Scalable Optimization in the Modular Norm

Tim Large[⋆]
Columbia University

Yang Liu
Lawrence Livermore National Lab

Minyoung Huh
MIT CSAIL

Hyojin Bahng
MIT CSAIL

Phillip Isola
MIT CSAIL

Jeremy Bernstein[⋆]
MIT CSAIL

Abstract

To improve performance in contemporary deep learning, one is interested in scaling up the neural network in terms of both the number and the size of the layers. When ramping up the width of a single layer, graceful scaling of training has been linked to the need to normalize the weights and their updates in the "natural norm" particular to that layer. In this paper, we significantly generalize this idea by defining the *modular norm*, which is the natural norm on the full weight space of any neural network architecture. The modular norm is defined recursively in tandem with the network architecture itself. We show that the modular norm has several promising applications. On the practical side, the modular norm can be used to normalize the updates of any base optimizer so that the learning rate becomes transferable across width and depth. This means that the user does not need to compute optimizer-specific scale factors in order to scale training. On the theoretical side, we show that for any neural network built from "well-behaved" atomic modules, the gradient of the network is Lipschitz-continuous in the modular norm, with the Lipschitz constant admitting a simple recursive formula. This characterization opens the door to porting standard ideas in optimization theory over to deep learning. We have created a Python package called Modula that automatically normalizes weight updates in the modular norm of the architecture. The package is available via `pip install modula` with source code `here`.

1 Introduction

Given the practical impact of deep learning systems trained at the largest scale, there is a need for training algorithms that scale gracefully: without instability and—if possible—without manual tuning. However, current best practices for training have developed somewhat organically and do not live on a bedrock of sound numerical analysis. For example, while the Adam optimizer [1] is ubiquitous in the field, errors have been found in its proof of convergence [2], and empirically Adam has been found to scale poorly as either the width [3] or the depth [4] of the network is ramped up.

To remedy this situation, a patchwork of learning rate correction factors have recently been proposed [3–6]. The general idea is to retrofit a base optimizer such as Adam or SGD with special correction factors intended to render the optimizer's optimal learning rate invariant to scale. But this situation is not ideal: the correction factors are reportedly difficult to use. Lingle [7] suggests that this may be due to their "higher implementation complexity, many variations, or complex theoretical background". What's more, the correction factors are optimizer-specific, meaning that if one switches to a different optimizer one must either look up or recalculate a separate set of correction factors.

The goal of this paper is to simplify matters. We show that both Adam and SGD can be made to scale gracefully with width and depth by simply normalizing their updates in a special norm associated with

[⋆] denotes equal contribution. Correspondence to `{jbernstein,minhuh}@mit.edu`.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

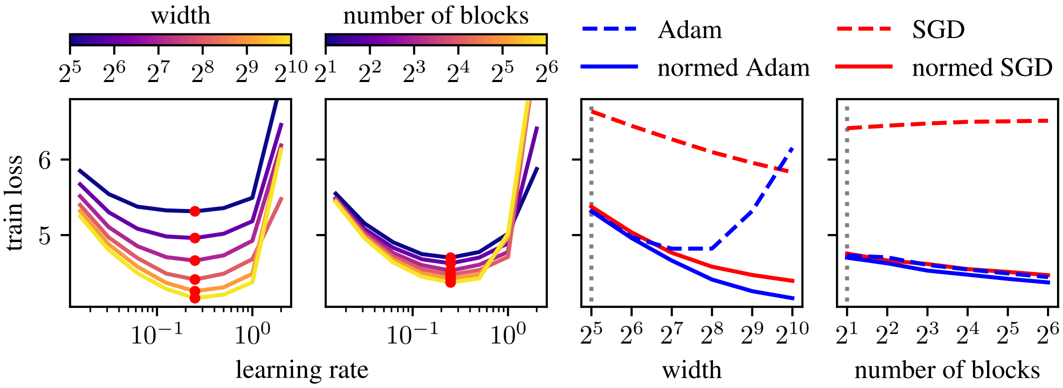

Figure 1: Learning rate transfer in the modular norm. We train GPT with context length 128 for 10k steps on OpenWebText. **Left:** Learning rate sweeps for normed Adam (Adam with updates normalized in the modular norm) with three transformer blocks and varying width. The optimal learning rate (marked by red dots) transfers well across scales. **Mid-left:** The same, but varying the number of blocks at width 128. **Mid-right:** Comparing normed versus unnormed Adam and SGD at fixed learning rate and varying width. For each method, we tune the learning rate at the scale marked by the dotted line. The normed methods scale better. **Right:** The same, but scaling number of blocks.

the network architecture—see Figure 1. We call this norm the *modular norm*, and provide a Python package called Modula that constructs this norm automatically and in tandem with the architecture.

The modular norm is constructed recursively, leveraging the module tree perspective on neural architectures. It is enough to define how the modular norm propagates through only two elementary operations: composition and concatenation. We show how other basic operations on modules, such as addition and scalar-multiplication, can be implemented through composition and concatenation. And then higher-order structures, such as residual networks, can be built using these basic operations.

Beyond its practical relevance, the modular norm may also prove useful to theoreticians. Various optimization-theoretic quantities are accessible and efficiently calculable in the modular norm. For instance, we show that the gradient of any neural network built from “well-behaved” atomic modules is Lipschitz-continuous in the modular norm of the architecture. This opens the door to porting several more-or-less textbook optimization theory analyses [8] over to the world of deep learning.

