$ RMS *induced operator norm resolves to a rescaled spectral norm: $\|\boldsymbol{W}\|_{\text{RMS}\to\text{RMS}} = \sqrt{d_{\text{in}}/d_{\text{out}}} \times \|\boldsymbol{W}\|_*$, where $\|\cdot\|_*$ denotes the standard spectral norm. For a matrix $\boldsymbol{G} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ with reduced singular value decomposition $\boldsymbol{G} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, the corresponding duality map is given by* $\text{dualize}_{\|\cdot\|_{\text{RMS}\to\text{RMS}}} \boldsymbol{G} = \sqrt{d_{\text{out}}/d_{\text{in}}} \times \boldsymbol{U}\boldsymbol{V}^\top$.

And for embedding layers, we will need the duality map for the $\ell_1 \to$ RMS operator norm:

Example 4 (Duality map for the $\ell_1 \to$ RMS operator norm)**.** *Given a matrix $\boldsymbol{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, the $\ell_1 \to$* RMS *induced operator norm resolves to the max* RMS *norm of the columns: $\|\boldsymbol{W}\|_{\ell_1\to\text{RMS}} = \max_i \|\text{col}_i(\boldsymbol{W})\|_{\text{RMS}}$. For a matrix $\boldsymbol{G} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, the corresponding duality map* $\text{dualize}_{\|\cdot\|_{\ell_1\to\text{RMS}}} \boldsymbol{G}$ *simply normalizes each column of $\boldsymbol{G}$ to have unit RMS norm: $\text{col}_i(\boldsymbol{G}) \mapsto \text{col}_i(\boldsymbol{G})/\|\text{col}_i(\boldsymbol{G})\|_{\text{RMS}}$ for each $i = 1, ..., d_{\text{in}}$.*

3.4 The Modular Norm

The *modular norm* (Large et al., 2024) is intended to help characterize the heterogeneous curvature of general neural architectures. The construction first defines an abstract *module* type along with a notion of what is a good, or *well-normed*, module. Then *combination rules* are given for constructing new well-normed modules from a library of existing well-normed modules. So modules are a special case of *combinator pattern* from functional programming (Haskell Wiki Contributors, 2007) and are related to a *monoidal category* from category theory (Fong & Spivak, 2019). Large et al. (2024) begin by defining an abstract *module*:

Definition 4 (Module)**.** *Given input vector space $\mathcal{X}$, output vector space $\mathcal{Y}$ and weight vector space $\mathcal{W}$, a module* M *is an object with the following four attributes:*

(a) *a function,* $\mathsf{M.forward} : \mathcal{W} \times \mathcal{X} \to \mathcal{Y}$*, which maps an input and a weight vector to an output;*

(b) *a number,* $\mathsf{M.mass} \geqslant 0$*, which is used to set the proportion of feature learning that this module contributes to any supermodule;*

(c) *a number,* $\mathsf{M.sensitivity} \geqslant 0$*, which estimates the module's sensitivity to input perturbations;*

(d) *a norm over the weight space,* $\mathsf{M.norm} : \mathcal{W} \to \mathbb{R}_{\geqslant 0}$*, sometimes abbreviated to just* $\|\cdot\|_{\mathsf{M}}$.

We shall care most about modules that are *well-normed* (Large et al., 2024), which amounts to requiring that the forward function is Lipschitz-continuous in the weights with constant 1 and in the inputs with constant M.sensitivity:

Definition 5 (Well-normed module)**.** *Let* M *be a module on $(\mathcal{X}, \mathcal{Y}, \mathcal{W})$, where the input and output spaces have respective norms $\|\cdot\|_{\mathcal{X}}$ and $\|\cdot\|_{\mathcal{Y}}$.* M *is well-normed if for all inputs $\boldsymbol{x} \in \mathcal{X}$, weights $\boldsymbol{w} \in \mathcal{W}$, weight perturbations $\Delta\boldsymbol{w} \in \mathcal{W}$ and input perturbations $\Delta\boldsymbol{x} \in \mathcal{X}$, we have both:*

$$\|\nabla_{\boldsymbol{w}} \mathsf{M.forward}(\boldsymbol{w}, \boldsymbol{x}) \diamond \Delta\boldsymbol{w}\|_{\mathcal{Y}} \leqslant \mathsf{M.norm}(\Delta\boldsymbol{w});$$
$$\|\nabla_{\boldsymbol{x}} \mathsf{M.forward}(\boldsymbol{w}, \boldsymbol{x}) \diamond \Delta\boldsymbol{x}\|_{\mathcal{Y}} \leqslant \mathsf{M.sensitivity} \times \|\Delta\boldsymbol{x}\|_{\mathcal{X}}.$$

The $\diamond$ operator denotes summation over any shared tensor indices[2]. This definition of well-normed-ness can be used as a guiding principle in the design of a library of atomic (i.e. handwritten) modules. First, norms should be assigned to the input and output space of each module based on the semantics of M.forward. Then a norm M.norm should be assigned to the module's weight space and a number M.sensitivity should be chosen to make the module well-normed. Examples are given in Section 4.1.

Given such a library of well-normed atomic modules, a compound module built through any arbitrary sequence of *module compositions* and *module concatenations* is automatically well-normed (Large et al., 2024). And if the atomic modules are not only well-normed but are also *smooth* (Large et al., 2024, Definition 5), then there is an automatic procedure for computing *sharpness coefficients* for any compound module built from the library (Large et al., 2024, Appendix C). The relevant definition of module composition is as follows:

Definition 6 (Module composition)**.** *Consider module* M_1 *with input, output and weight space $(\mathcal{X}_1, \mathcal{Y}_1, \mathcal{W}_1)$ and module* M_2 *with input, output and weight space $(\mathcal{X}_2, \mathcal{Y}_2, \mathcal{W}_2)$.* M_1 *and* M_2 *are composable if $\mathcal{X}_2 = \mathcal{Y}_1$. Their composite module $\mathsf{M} = \mathsf{M}_2 \circ \mathsf{M}_1$ has input, output and weight space $(\mathcal{X}_1, \mathcal{Y}_2, \mathcal{W}_1 \times \mathcal{W}_2)$ and attributes:*