1.1 Related work

Metrization It is by now well-known that deep networks do not easily or naturally admit Lipschitz-continuity or smoothness guarantees in the Euclidean norm [9–13]. Researchers have attempted to address this problem: for instance, Bernstein et al. [12] propose a distance function called *deep relative trust*, which combines Frobenius norms across network layers. However, deep relative trust is only constructed for the multilayer perceptron and, when used to normalize updates, its employment of the Frobenius norm precludes good width scaling. In contrast, Yang et al. [14] equip individual layers with the RMS–RMS operator norm, finding this to enable good width scaling. Researchers have also looked at building neural net distance functions outside the context of scalability [15–17].

Asymptotics The metrization-based approach to scaling developed in this paper contrasts with the tradition of asymptotic scaling analyses—the study of infinite width and depth limits—more common in the deep learning theory literature [3–5, 18, 19]. These asymptotic analyses follow an old observation of Neal [20] that interesting properties of the neural network function space are exactly calculable in the infinite width limit and at initialization. This tradition has continued with asymptotic studies of the neural tangent kernel [21] as well as infinite depth limits [4, 5, 22]. However, there is increasing recognition of the limits of these limits, with researchers now often trying to relax limiting results [23–25]. And ultimately, from a practitioner's perspective, these results can be difficult to make sense of [7]. In contrast, our framework eschews any kind of limiting or probabilistic analysis. As a consequence, we believe our framework is simpler, more easily relatable to basic mathematical concepts, and ultimately more relevant to what one may encounter in, say, a PyTorch [26] program.

Majorization In recent work, Streeter and Dillon [27] propose a *universal majorize-minimize algorithm* [28]: a method that automatically computes and minimizes a majorizer for any computational graph. Despite its generality, current downsides to the method include its overhead, which can be 2× per step [29], as well as the risk that use of a full majorization may be overly pessimistic. Indeed, Cho and Shin [30] find that an optimization approach leveraging second-order information converges significantly faster than a majorization-inspired approach. Related ideas appear in [31, 32].

2 Descent in Normed Spaces

We define the modular norm in §3. This section is intended to prime the reader for what is to come. In this section, and the rest of the document, the diamond operator $\diamond$ denotes tensor contraction.

2.1 What's in a norm?

Suppose that we wish to use gradient descent to minimize a loss function $\mathcal{L} : \mathcal{W} \to \mathbb{R}$ over a weight space $\mathcal{W} = \mathbb{R}^N$. What properties of the loss $\mathcal{L}$ and weight space $\mathcal{W}$ would we desire for this to be sensible? Three such properties are:

(i) the loss function is differentiable, meaning that the gradient map $\nabla_{\boldsymbol{w}}\mathcal{L} : \mathcal{W} \to \mathcal{W}$ exists;

(ii) the weight space $\mathcal{W}$ carries a norm $\|\cdot\| : \mathcal{W} \to \mathbb{R}$, which need not be the Euclidean norm;

(iii) the loss is Lipschitz smooth in the norm $\|\cdot\|$, with sharpness constant $\lambda > 0$, meaning that:

$$\mathcal{L}(\boldsymbol{w} + \Delta\boldsymbol{w}) \leq \mathcal{L}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}) \diamond \Delta\boldsymbol{w} + \frac{\lambda}{2}\|\Delta\boldsymbol{w}\|^2. \tag{2.1}$$

Under these conditions, the weight update given by $\Delta\boldsymbol{w} = \arg\min\left[\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}) \diamond \Delta\boldsymbol{w} + \frac{\lambda}{2}\|\Delta\boldsymbol{w}\|^2\right]$ is guaranteed to reduce the loss. The particular norm $\|\cdot\|$ influences the direction of this weight update, while the sharpness constant λ influences the size of the update.

In deep learning, we would ideally like the optimal step-size to remain invariant as we scale, say, the width and the depth of the network. Thus, a fundamental problem is to design a norm such that, first, Inequality (2.1) actually holds (and is not hopelessly lax), and second, the corresponding sharpness constant λ is invariant to the relevant architectural dimensions. If the norm is chosen poorly, the practitioner may end up having to re-tune the step size as the network is scaled up. In this paper, we design a norm for neural networks that meets these requirements: the *modular norm*.

2.2 Preview of the modular norm

The weight space of a deep neural network is a Cartesian product $\mathcal{W} = \mathcal{W}_1 \times \ldots \times \mathcal{W}_L$, where $\mathcal{W}_k$ is the weight space at layer k. Yang et al. [14] consider the problem of metrizing individual layers. For instance, if layer k is a linear layer with weight space $\mathcal{W}_k = \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, then they equip this layer with the *RMS–RMS operator norm*, $\|\cdot\|_{\mathsf{RMS}\to\mathsf{RMS}}$. This is the matrix norm induced by equipping the input and output space of the layer with the root-mean-square (RMS) vector norm, $\|\boldsymbol{x}\|_{\mathsf{RMS}}^2 := \frac{1}{d}\Sigma_i\, \boldsymbol{x}_i^2$ for $\boldsymbol{x} \in \mathbb{R}^d$. The advantage of this non-standard matrix norm is that it allows one to estimate the amount of feature change induced by a gradient update. In other words, the inequality

$$\|\Delta\boldsymbol{W}\boldsymbol{x}\|_{\mathsf{RMS}} \leq \|\Delta\boldsymbol{W}\|_{\mathsf{RMS}\to\mathsf{RMS}} \cdot \|\boldsymbol{x}\|_{\mathsf{RMS}}, \tag{2.2}$$

turns out to hold quite tightly when $\Delta\boldsymbol{W}$ is a gradient update and $\boldsymbol{x}$ is a corresponding layer input. This is because gradient updates to a layer are (sums of) outer products that align with layer inputs.