(a) $\mathsf{M.forward}((\boldsymbol{w}_1, \boldsymbol{w}_2), \boldsymbol{x}))$
$= \mathsf{M}_2.\mathsf{forward}(\boldsymbol{w}_2, \mathsf{M}_1.\mathsf{forward}(\boldsymbol{w}_1, \boldsymbol{x}))$*;*

(b) $\mathsf{M.mass} = \mathsf{M}_1.\mathsf{mass} + \mathsf{M}_2.\mathsf{mass}$*;*

(c) $\mathsf{M.sensitivity} = \mathsf{M}_1.\mathsf{sensitivity} \times \mathsf{M}_2.\mathsf{sensitivity}$*;*

(d) $\mathsf{M.norm}((\boldsymbol{w}_1, \boldsymbol{w}_2)) = \max(\alpha, \beta)$*, where:*

1) $\alpha = \mathsf{M}_2.\mathsf{sensitivity} \times \frac{\mathsf{M.mass}}{\mathsf{M}_1.\mathsf{mass}} \times \mathsf{M}_1.\mathsf{norm}(\boldsymbol{w}_1)$

2) $\beta = \frac{\mathsf{M.mass}}{\mathsf{M}_2.\mathsf{mass}} \times \mathsf{M}_2.\mathsf{norm}(\boldsymbol{w}_2)$

and if $\mathsf{M}_1.\mathsf{mass}$ *or* $\mathsf{M}_2.\mathsf{mass}$ *is zero, the corresponding term in the* max *is set to zero.*

[2]The expressions inside the norms on the left-hand side are examples of "Jacobian vector products", or JVPs for short (Blondel & Roulet, 2024).

So the composite norm is taken to be a weighted max over the norms of the two sub-modules, where the weight space of the first module is coupled to the input sensitivity of the second module. The module masses provide freedom to tune the importance of each sub-module in the norm, and Large et al. (2024) prove that module mass provides precise control over the amount of feature learning that can happen in each sub-module.

Concatenation is defined in a similar way to composition:

Definition 7 (Module concatenation)**.** *Consider module* M_1 *with input, output and weight space* $(\mathcal{X}_1, \mathcal{Y}_1, \mathcal{W}_1)$ *and module* M_2 *with input, output and weight space* $(\mathcal{X}_2, \mathcal{Y}_2, \mathcal{W}_2)$*. We say that* M_1 *and* M_2 *are concatenatable if their input spaces match:* $\mathcal{X}_1 = \mathcal{X}_2$*. The tuple module* $\mathsf{M} = (\mathsf{M}_1, \mathsf{M}_2)$ *has input, output and weight space* $(\mathcal{X}_1, \mathcal{Y}_1 \times \mathcal{Y}_2, \mathcal{W}_1 \times \mathcal{W}_2)$ *and the following list of attributes:*

(a) $\mathsf{M}.\mathsf{forward}((\boldsymbol{w}_1, \boldsymbol{w}_2), \boldsymbol{x}))$
$= (\mathsf{M}_1.\mathsf{forward}(\boldsymbol{w}_1, \boldsymbol{x}), \mathsf{M}_2.\mathsf{forward}(\boldsymbol{w}_2, \boldsymbol{x}))$*;*

(b) $\mathsf{M}.\mathsf{mass} = \mathsf{M}_1.\mathsf{mass} + \mathsf{M}_2.\mathsf{mass}$*;*

(c) $\mathsf{M}.\mathsf{sensitivity} = \mathsf{M}_1.\mathsf{sensitivity} + \mathsf{M}_2.\mathsf{sensitivity}$*;*

(d) $\mathsf{M}.\mathsf{norm}((\boldsymbol{w}_1, \boldsymbol{w}_2)) = \max(\alpha, \beta)$*, where:*

1) $\alpha = \frac{\mathsf{M}.\mathsf{mass}}{\mathsf{M}_1.\mathsf{mass}} \times \mathsf{M}_1.\mathsf{norm}(\boldsymbol{w}_1)$

2) $\beta = \frac{\mathsf{M}.\mathsf{mass}}{\mathsf{M}_2.\mathsf{mass}} \times \mathsf{M}_2.\mathsf{norm}(\boldsymbol{w}_2)$

and if $\mathsf{M}_1.\mathsf{mass}$ *or* $\mathsf{M}_2.\mathsf{mass}$ *is zero, the corresponding term in the* $\max$ *is set to zero.*

A shortcoming of the paper by Large et al. (2024) is that the power of the modular norm is not fully leveraged. In particular, the authors do *modular normalization* of training, where weight updates to certain modules are naïvely divided by their norm. This paper makes fuller use of the geometry of the modular norm by constructing the corresponding duality map, which we call *modular dualization*.

4 Modular Dualization

In this section, we construct a duality map for general neural architectures. Our strategy is to first write down duality maps for atomic modules, i.e. individual layers. We then extend to arbitrary compound modules, i.e. full neural networks, by showing how duality maps should pass through composition and concatenation.

4.1 Duality Maps for Atomic Modules

To construct a duality map for an atomic module A, the idea is to first fix norms on the input and output spaces that respect the semantics of $\mathsf{A}.\mathsf{forward}$. We should select norms that describe both how large we would like the inputs and outputs to be, and in what geometry we would like the outputs to evolve. Then we place a norm on the weight space such that A is well-normed: this is typically the operator norm (Definition 3) induced by the input and output norms. Finally we are in position to solve for the duality map, which we shall call $\mathsf{A}.\mathsf{dualize}$. We now give some examples of this procedure for the basic layer types of Linear, Embed and Conv2D. The results are summarized in Table 1.

We start with the canonical example of an atomic module:

Example 5 (The Linear module)**.** *The* Linear *module sends inputs from* $\mathcal{X} = \mathbb{R}^{d_{\text{in}}}$ *to outputs in* $\mathcal{Y} = \mathbb{R}^{d_{\text{out}}}$*. The weight space is given by the matrix space* $\mathcal{W} = \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$*. We endow the* Linear *module with attributes:*

1. $\mathsf{Linear}.\mathsf{forward}(\boldsymbol{W}, \boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x}$*, matrix-vector product;*
2. $\mathsf{Linear}.\mathsf{sensitivity} = 1$*;*
3. $\mathsf{Linear}.\mathsf{mass} = \mu$*, where* $\mu \geqslant 0$ *is a hyperparameter;*
4. $\mathsf{Linear}.\mathsf{norm}(\boldsymbol{W}) = \|\boldsymbol{W}\|_{\text{RMS}\to\text{RMS}}$*, the* $\text{RMS} \to \text{RMS}$ *induced operator norm.*

Since the Linear *module is intended to map to and from vectors of roughly unit* RMS *norm, we place the* RMS *norm on both the input and output space:* $\|\cdot\|_{\mathcal{X}} = \|\cdot\|_{\text{RMS}}$ *and* $\|\cdot\|_{\mathcal{Y}} = \|\cdot\|_{\text{RMS}}$*. Then* Linear *is well-normed if the inputs and weights belong to the unit balls* $\left\{\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}} : \|\boldsymbol{x}\|_{\mathcal{X}} \leqslant 1\right\}$ *and* $\left\{\boldsymbol{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}} : \mathsf{Linear}.\mathsf{norm}(\boldsymbol{W}) \leqslant 1\right\}$*. Referring back to Section 3.3, the duality map corresponding to* $\mathsf{Linear}.\mathsf{norm}$ *is then given by:*

5. $\mathsf{Linear}.\mathsf{dualize}(\boldsymbol{G}) = \sqrt{\frac{d_{\text{out}}}{d_{\text{in}}}} \times \boldsymbol{U}\boldsymbol{V}^\top$*, where the gradient* $\boldsymbol{G} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ *has reduced SVD* $\boldsymbol{G} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$*.*

This single duality map recovers essential features of both *maximal update parameterization* (Yang & Hu, 2021, μP) and *Shampoo* (Gupta et al., 2018). In particular, the factor of $\sqrt{d_{\text{out}}/d_{\text{in}}}$ in $\mathsf{Linear}.\mathsf{dualize}$ recovers spectral update scaling (Yang et al., 2023) that leads to μP. (Initializing such that $\mathsf{Linear}.\mathsf{norm}(\boldsymbol{W}) = 1$ also recovers μP initialization scaling.) And the mapping $\boldsymbol{G} \mapsto \boldsymbol{U}\boldsymbol{V}^\top$ is equivalent to Shampoo without accumulation (Bernstein & Newhouse, 2024). As such, we believe duality maps may help reconcile different strands of deep learning research and provide a unifying basis for fast and scalable training algorithms.

The Embed module provides a useful counterpoint to the Linear module. The difference between the two modules stems from the fact that the input spaces of Embed and Linear have different semantics.

Example 6 (The Embed module)**.** *The* Embed *module sends inputs from* $\mathcal{X} = \mathbb{R}^{d_{\text{in}}}$ *to outputs in* $\mathcal{Y} = \mathbb{R}^{d_{\text{out}}}$*. The weight space is given by the matrix space* $\mathcal{W} = \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$*. We endow the* Embed *module with attributes:*

1. $\mathsf{Embed}.\mathsf{forward}(\boldsymbol{W}, \boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x}$*, matrix-vector product;*
2. $\mathsf{Embed}.\mathsf{sensitivity} = 1$*;*

Module	Weight Space $\mathcal{W}$	Module.norm	Module.dualize
Linear	$\mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$	$\boldsymbol{W} \mapsto \|\boldsymbol{W}\|_{\text{RMS}\to\text{RMS}}$	$\boldsymbol{G} \mapsto \sqrt{\frac{d_{\text{out}}}{d_{\text{in}}}} \times \boldsymbol{U}\boldsymbol{V}^\top$
Embed	$\mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$	$\boldsymbol{W} \mapsto \|\boldsymbol{W}\|_{\ell_1\to\text{RMS}}$	$\text{col}_j(\boldsymbol{G}) \mapsto \frac{\text{col}_j(\boldsymbol{G})}{\|\text{col}_j(\boldsymbol{G})\|_{\text{RMS}}}$
Conv2D	$\mathbb{R}^{d_{\text{out}} \times d_{\text{in}} \times k \times k}$	$\boldsymbol{W} \mapsto k^2 \max_{i,j=1}^{k} \|\boldsymbol{W}_{\cdot\cdot ij}\|_{\text{RMS}\to\text{RMS}}$	$\boldsymbol{G}_{\cdot\cdot ij} \mapsto \frac{1}{k^2}\sqrt{\frac{d_{\text{out}}}{d_{\text{in}}}} \times \boldsymbol{U}_{ij}\boldsymbol{V}_{ij}^\top$

Table 1: Duality maps for three atomic modules: Linear, Embed, and Conv2D. These atomic modules are sufficient to build CNNs and transformers. In Linear.dualize, $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ denotes the reduced SVD of the gradient matrix $\boldsymbol{G}$. In Conv2D.dualize, $\boldsymbol{U}_{ij}\boldsymbol{\Sigma}_{ij}\boldsymbol{V}_{ij}^\top$ denotes the reduced SVD of the slice of the gradient tensor $\boldsymbol{G}_{\cdot\cdot ij}$ at kernel indices i, j. Section 5 provides GPU-friendly algorithms for computing these duality maps using a family of Newton-Schulz iterations.