Once we know how to metrize individual layers, a natural question is: can we combine layer-wise norms to produce a norm on the full weight space $\mathcal{W} = \prod_k \mathcal{W}_k$ of the network? Naïvely, there are many ways to do this: one could take any positive linear combination of the layer-wise norms (L^1 combination), the square root of any combination of the squared layer-wise norms (L^2 combination), and so on. But we want the norm to be useful by the criteria of §2.1. To this end, we propose the *modular norm* $\|\cdot\|_{\mathcal{W}}$, which ends up as a max (L^∞ combination) of scaled layer-wise norms $\|\cdot\|_{\mathcal{W}_k}$:

$$\|(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_L)\|_{\mathcal{W}} := \max\left(s_1\|\boldsymbol{w}_1\|_{\mathcal{W}_1}, \ldots, s_L\|\boldsymbol{w}_L\|_{\mathcal{W}_L}\right). \tag{2.3}$$

The positive scalar constants $s_1, \ldots, s_L$ are determined by both the architecture of the network and a set of user-specified "mass" parameters. The precise construction of the modular norm, working

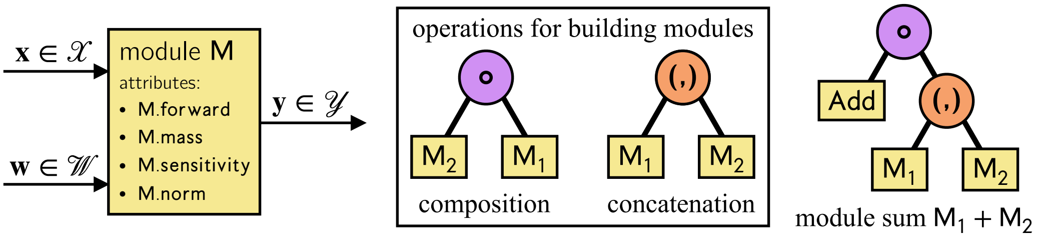

Figure 2: Modules and trees of modules. A module is an object that maps an input and a weight vector to an output. **Left:** In addition to the standard *forward* function, our modules are endowed with two numbers—a *mass* and *sensitivity*—and a *norm*. **Middle:** New *compound modules* are built via the binary operations of composition and concatenation. We provide rules for composing and concatenating all module attributes. **Right:** Compound modules are binary trees, where the leaves are modules and the internal nodes compose and concatenate their children. Here we illustrate a sum of modules, which leverages a special utility module Add—see Table 1 for more on this.

recursively over the module tree of the network, is given in §3; there, we also explain how the modular norm satisfies the criteria of §2.1, and the role played by the mass parameters. For now, let us explain what good the modular norm yields in practice.

2.3 Normed optimization

The main practical use of the modular norm is to normalize weight updates. With reference to Equation (2.3), we define the following operation on weight updates $\Delta\boldsymbol{w} = (\Delta\boldsymbol{w}_1, \dots, \Delta\boldsymbol{w}_L) \in \mathcal{W}$:

$$\mathsf{normalize}(\Delta\boldsymbol{w}) := \left(\frac{\Delta\boldsymbol{w}_1}{s_1\|\Delta\boldsymbol{w}_1\|_{\mathcal{W}_1}}, \dots, \frac{\Delta\boldsymbol{w}_L}{s_L\|\Delta\boldsymbol{w}_L\|_{\mathcal{W}_L}}\right). \tag{2.4}$$

Provided none of the $\Delta\boldsymbol{w}_k$ are zero, then $\mathsf{normalize}(\Delta\boldsymbol{w})$ is a unit vector in the modular norm. We propose using normalize as a wrapper, along with an explicit learning rate schedule, for any base optimizer such as Adam or SGD. The resulting *normed optimizer* is thus made architecture-aware via the normalize function. In pseudo-code—and actual Modula code—this amounts to:

```
delta_w = optim(w.grad())                # get update from base optimizer
net.normalize(delta_w)             # normalize update in the modular norm
w -= eta(step) * delta_w            # apply update with learning rate eta
```

We find this wrapper to significantly improve the scalability of the base optimizer. It renders the optimal learning rate roughly invariant to width and depth, with seemingly no cost to accuracy. In some instances, it enables training with a simpler optimizer—for example, training GPT with SGD rather than Adam—thus incurring a smaller memory footprint.

Normalization in the modular norm essentially forces individual layers to learn at specified, regulated rates. We view this as *balancing* learning across the network; no individual layer can learn too fast and destabilize training. This balance is determined by the architecture, along with user-specified mass parameters that provide precise control over the relative learning speed in different submodules.

For a variety of experiments with normed optimization, see §4 and Appendix D. But first, we detail the construction of the modular norm along with its core properties.

3 Constructing the Modular Norm

Our strategy is to first define the abstract notion of a *module*, which includes a norm as an attribute. We depict this concept in Figure 2. Then, by providing rules for composing and concatenating modules, we recursively define a norm for any module built via an arbitrary sequence of compositions and concatenations: the modular norm!

3.1 Modules

A *module* is a re-usable, composable object useful for building complicated neural networks. Our definition of a module augments the PyTorch module [26] with two real numbers and a norm:

Definition 1 (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—we often abbreviate this attribute to just* $\mathsf{M} \equiv \mathsf{M.forward}$*;*

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

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

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

Before we say more about the intended roles of these attributes, let us mention the three kinds of modules that we will care about in practice:

(i) *atomic modules*, whose attributes are hand-declared, and have weights. Examples include linear modules, embedding modules, and convolution modules.

(ii) *bond modules*, whose attributes are hand-declared, but have no weights. Formally, their weight space is the zero vector space $\mathcal{W} = 0$. An example is the ReLU non-linearity module.

(iii) *compound modules*, built out of other modules, with automatically inferred attributes.

Note that the space of objects that type-check as a module by Definition 1 is vast. Since we need to hand-declare atomic and bond modules in order to build interesting compound modules, we should have an idea of what makes for a "good" module. Simply put, a module is good when its attributes are predictive of its behaviour. To formalize this idea, we say that a module is *well-normed* if its forward function, sensitivity, and norm satisfy the following two relationships:

Definition 2 (Well-normed)**.** *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 $\boldsymbol{x} \in \mathcal{X}$ and weights $\boldsymbol{w} \in \mathcal{W}$:*

$$\|\nabla_{\boldsymbol{w}} \mathsf{M.forward}(\boldsymbol{w}, \boldsymbol{x}) \diamond \Delta \boldsymbol{w}\|_{\mathcal{Y}} \leq \mathsf{M.norm}(\Delta \boldsymbol{w}) \qquad \textit{for all } \Delta \boldsymbol{w} \in \mathcal{W}; \tag{3.1}$$

$$\|\nabla_{\boldsymbol{x}} \mathsf{M.forward}(\boldsymbol{w}, \boldsymbol{x}) \diamond \Delta \boldsymbol{x}\|_{\mathcal{Y}} \leq \mathsf{M.sensitivity} * \|\Delta \boldsymbol{x}\|_{\mathcal{X}} \qquad \textit{for all } \Delta \boldsymbol{x} \in \mathcal{X}. \tag{3.2}$$

Well-normed-ness means that the norm function and sensitivity are a good match for the forward function. The first inequality says that a well-normed module is Lipschitz-continuous over its weight space with a constant one. The second inequality says that a well-normed module is Lipschitz-continuous over its input space with constant $\mathsf{M.sensitivity}$. In practice, we will be interested in well-normed modules where these inequalities hold fairly tightly, since then $\mathsf{M.sensitivity}$ and $\mathsf{M.norm}$ will let us estimate the sensitivity of the module to input and weight perturbations. Appendix B provides many examples of well-normed atomic and bond modules.

The remaining attribute $\mathsf{M.mass}$ will turn out to control the proportion of feature learning that a module contributes to any compound module in which it participates. We formalize this concept in §3.3. But before that, we need to understand how to build compound modules.

3.2 Compound modules: Building new modules from old

We consider building new modules from old ones via the binary operations of composition and concatenation, illustrated in Figure 2. Composition is denoted via the serial combination $\mathsf{M}_2 \circ \mathsf{M}_1$, and concatenation via the parallel combination $(\mathsf{M}_1, \mathsf{M}_2)$, alternatively referred to as a *module tuple*. These simple binary combinations will let us build basic algebraic operations on modules (Table 1) as well as complex neural network architectures. We start by defining module composition:

Definition 3 (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 $\mathsf{M} = \mathsf{M}_2 \circ \mathsf{M}_1$ lives on $(\mathcal{X}_1, \mathcal{Y}_2, \mathcal{W}_1 \times \mathcal{W}_2)$ with attributes:*

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

Operation	Shorthand	Definition	Modula Expression
module addition	$\mathsf{M}_1 + \mathsf{M}_2$	$\mathsf{Add} \circ (\mathsf{M}_1, \mathsf{M}_2)$	`M_1 + M_2`
scalar multiplication	$a * \mathsf{M}$	$\mathsf{Mul}_a \circ \mathsf{M}$	`a * M`
iterated composition	M^L	$\mathsf{M} \circ \mathsf{M}^{L-1}$ with $\mathsf{M}^0 := \mathsf{Identity}$	`M ** L`

Table 1: Arithmetic with modules. Composition and concatenation let us define an extended arithmetic on modules. The utility modules Add, Mul_a and $\mathsf{Identity}$ are defined in Appendix B.2.

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

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

(d) $\mathsf{M.norm}((\boldsymbol{w}_1, \boldsymbol{w}_2))$ *given by:*

$$\max\left(\mathsf{M}_2.\mathsf{sensitivity} * \frac{\mathsf{M.mass}}{\mathsf{M}_1.\mathsf{mass}} * \mathsf{M}_1.\mathsf{norm}(\boldsymbol{w}_1), \frac{\mathsf{M.mass}}{\mathsf{M}_2.\mathsf{mass}} * \mathsf{M}_2.\mathsf{norm}(\boldsymbol{w}_2)\right),$$

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

At this stage, we make two comments about this definition. First, in the definition of the composite norm, notice that the norm of the first module couples with the sensitivity of the second module. This reflects the fact that the output of the first module is fed into the second module and not vice versa. Second, observe that the masses of the submodules are involved in setting the balance of the composite norm. Before we further motivate this definition, let us first define module concatenation:

Definition 4 (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* $\mathsf{M} = (\mathsf{M}_1, \mathsf{M}_2)$ *has input, output and weight space* $(\mathcal{X}_1, \mathcal{Y}_1 \times \mathcal{Y}_2, \mathcal{W}_1 \times \mathcal{W}_2)$ *and attributes:*