3. $\mathsf{Embed.mass} = \mu$*, where* $\mu \geqslant 0$ *is a hyperparameter;*
4. $\mathsf{Embed.norm}(\boldsymbol{W}) = \|\boldsymbol{W}\|_{\ell_1\to\text{RMS}}$*, the* $\ell_1 \to \text{RMS}$ *induced operator norm.*

Embed *is intended to map from one-hot vectors to vectors of roughly unit* RMS *norm, so we place the* ℓ_1 *norm on the input space and the* RMS *norm on the output space:* $\|\cdot\|_{\mathcal{X}} = \|\cdot\|_{\ell_1}$, $\|\cdot\|_{\mathcal{Y}} = \|\cdot\|_{\text{RMS}}$. *Then* Embed *is well-normed if the inputs and weights belong to the unit balls* $\{\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}} : \|\boldsymbol{x}\|_{\mathcal{X}} \leqslant 1\}$ *and* $\{\boldsymbol{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}} : \mathsf{Embed.norm}(\boldsymbol{W}) \leqslant 1\}$. *Referring back to Section 3.3, the duality map for* Embed.norm *is:*

5. $\mathsf{Embed.dualize}(\boldsymbol{G})$ *performs the mapping* $\text{col}_j(\boldsymbol{G}) \mapsto \frac{\text{col}_j(\boldsymbol{G})}{\|\text{col}_j(\boldsymbol{G})\|_{\text{RMS}}}$ *for each column index* $j = 1, ..., d_{\text{in}}$.

Finally, we consider a Conv2D module with a $k \times k$ kernel. Conv2D has a more involved tensor structure than Linear and Embed. The calculations work by slicing up the weight tensor into a collection of k^2 matrices.

Example 7 (The Conv2D module)**.** *The* Conv2D *module sends inputs from* $\mathcal{X} = \mathbb{R}^{W_{\text{in}} \times H_{\text{in}} \times d_{\text{in}}}$ *to outputs in* $\mathcal{Y} = \mathbb{R}^{W_{\text{out}} \times H_{\text{out}} \times d_{\text{out}}}$. *We think of this as mapping an input image of width* W_{in}*, height* H_{in} *and with* d_{in} *color channels to an output image of width* W_{out}*, height* H_{out} *and with* d_{out} *color channels. The weight space is given by the tensor space* $\mathcal{W} = \mathbb{R}^{d_{\text{out}} \times d_{\text{in}} \times k \times k}$*, where* k *is the kernel size. We endow* Conv2D *with attributes:*

1. $\mathsf{Conv2D.forward}(\boldsymbol{W}, \boldsymbol{x}) = \boldsymbol{W} \circledast \boldsymbol{x}$*, where* $\circledast$ *denotes 2D convolution;*
2. $\mathsf{Conv2D.sensitivity} = 1$*;*
3. $\mathsf{Conv2D.mass} = \mu$*, where* $\mu \geqslant 0$ *is a hyperparameter;*
4. $\mathsf{Conv2D.norm}(\boldsymbol{W}) = k^2 \max_{i,j=1}^{k} \|\boldsymbol{W}_{\cdot\cdot ij}\|_{\text{RMS}\to\text{RMS}}$*, the max* $\text{RMS} \to \text{RMS}$ *norm over kernel indices.*

We would like pixel intensities in the inputs and outputs to be order one and undergo order one change. We formalize this by taking the input and output norms to be the spatial maximum of the RMS norms of all the color channel vectors: $\|\boldsymbol{x}\|_{\mathcal{X}} = \max_{w=1}^{W_{\text{in}}} \max_{h=1}^{H_{\text{in}}} \|\boldsymbol{x}_{wh\cdot}\|_{\text{RMS}}$ *and* $\|\boldsymbol{y}\|_{\mathcal{Y}} = \max_{w=1}^{W_{\text{out}}} \max_{h=1}^{H_{\text{out}}} \|\boldsymbol{y}_{wh\cdot}\|_{\text{RMS}}$. *Then* Conv2D *is well-normed if the inputs and weights belong to the unit balls* $\{\boldsymbol{x} \in \mathbb{R}^{W_{\text{in}} \times H_{\text{in}} \times d_{\text{in}}} : \|\boldsymbol{x}\|_{\mathcal{X}} \leqslant 1\}$ *and* $\{\boldsymbol{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}} \times k \times k} : \mathsf{Conv2D.norm}(\boldsymbol{W}) \leqslant 1\}$. *Since the duality map for a max of norms decouples into one duality map per sub-norm, the duality map corresponding to* Conv2D.norm *is given by:*

5. $\mathsf{Conv2D.dualize}(\boldsymbol{G})$ *does* $\boldsymbol{G}_{\cdot\cdot ij} \mapsto \frac{1}{k^2}\sqrt{\frac{d_{\text{out}}}{d_{\text{in}}}} \times \boldsymbol{U}_{ij}\boldsymbol{V}_{ij}^\top$*, where* $\boldsymbol{G}_{\cdot\cdot ij}$ *has reduced SVD* $\boldsymbol{U}_{ij}\boldsymbol{\Sigma}_{ij}\boldsymbol{V}_{ij}^\top$.

4.2 Duality Maps for Bond Modules

Large et al. (2024) define another class of basic modules: *bond modules*. Bonds are handwritten modules without weights. An example of a bond is the ReLU nonlinearity. For a bond B, the weight space is the zero vector space $\mathcal{W} = \{0\}$ and the modular norm $\mathsf{B.norm} = 0 \mapsto 0$. As such, the corresponding duality map is also $\mathsf{B.dualize} = 0 \mapsto 0$. In a software package, one need not write norms or duality maps for bond modules.

4.3 Duality Maps for Compound Modules

Given two composable modules M_1 and M_2, the duality map for the composite $\mathsf{M} = \mathsf{M}_2 \circ \mathsf{M}_1$ is given by:

$$\mathsf{M.dualize}(\boldsymbol{g}_1, \boldsymbol{g}_2) = \left(\frac{\mathsf{M}_1.\mathsf{mass}}{\mathsf{M.mass}} \times \frac{\mathsf{M}_1.\mathsf{dualize}(\boldsymbol{g}_1)}{\mathsf{M}_2.\mathsf{sensitivity}}, \frac{\mathsf{M}_2.\mathsf{mass}}{\mathsf{M.mass}} \times \mathsf{M}_2.\mathsf{dualize}(\boldsymbol{g}_2)\right).$$

Given two concatenatable modules M_1 and M_2, the duality map for the tuple $\mathsf{M} = (\mathsf{M}_1, \mathsf{M}_2)$ is:

$$\mathsf{M.dualize}(\boldsymbol{g}_1, \boldsymbol{g}_2) = \left(\frac{\mathsf{M}_1.\mathsf{mass}}{\mathsf{M.mass}} \times \mathsf{M}_1.\mathsf{dualize}(\boldsymbol{g}_1), \frac{\mathsf{M}_2.\mathsf{mass}}{\mathsf{M.mass}} \times \mathsf{M}_2.\mathsf{dualize}(\boldsymbol{g}_2)\right).$$

The proofs of Section 4.3 follow in a straightforward manner from Definitions 6 and 7.