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

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

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

(d) $\mathsf{M.norm}(\boldsymbol{w}_1, \boldsymbol{w}_2)$ *given by:*

$$\max\left(\frac{\mathsf{M.mass}}{\mathsf{M}_1.\mathsf{mass}} * \mathsf{M}_1.\mathsf{norm}(\boldsymbol{w}_1), \frac{\mathsf{M.mass}}{\mathsf{M}_2.\mathsf{mass}} * \mathsf{M}_2.\mathsf{norm}(\boldsymbol{w}_2)\right),$$

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

Concatenation is simpler than composition in the sense that neither module is fed through the other, and therefore, sensitivity does not appear in the concatenated norm. To further motivate these definitions, observe that two basic and desirable properties follow as immediate consequences:

Proposition 1 (Composition and concatenation are associative)**.** *If modules* $\mathsf{M}_1, \mathsf{M}_2, \mathsf{M}_3$ *are successively composable, then* $\mathsf{M}_3 \circ (\mathsf{M}_2 \circ \mathsf{M}_1)$ *equals* $(\mathsf{M}_3 \circ \mathsf{M}_2) \circ \mathsf{M}_1$ *in all attributes. If modules* $\mathsf{M}_1, \mathsf{M}_2, \mathsf{M}_3$ *are mutually concatenatable, then* $((\mathsf{M}_1, \mathsf{M}_2), \mathsf{M}_3)$ *equals* $(\mathsf{M}_1, (\mathsf{M}_2, \mathsf{M}_3))$ *in all attributes.*

Proposition 2 (Composition and concatenation preserve well-normedness)**.** *If modules* M_1 *and* M_2 *are well-normed and composable, then their composite* $\mathsf{M}_2 \circ \mathsf{M}_1$ *is also well-normed. If modules* M_1 *and* M_2 *are well-normed and concatenatable, then their tuple* $(\mathsf{M}_1, \mathsf{M}_2)$ *is also well-normed with respect to the* L^1 *combination norm on the output space:* $\|(\cdot, \cdot)\|_{\mathcal{Y}_1 \times \mathcal{Y}_2} = \|\cdot\|_{\mathcal{Y}_1} + \|\cdot\|_{\mathcal{Y}_2}$.

The proofs follow directly from the definitions and the chain rule. Proposition 1 implies that one may build complicated compound modules without worrying in which order successive combinations are taken. Proposition 2 implies that complicated compounds automatically inherit Lipschitz guarantees.

Taken together, Definitions 3 and 4 define the *modular norm* $\mathsf{M.norm}$ of any compound module M.

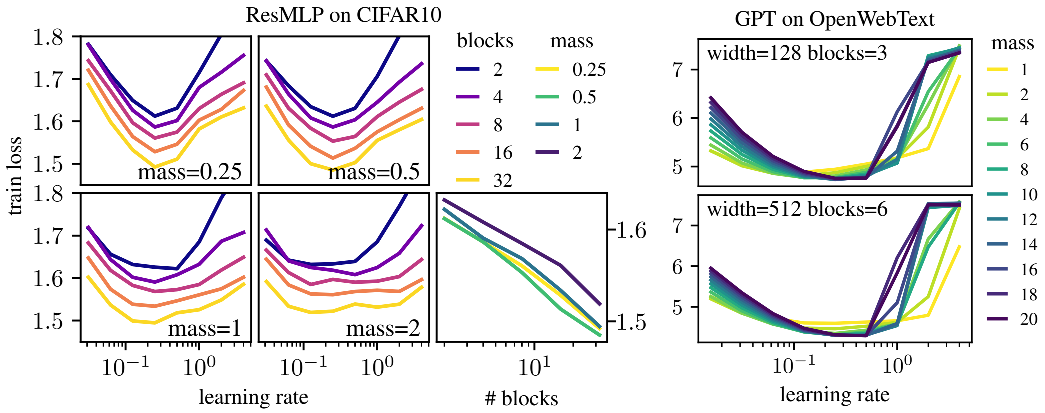

Figure 3: Exploring mass allocation. We tune the total mass of the hidden layers, training with normed Adam. **Left group:** Learning rate sweeps for ResMLP on CIFAR-10, for varying depth and mass. The bottom right subplot reports the best train loss at each mass and depth. Mass 0.5 was best at all depths. **Right group:** Learning rate sweeps for GPT on OpenWebText, for varying mass. Both optimal mass and learning rate transferred from the small model (top) to the large model (bottom).

3.3 Mass allocation in compound modules

Suppose we wish to train a network with an input layer, an output layer, and L blocks between:

$$\begin{aligned} \mathsf{Network} &= \mathsf{OutputLayer} \circ \mathsf{HiddenLayers} \circ \mathsf{InputLayer} && (3.3)\\ &= \mathsf{OutputLayer} \circ \mathsf{Block}^L \circ \mathsf{InputLayer}. && (3.4) \end{aligned}$$

Then how much learning should happen in the output layer, compared to the blocks, compared to the input layer? And what if we scale the number of blocks L—do we want relatively less learning to occur in the network's extremities? Or do we want the input and output layers to learn non-trivially even in the $L \to \infty$ limit? Since answering these questions is difficult a priori, we introduced the mass parameter to allow a user to set the proportional contribution each module has toward learning:

Proposition 3 (Feature learning is apportioned by mass)**.** *Consider a compound module* M *derived in any fashion from* L *well-normed modules* $\mathsf{M}_1, \ldots, \mathsf{M}_L$*. Given weight setting* $\boldsymbol{w} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_L)$*, where* $\boldsymbol{w}_k$ *denote the weights of module* M_k*, let us perturb* $\boldsymbol{w}$ *by* $\Delta\boldsymbol{w} = (\Delta\boldsymbol{w}_1, \ldots, \Delta\boldsymbol{w}_L)$*. If we decompose the linearized change in the output of module* M *into one contribution per sub-module:*

$$\nabla_{\boldsymbol{w}} \mathsf{M}(\boldsymbol{w}, \boldsymbol{x}) \diamond \Delta\boldsymbol{w} = \nabla_{\boldsymbol{w}_1} \mathsf{M}(\boldsymbol{w}, \boldsymbol{x}) \diamond \Delta\boldsymbol{w}_1 + \cdots + \nabla_{\boldsymbol{w}_L} \mathsf{M}(\boldsymbol{w}, \boldsymbol{x}) \diamond \Delta\boldsymbol{w}_L, \tag{3.5}$$

then the k*th term in this decomposition satisfies:*

$$\|\nabla_{\boldsymbol{w}_k} \mathsf{M}(\boldsymbol{w}, \boldsymbol{x}) \diamond \Delta\boldsymbol{w}_k\|_{\mathcal{Y}} \leq \frac{\mathsf{M}_k.\mathsf{mass}}{\mathsf{M}.\mathsf{mass}} * \mathsf{M}.\mathsf{norm}(\Delta\boldsymbol{w}). \tag{3.6}$$

In words: module mass provides the flexibility needed to build complicated compound modules involving many sub-modules, while maintaining precise control over how much learning any sub-module can contribute to the overall compound. Proposition 3 is proved in Appendix E.

In practice, we obtained the best training performance by maintaining a constant amount of learning in the input and output layers even as the number of blocks is scaled (Figure 6). In other words, it seems to be a good idea to assign $\mathsf{OutputLayer.mass} : \mathsf{HiddenLayers.mass} : \mathsf{InputLayer.mass}$ in proportion $1 : m : 1$, where m is independent of the number of blocks L. The exact mass of the hidden layers m needs to be tuned on a new architecture—just as one needs to tune separate learning rates in the input and output layers in μP [18]; this tuning can be done on a small model prior to scaling (Figure 3). We further discuss mass allocation in Appendix D.6.

3.4 Smoothness in the modular norm

In this section, we study the second derivatives of a module using the modular norm as a measuring stick. Let us start by defining the notion of sharpness that we will consider:

Definition 5 (Module sharpness)**.** *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* $\|\cdot\|_{\mathcal{Y}}$*. We say that* M *is* (α, β, γ)*-sharp for constants* $\alpha, \beta, \gamma \geq 0$ *if, at all inputs* $\boldsymbol{x} \in \mathcal{X}$ *and weights* $\boldsymbol{w} \in \mathcal{W}$*, the second derivatives of* M *are bounded as:*

$$\|\Delta \boldsymbol{w} \diamond \nabla^2_{\boldsymbol{ww}} \mathsf{M}(\boldsymbol{w}, \boldsymbol{x}) \diamond \Delta \widetilde{\boldsymbol{w}}\|_{\mathcal{Y}} \leq \alpha \, \|\Delta \boldsymbol{w}\|_{\mathsf{M}} \|\Delta \widetilde{\boldsymbol{w}}\|_{\mathsf{M}} \quad \textit{for all } \Delta \boldsymbol{w}, \Delta \widetilde{\boldsymbol{w}} \in \mathcal{W}; \tag{3.7}$$

$$\|\Delta \boldsymbol{w} \diamond \nabla^2_{\boldsymbol{wx}} \mathsf{M}(\boldsymbol{w}, \boldsymbol{x}) \diamond \Delta \boldsymbol{x}\|_{\mathcal{Y}} \leq \beta \, \|\Delta \boldsymbol{w}\|_{\mathsf{M}} \, \|\Delta \boldsymbol{x}\|_{\mathcal{X}} \quad \textit{for all } \Delta \boldsymbol{w} \in \mathcal{W} \textit{ and } \Delta \boldsymbol{x} \in \mathcal{X}; \tag{3.8}$$

$$\|\Delta \boldsymbol{x} \diamond \nabla^2_{\boldsymbol{xx}} \mathsf{M}(\boldsymbol{w}, \boldsymbol{x}) \diamond \Delta \widetilde{\boldsymbol{x}}\|_{\mathcal{Y}} \leq \gamma \, \|\Delta \boldsymbol{x}\|_{\mathcal{X}} \|\Delta \widetilde{\boldsymbol{x}}\|_{\mathcal{X}} \quad \textit{for all } \Delta \boldsymbol{x}, \Delta \widetilde{\boldsymbol{x}} \in \mathcal{X}. \tag{3.9}$$

While one may ultimately be interested in the sharpness of a module with respect to weight perturbations, Definition 5 also tracks sharpness with respect to input perturbations. In fact, tracking this extra information is essential for propagating sharpness bounds up the module tree. Appendix C details the procedure for automatically calculating the sharpness constants of a compound module starting from the sharpness constants of all its submodules; see Propositions 8 and 9 for the specific formulae. Here we highlight one major corollary of these formulae, proved in Appendix E: *for a specific choice of block multipliers, the sharpness constant of a residual network is independent of depth*:

Proposition 4. *Suppose* M *is a well-normed,* (α, β, γ)*-sharp module on* $(\mathcal{X}, \mathcal{X}, \mathcal{W})$ *with unit sensitivity. Define the depth* L residual module $\mathsf{Res}_L(\mathsf{M})$ *via the module arithmetic of Table 1 as:*

$$\mathsf{Res}_L(\mathsf{M}) := \left(\tfrac{L-1}{L} * \mathsf{Identity} + \tfrac{1}{L} * \mathsf{M}\right)^L. \tag{3.10}$$

Then this residual module $\mathsf{Res}_L(\mathsf{M})$ *is in fact* $(\alpha + \beta + \frac{\gamma}{3}, \beta + \frac{\gamma}{2}, \gamma)$*-sharp, independent of depth L.*

For optimization purposes, one may be more interested in the sharpness of the loss function rather than the sharpness of the neural network. Fortunately, it is possible to convert sharpness bounds on modules into sharpness bounds on loss functions, provided a little is known about the error measure:

Proposition 5 (Loss functions are smooth in the modular norm)**.** *Let* M *be a module on* $(\mathcal{X}, \mathcal{Y}, \mathcal{W})$ *and let* $\ell : \mathcal{Y} \times \mathcal{T} \to \mathbb{R}$ *measure the error between a module output and a target in target space* $\mathcal{T}$*. The loss* $\mathcal{L} : \mathcal{W} \to \mathbb{R}$ *records the module's average error on data distribution* $\mathcal{D}$ *over* $\mathcal{X} \times \mathcal{T}$*:*

$$\mathcal{L}(\boldsymbol{w}) := \mathbb{E}_{\boldsymbol{x}, \boldsymbol{t} \sim \mathcal{D}} \, \ell(\mathsf{M}(\boldsymbol{w}, \boldsymbol{x}), \boldsymbol{t}). \tag{3.11}$$

Suppose that the error measure ℓ *is* σ*-Lipschitz and* τ*-smooth in the module output, in the sense that:*

$$|\nabla_{\boldsymbol{y}} \ell(\boldsymbol{y}, \boldsymbol{t}) \diamond \Delta \boldsymbol{y}| \leq \sigma \, \|\Delta \boldsymbol{y}\|_{\mathcal{Y}} \quad \text{for all } \Delta \boldsymbol{y} \in \mathcal{Y} \text{ and } \boldsymbol{t} \in \mathcal{T}; \tag{3.12}$$

$$|\Delta \boldsymbol{y} \diamond \nabla^2_{\boldsymbol{yy}} \ell(\boldsymbol{y}, \boldsymbol{t}) \diamond \Delta \widetilde{\boldsymbol{y}}| \leq \tau \, \|\Delta \boldsymbol{y}\|_{\mathcal{Y}} \, \|\Delta \widetilde{\boldsymbol{y}}\|_{\mathcal{Y}} \quad \text{for all } \Delta \boldsymbol{y}, \Delta \widetilde{\boldsymbol{y}} \in \mathcal{Y} \text{ and } \boldsymbol{t} \in \mathcal{T}. \tag{3.13}$$

If the module M *is well-normed and* (α, β, γ)*-sharp, then the loss function* $\mathcal{L}$ *satisfies the following three inequalities at all weight settings* $\boldsymbol{w} \in \mathcal{W}$ *and for all weight perturbations* $\Delta \boldsymbol{w}, \Delta \widetilde{\boldsymbol{w}} \in \mathcal{W}$*:*

(i) $|\Delta \boldsymbol{w} \diamond \nabla^2_{\boldsymbol{ww}} \mathcal{L} \diamond \Delta \widetilde{\boldsymbol{w}}| \leq (\sigma\alpha + \tau) \, \|\Delta \boldsymbol{w}\|_{\mathsf{M}} \, \|\Delta \widetilde{\boldsymbol{w}}\|_{\mathsf{M}}$;

(ii) $\|\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w} + \Delta \boldsymbol{w}) - \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})\|^*_{\mathsf{M}} \leq (\sigma\alpha + \tau) \, \|\Delta \boldsymbol{w}\|_{\mathsf{M}}$,
where $\|\cdot\|^*_{\mathsf{M}}$ *is the dual norm of* $\|\cdot\|_{\mathsf{M}}$*;*

(iii) $|\mathcal{L}(\boldsymbol{w} + \Delta \boldsymbol{w}) - [\mathcal{L}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} \mathcal{L} \diamond \Delta \boldsymbol{w}]| \leq \frac{1}{2}(\sigma\alpha + \tau) \, \|\Delta \boldsymbol{w}\|^2_{\mathsf{M}}$.

The proof is given in Appendix E, and we present estimates for σ and τ for common error measures in Appendix C.4. Notice that inequalities (i), (ii) and (iii) are the standard inequalities of smooth optimization [8], albeit expressed in the modular norm. In fact, (i) implies (ii) implies (iii). In words, inequality (ii) says that the gradient of the loss is Lipschitz-continuous in the modular norm. The Lipschitz constant depends on the module only through the module's first sharpness coefficient α.

4 Experiments

Our experiments aimed to test the *scalability of training with normed versions of Adam and SGD*: whether one can tune the learning rate on a small model, and expect the learning rate to remain close to optimal on models of much larger width and depth. In addition to the learning rate, normed optimization in Modula requires a *mass parameter* to apportion feature learning between the input, output and hidden layers; we also tested the sensitivity of this parameter, whether it affects learning rate transfer, and to what extent the optimal mass itself transfers across width and depth.