5 Fast Duality Maps

For modular dualization to be practically feasible, we need ways of computing duality maps quickly. Inspecting the duality maps listed in Table 1, we see that Embed.dualize is

easy to implement since it just involves computing vector norms of matrix columns. But Linear.dualize and Conv2D.dualize involve the projection:

$$\boldsymbol{G} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top \mapsto \boldsymbol{U}\boldsymbol{V}^\top, \tag{3}$$

where $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ is the reduced SVD of the matrix $\boldsymbol{G}$. Since computing SVDs can be slow (Carlson et al., 2015b; Flynn, 2017), we discuss three approximations to this map via sketching, iterations for inverse matrix roots, and a family of *rectangular Newton-Schulz* iterations.

5.1 Sketching

Sketching is a randomized method (Martinsson & Tropp, 2020) that can be used to build low-rank approximations to the SVD. Carlson et al. (2015b) used sketching to provide a fast approximation to their #-operator. More recent papers have experimented with sketching in the context of Shampoo-type algorithms (Feinberg et al., 2023). A potential downside of approximating Equation (3) via sketching is that randomized SVD methods usually try to accurately approximate the largest singular values of a matrix (Martinsson & Tropp, 2020, Section 11.2) while the value of Equation (3) may lie in its action on small singular values.

5.2 Iterations for Inverse Matrix Roots

If $\boldsymbol{G}$ is a full rank matrix with reduced SVD $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, then:

$$\begin{aligned}\boldsymbol{U}\boldsymbol{V}^\top &= (\boldsymbol{G}\boldsymbol{G}^\top)^{-1/4}\boldsymbol{G}\,(\boldsymbol{G}^\top\boldsymbol{G})^{-1/4} \\ &= (\boldsymbol{G}\boldsymbol{G}^\top)^{-1/2}\boldsymbol{G} = \boldsymbol{G}\,(\boldsymbol{G}^\top\boldsymbol{G})^{-1/2}.\end{aligned}$$

This equation provides a route to approximating the map $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top \mapsto \boldsymbol{U}\boldsymbol{V}^\top$ since one can compute inverse matrix roots such as $(\boldsymbol{G}\boldsymbol{G}^\top)^{-1/2}$ via Newton iteration (Lakić, 1998). This is discussed in Chapter 7 of Higham (2008)'s book and also see Anil et al. (2020)'s paper. Care must be taken with inverses when the matrix $\boldsymbol{G}$ is ill-conditioned.

5.3 Rectangular Newton-Schulz Iteration

We developed a "rectangular Newton-Schulz iteration" for computing $\boldsymbol{U}\boldsymbol{V}^\top$ by adapting Equation 5.22 in Higham (2008)'s book for computing the "matrix sign function". We later discovered this iteration has a long history (Kovarik, 1970; Björck & Bowie, 1971). The method works by first normalizing the matrix $\boldsymbol{G}$ according to $\boldsymbol{X}_0 = \boldsymbol{G}/\|\boldsymbol{G}\|_{\ell_2\to\ell_2}$ (or alternatively $\boldsymbol{X}_0 = \boldsymbol{G}/\|\boldsymbol{G}\|_F$) and then iterating:

$$\boldsymbol{X}_{t+1} = \frac{3}{2}\boldsymbol{X}_t - \frac{1}{2}\boldsymbol{X}_t\boldsymbol{X}_t^\top\boldsymbol{X}_t,$$

then as $t\to\infty$, the sequence $\boldsymbol{X}_t \to \boldsymbol{U}\boldsymbol{V}^\top$. To see this, one can plot the univariate cubic function $f(x) := \frac{3}{2}x - \frac{1}{2}x^3$ and see that, for $0 < x < \sqrt{3}$, iterating this cubic will push x closer and closer to $+1$. The final step is to realize that the effect of the matrix iteration is to apply this cubic $f(x)$ to each singular value of $\boldsymbol{X}_t$. This shows that the spectral normalization $\boldsymbol{X}_0 = \boldsymbol{G}/\|\boldsymbol{G}\|_{\ell_2\to\ell_2}$ is stronger than what is required: we need only ensure that $\boldsymbol{X}_0$ has singular values no greater than $\sqrt{3}$ for the iteration to converge.

This iteration has the advantage over sketching that it always works on all singular values, and since it does not compute inverse matrix roots the iteration is well-behaved even on low-rank matrices.

Finally, there are in fact a family of degree $2n+1$ polynomial iterations of the form

$$\boldsymbol{X}_{t+1} = a\boldsymbol{X}_t + b(\boldsymbol{X}_t\boldsymbol{X}_t^\top)\boldsymbol{X}_t + \cdots + z(\boldsymbol{X}_t\boldsymbol{X}_t^\top)^n\boldsymbol{X}_t$$

for suitable $a, b, \ldots, z$ instead of $a, b = \frac{3}{2}, -\frac{1}{2}$. One should choose coefficients $a, b, \ldots, z$ so that the univariate polynomial $g(x) = ax + bx^3 + \cdots + zx^{2n+1}$ is a suitable approximation to $\operatorname{sign}(x)$. One may further accelerate the iteration by "tuning" the coefficients $a, b, \ldots, z$ empirically.

6 Discussion

This paper develops the theory of *modular duality* and the procedure of *modular dualization* as means to construct duality maps for general neural architectures. Here, we comment on implications and connections.

6.1 Neural Network Speedrunning

We believe that the ideas in this paper can help in the design of faster training methods. In fact, based on our work, a new NanoGPT training speed record was recently set using a Newton-Schulz-based duality map, packaged into an open-source optimizer called Muon (Jordan et al., 2024b).

6.2 A Type System for Deep Learning

Part of the inspiration for this work is to build a fully-fledged *type system* for deep learning. We think that activation spaces should be typed by their intended norm and the intended size of activations in that norm. This information would help to construct well-normed modules (see Section 4.1). Modules should be typed according to Definition 4. And, as suggested in the introduction, gradients should be explicitly typed as dual vectors. A duality map should flip the type of a dual vector to a primal vector. We plan to use the Modula deep learning package (Large et al., 2024) as a testbed for these ideas.

6.3 Modular Duality: A Unifying Theoretical Framework for Fast and Scalable Training

An important topic in contemporary optimization research is the design of *fast* and *scalable* training methods for neural

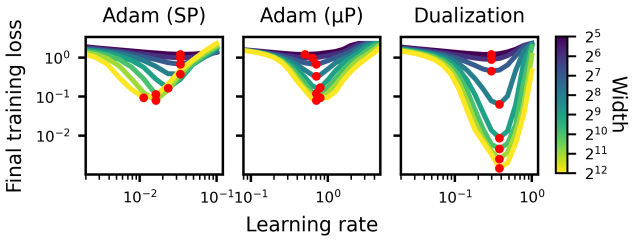

Figure 1: Learning rate transfer with dualization. To test learning rate transfer across width, we train an MLP on CIFAR-10 for 20 epochs at a range of widths and learning rates. We plot the final training loss and mark the best learning rate at each width with a red dot. **Left:** In standard parameterization (SP), Adam's optimal learning rate drifts to the left. **Middle**: Maximal update parameterization (Yang & Hu, 2021, μP) mostly corrects this drift. **Right**: Our duality-based method has a fairly stable optimal learning rate and also reaches much lower loss. More experimental details are given in Appendix A.

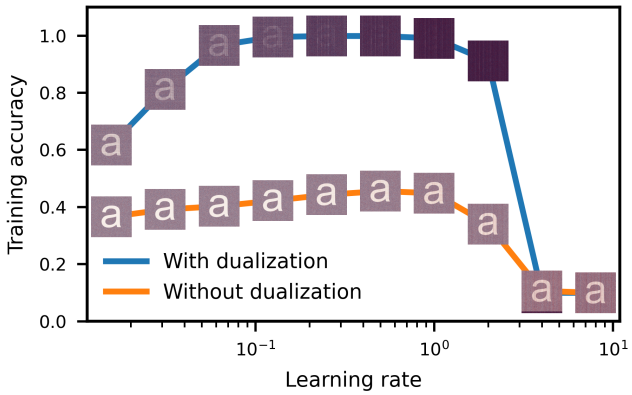

Figure 2: Erasure of watermarked initial weights. It is commonly held that the weights stay close to initialization in very wide networks (Lee et al., 2019; Jesus et al., 2021). To visualize the change in weights, we "watermark" the hidden layer weights of an MLP of width 1024 at initialization by zeroing out matrix entries in the shape of the letter "a". We then train for 1000 steps on CIFAR-10, across ten different 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 than the non-dualized method, but dualized gradient descent also "erases" the watermark at the highest stable learning rate, constituting substantial weight change. More experimental details are given in Appendix A.

networks. Two popular methods in this research space are *maximal update parameterization* (Yang & Hu, 2021, μP), which allows increasing network width without changing the optimal learning rate, and *Shampoo* (Gupta et al., 2018), a variant of which (Shi et al., 2023) won a speed challenge at the AlgoPerf optimizer competition (Dahl et al., 2023).

We showed in Section 4.1 that essential features of both μP and Shampoo are recovered from the single duality map Linear.dualize. We think that, on a basic theoretical level, μP and Shampoo should be viewed as partial approximations to this duality map. This observation helps put μP and Shampoo on a consistent theoretical footing, orients the methods with respect to overlooked prior work on spectral descent (Carlson et al., 2015b) and duality structure gradient descent (Flynn, 2017), and suggests new ways to generalize these methods to arbitrary layer types and network architectures via the modular norm and modular dualization.

Figure 1 shows that our duality-based optimizer is both scalable and fast: it transfers learning rate across width like μP, and it reaches lower loss in the same number of steps.

6.4 On the Alignment of Activations and Updates

Recent work (Yang et al., 2023; Everett et al., 2024; Large et al., 2024) has singled out the following question as important to the design of scalable deep learning systems: *to what extent do gradient updates to neural network layers align with incoming activation vectors?* This question is important since it helps inform how large weight updates need to be to induce a certain amount of change in layer outputs. Duality maps such as Linear.dualize and Conv2D.dualize may help simplify the answer to this question, since they project gradients to scaled semi-orthogonal matrices for which all singular values have the same magnitude. For the case of a square weight matrix, the weight update is simply an orthogonal matrix, which acts as an isometry on inputs vectors—meaning that feature learning happens trivially.

6.5 A Numerical Paradox: *The Weights Don't Change!*

Past work (Lee et al., 2019; Jesus et al., 2021) has pointed out an apparent paradox in deep learning: the weights seem to move a vanishing amount from initialization in the limit of large network width. This finding led to substantial interest in linearized training dynamics (Jacot et al., 2018). Prior work attempted to resolve this paradox by showing that the weights move a roughly constant amount at any width when the change is measured in spectral norm (Yang et al., 2023). But duality maps lead to a new story: Linear.dualize ramps up the stable rank of updates, causing the weights to move at large width *even in the Frobenius norm*—provided the batch size is not too small. This result, shown in Figure 2, challenges the belief that very wide neural networks cannot stray from their initialization; instead the numerical movement of the weights depends on the choice of optimizer.

7 Conclusion

This paper has proposed a recursive procedure called *modular dualization* for building duality maps for general neural architectures. The procedure unifies past strands of optimization research on Shampoo (Gupta et al., 2018) and μP (Yang & Hu, 2021). Duality-based optimizers have already led to significant wall-clock speedups in transformer training ranging from 124M to 1.5B parameters (Jordan et al., 2024b). The rectangular Newton-Schulz iteration provides a GPU-friendly and numerically stable means of dualizing under the $\mathrm{RMS} \to \mathrm{RMS}$ operator norm, while avoiding some of the downsides of sketching-based approaches (Carlson et al., 2015b). Overall, we hope that our theory of *modular duality* provides a clarifying toolkit for the design and analysis of deep learning systems.

Impact Statement

The methods developed in this paper may be used to make the training of machine learning systems more efficient, for either good or ill.

Acknowledgements

Many ideas in this paper, including Section 5.3, were developed jointly with Tim Large before he left to work at a tech company. We are grateful to Phillip Isola for invaluable discussions and for granting us the freedom to pursue this work. We also thank Jack Gallagher, Keller Jordan, Simo Ryu, Rogier Brussee, Tongzhou Wang, Victor Butoi, Jeffrey Cider and the anonymous reviewers for helpful conversations.

References

Amari, S. *Information Geometry and Its Applications*. Springer, 2016. Cited on page 1.

Anil, R., Gupta, V., Koren, T., Regan, K., and Singer, Y. Scalable second order optimization for deep learning. *arXiv:2002.09018*, 2020. Cited on page 7.

Bernstein, J. and Newhouse, L. Old optimizer, new norm: An anthology. In *Workshop on Optimization for Machine Learning*, 2024. Cited on pages 3 and 5.

Bernstein, J., Mingard, C., Huang, K., Azizan, N., and Yue, Y. Automatic Gradient Descent: Deep Learning without Hyperparameters. *arXiv:2304.05187*, 2023. Cited on page 2.

Björck, Å. and Bowie, C. An iterative algorithm for computing the best estimate of an orthogonal matrix. *SIAM Journal on Numerical Analysis*, 1971. Cited on page 7.

Blondel, M. and Roulet, V. The Elements of Differentiable Programming. *arXiv:2403.14606*, 2024. Cited on pages 1 and 4.

Boyd, S. and Vandenberghe, L. *Convex Optimization*. Cambridge University Press, 2004. Cited on page 1.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/jax-ml/jax`. Cited on page 1.

Carlson, D., Cevher, V., and Carin, L. Stochastic spectral descent for restricted Boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, 2015a. Cited on pages 2 and 3.

Carlson, D., Hsieh, Y.-P., Collins, E., Carin, L., and Cevher, V. Stochastic spectral descent for discrete graphical models. *Selected Topics in Signal Processing*, 2016. Cited on pages 2 and 3.

Carlson, D. E., Collins, E., Hsieh, Y.-P., Carin, L., and Cevher, V. Preconditioned spectral descent for deep learning. In *Neural Information Processing Systems*, 2015b. Cited on pages 1, 2, 3, 4, 7, 8, and 9.

Carroll, S. M. *Spacetime and Geometry: An Introduction to General Relativity*. Cambridge University Press, 2019. Cited on page 1.

Dahl, G. E., Schneider, F., Nado, Z., Agarwal, N., Sastry, C. S., Hennig, P., Medapati, S., Eschenhagen, R., Kasimbeg, P., Suo, D., Bae, J., Gilmer, J., Peirson, A. L., Khan, B., Anil, R., Rabbat, M., Krishnan, S., Snider, D., Amid, E., Chen, K., Maddison, C. J., Vasudev, R., Badura, M., Garg, A., and Mattson, P. Benchmarking neural network training algorithms. *arXiv:2306.07179*, 2023. Cited on page 8.

Deimling, K. *Nonlinear Functional Analysis*. Springer Berlin, Heidelberg, 1985. Cited on page 2.

Everett, K. E., Xiao, L., Wortsman, M., Alemi, A. A., Novak, R., Liu, P. J., Gur, I., Sohl-Dickstein, J., Kaelbling, L. P., Lee, J., and Pennington, J. Scaling exponents across parameterizations and optimizers. In *International Conference on Machine Learning*, 2024. Cited on page 8.

Feinberg, V., Chen, X., Sun, Y. J., Anil, R., and Hazan, E. Sketchy: Memory-efficient adaptive regularization with frequent directions. In *Neural Information Processing Systems*, 2023. Cited on page 7.

Flynn, T. The duality structure gradient descent algorithm: Analysis and applications to neural networks. *arXiv:1708.00523*, 2017. Cited on pages 1, 2, 3, 4, 7, and 8.

Fong, B. and Spivak, D. I. *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*. Cambridge University Press, 2019. Cited on page 4.

Grant, M. C. *Disciplined Convex Programming*. PhD dissertation, Stanford University, 2004. Cited on page 2.

Grosse, R. Metrics. Lecture 3 of CSC2541: Neural Net Training Dynamics, 2022. Cited on pages 1 and 2.

Gupta, V., Koren, T., and Singer, Y. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, 2018. Cited on pages 5, 8, and 9.

Haskell Wiki Contributors. Combinator pattern. Haskell Wiki, 2007. URL https://wiki.haskell.org/Combinator_pattern. Cited on page 4.

Higham, N. J. *Functions of Matrices*. Society for Industrial and Applied Mathematics, 2008. Cited on page 7.

Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. In *Neural Information Processing Systems*, 2018. Cited on page 8.

Jesus, R. J., Antunes, M. L., da Costa, R. A., Dorogovtsev, S. N., Mendes, J. F. F., and Aguiar, R. L. Effect of initial configuration of weights on training and function of artificial neural networks. *Mathematics*, 2021. Cited on page 8.

Jordan, K., Bernstein, J., Rappazzo, B., @fernbear.bsky.social, Vlado, B., Jiacheng, Y., Cesista, F., Koszarsky, B., and @Grad62304977. modded-nanogpt: Speedrunning the nanogpt baseline, 2024a. URL https://github.com/KellerJordan/modded-nanogpt. Cited on page 11.

Jordan, K., Jin, Y., Boza, V., You, J., Cecista, F., Newhouse, L., and Bernstein, J. Muon: An optimizer for hidden layers in neural networks, 2024b. URL https://web.archive.org/web/20250106014946/https://kellerjordan.github.io/posts/muon/. Cited on pages 2, 7, and 9.

Kovarik, Z. Some iterative methods for improving orthonormality. *SIAM Journal on Numerical Analysis*, 1970. Cited on page 7.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. Cited on page 11.

Lakić, S. On the computation of the matrix k-th root. *Journal of Applied Mathematics and Mechanics*, 1998. Cited on page 7.

Large, T., Liu, Y., Huh, M., Bahng, H., Isola, P., and Bernstein, J. Scalable optimization in the modular norm. In *Neural Information Processing Systems*, 2024. Cited on pages 2, 4, 5, 6, 7, and 8.

Lee, J., Xiao, L., Schoenholz, S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. Wide neural networks of any depth evolve as linear models under gradient descent. In *Neural Information Processing Systems*, 2019. Cited on page 8.

Martinsson, P.-G. and Tropp, J. A. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numerica*, 2020. Cited on pages 2 and 7.

Nemirovsky, A. S. and Yudin, D. B. *Problem complexity and method efficiency in optimization*. Wiley, 1983. Cited on page 1.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems*, 2019. Cited on page 1.

Qiu, S., Potapczynski, A., Finzi, M., Goldblum, M., and Wilson, A. G. Compute better spent: Replacing dense layers with structured matrices. In *International Conference on Machine Learning*, 2024. Cited on page 11.

Sakurai, J. J. and Napolitano, J. *Modern Quantum Mechanics*. Cambridge University Press, 2020. Cited on page 1.

Shi, H.-J. M., Lee, T.-H., Iwasaki, S., Gallego-Posada, J., Li, Z., Rangadurai, K., Mudigere, D., and Rabbat, M. A distributed data-parallel PyTorch implementation of the distributed Shampoo optimizer for training neural networks at-scale. *arXiv:2309.06497*, 2023. Cited on pages 1 and 8.

Streeter, M. Universal majorization-minimization algorithms. *arXiv:2308.00190*, 2023. Cited on page 2.

Streeter, M. J. and Dillon, J. V. Automatically bounding the Taylor remainder series: Tighter bounds and new applications. *arXiv:2212.11429*, 2022. Cited on page 2.

Tieleman, T. and Hinton, G. RMSprop. *Coursera: Neural Networks for Machine Learning*, Lecture 6.5, 2012. Cited on page 2.

Tran, D. T., Ono, N., and Vincent, E. Fast DNN training based on auxiliary function technique. *International Conference on Acoustics, Speech and Signal Processing*, 2015. Cited on page 2.

Yang, G. and Hu, E. J. Tensor programs IV: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, 2021. Cited on pages 1, 5, 8, and 9.

Yang, G., Simon, J. B., and Bernstein, J. A spectral condition for feature learning. *arXiv:2310.17813*, 2023. Cited on pages 5 and 8.

A Experimental Details

Datasets. The dataset for all experiments is CIFAR-10 (Krizhevsky & Hinton, 2009). We use the standard train and test splits with no data augmentation.

Architectures. The architecture for all experiments is a 3-layer MLP with a ReLU nonlinearity, one hidden layer, and no biases. Having three layers allows all three types of weight matrix shape to be present: square, wide rectangular, and tall rectangular. In the learning rate transfer experiment, the width of the hidden layer varies from 32 to 4096. In the weight erasure experiment, the width is fixed to 1024.

Precision. All experiments use the default precision `float32`. It is well demonstrated in the NanoGPT speedruns that Newton-Schulz iterations can run in `bfloat16` (Jordan et al., 2024a). We believe it is a promising future direction to explore whether dualization leads to improved speed and scalability in further reduced precisions.

A.1 Learning Rate Transfer

We run experiments with hidden layer widths 32, 64, 128, 256, 512, 1024, 2048, and 4096. For each width, we sweep between 10 and 20 different learning rates. We train for 20 epochs with batch size 128.

Adam (SP). The hyperparameters for Adam are the default ones in PyTorch, except for the sweep over the learning rate. Weight matrices are initialized according to the default PyTorch initialization (Kaiming uniform).

Adam (µP). In μP, the learning rate of each layer is equal to the global learning rate (which we sweep) divided by that layer's input dimension d_{in}, and weight matrices are initialized as zero-mean Gaussians with standard deviation $\sigma = \sqrt{\min(d_{\text{in}}, d_{\text{out}})/d_{\text{in}}^2}$ (Qiu et al., 2024). All other hyperparameters are the default ones in PyTorch.

Dualization. We use orthogonal weight initialization. Concretely, we create weight matrices with unit Gaussian entries and then iterate them through Newton-Schulz for 30 steps. Our duality-based optimizer in this experiment uses no momentum. It passes the raw gradient through the duality map $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top \mapsto \sqrt{d_{\text{out}}/d_{\text{in}}}\,\boldsymbol{U}\boldsymbol{V}^\top$, implemented via 5 steps of Newton-Schulz iteration and then multiplying by the dimensional constant. We use a quintic Newton-Schulz iteration with coefficients $(2, -1.5, 0.5)$.

A.2 Erasure of Watermarked Initial Weights

We run experiments with hidden layer width 1024 and learning rate ranging across $2^{-6}, \ldots, 2^3$. We train for 1000 steps with batch size 1024.

The dualization uses ten steps of a quintic Newton-Schulz iteration with coefficients $(3.0, -3.2, 1.2)$, followed by multiplication by the dimensional constant. These quintic coefficients are different from above but lead to the same duality map $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top \mapsto \sqrt{d_{\text{out}}/d_{\text{in}}} \times \boldsymbol{U}\boldsymbol{V}^\top$.

To align the maximum stable learning rate between dualized and non-dualized training, we spectrally normalize the regular gradient descent update as $\boldsymbol{g} \mapsto \boldsymbol{g}/\|\boldsymbol{g}\|_*/3$. We also divide by 3 to match the scalar division that occurs in `Linear.dualize` due to the module masses for a 3-layer MLP. This way both the dualized and non-dualized methods become unstable at the same learning rate, approximately $\mathsf{lr} = 1.1$, as seen in Figure 2. Disabling spectral normalization does not change the qualitative findings.