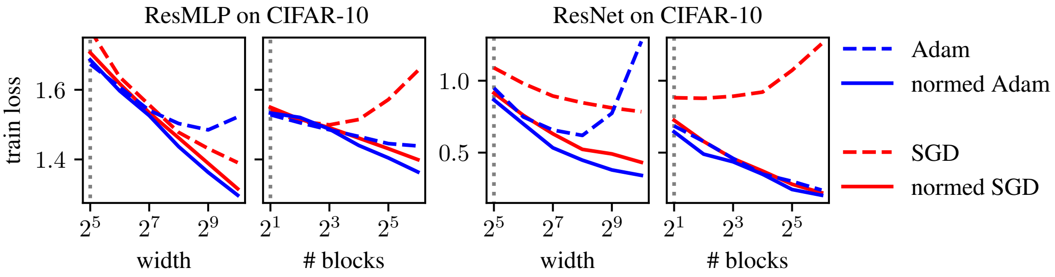

Figure 4: Learning rate transfer on CIFAR-10. We tune the learning rate on a small model—at the scale marked by the dotted line—and test the performance on models of increasing width and depth at this fixed learning rate. We find that normed Adam and SGD scale better than their unnormed counterparts on both ResMLPs and ResNets. See Figure 1 for the same experiment on GPT.

All SGD experiments were done with momentum $\beta = 0.9$, and all Adam experiments used $\beta_1 = 0.9$ and $\beta_2 = 0.99$. No weight decay was used in any experiment. Every experiment was done with a linear decay learning rate schedule. As for initialization, we used orthogonal initialization for Linear and Conv2D modules, and Gaussian weights projected to a unit norm ball for our Embed module. This was to ensure all modules were well-normed at initialization. Precise versions of our architectures are described in Appendices B.5 and B.7. We compare with nanoGPT using standard initialization in Appendix D.4 to make sure our changes recover standard performance. We actually found unnormed Adam using our GPT architecture transferred learning rate *better* than in nanoGPT.

We found that normed optimization, with both Adam and SGD as the base optimizer, allows for successful learning rate transfer across width and depth for GPT training on OpenWebText (Figure 1), as well as ResMLP and ResNet training on CIFAR-10 (Figure 4). We present expanded results in Appendix D.5, including results on test loss. We reproduce the standard finding that train and test loss are remarkably simillar in large language model pretraining. As for mass allocation, Figure 3 shows that optimal mass transfers with depth for training a ResMLP on CIFAR-10 with normed Adam, and also that both mass and learning rate transfer quite well from a smaller GPT on OpenWebText to a larger one. We detail more experiments on mass allocation in Appendix D.6.

5 Discussion: Limitations and Future Work

This paper was influenced by four main streams of work: first, the Tensor Programs series, starting at TP-IV [3, 4, 18]; second, the papers on universal majorize-minimize algorithms [27, 28]; third, work on deep network metrization [12, 14, 31]; and fourth, the open source deep learning ecosystem [26, 33, 34] including the PyTorch module tree and Karpathy's YouTube video on autograd [35]. We have distilled and synthesized key ideas from these sources, creating a framework that we believe to be simpler than Tensor Programs, computationally lighter than universal majorization-minimization, more general than prior work on metrization and more scalable than the PyTorch module tree. We have packaged these ideas into a (soon-to-be) open-source library called Modula. Inevitably, Modula has limitations. We highlight some of them here, along with associated avenues for future work.

Loss of well-normed-ness. We have emphasized well-normed-ness (Definition 2) as an important criterion in module design. We show in Appendix B.1 that, for example, the Linear module is well-normed when its weights lie within a spectral norm ball. In our experiments, we initialize all weights so that all modules are well-normed, but we do not enforce this property throughout training. Future work could explore regularization as a means to enforce well-normed-ness throughout training, with the hope of attaining better scalability or improved generalization.

Overhead of normalization. As discussed in Appendix A.3, we implement normalization for Linear and Conv2D modules using two steps of online power iteration. While online power iteration is an established and fast primitive in deep learning—in fact, coming from the GAN literature [36]—it does add a modest overhead to training time, as discussed in Appendix A.4. We think it may be possible to mitigate this overhead by constructing atomic modules with more exotic operator norms. For example, if one equips feature vectors with the L^∞ norm rather than the RMS norm, then the induced L^∞–L^∞ matrix norm is cheaper to compute than the RMS–RMS operator norm. In fact, L^∞–L^∞

operator normalization has the convenient feature that it decouples over matrix rows, making it more *local* than spectral normalization and, dare-we-say, more *biologically plausible*.

Automatic step-size selection. Beyond scalability, recent work has explored the question of automatic learning rate selection [31, 37–39], with the Prodigy optimizer [37] serving as a popular example. We tested the Adam version of Prodigy and found it performs well at small scales, essentially working by an implicit form of line search. However, Prodigy will always break at large enough widths, since it requires a lower bound (d_0) on Adam's initial learning rate; Yang et al. [3] showed that no such lower bound exists. We believe this issue could be fixed by rebuilding Prodigy on top of Modula. More broadly, we think that designing line search methods in a properly-normed space is a good idea.

Acknowledgements

We are grateful to Chris Mingard, Virgile Richard and Evan Kiely for useful discussions early in the project. Tongzhou Wang and Jyo Pari provided helpful feedback on the writing and figures.

The work was supported by a Packard Fellowship and a Sloan Research Fellowship to PI, by the MIT-IBM Watson AI Lab, by ONR MURI grant N00014-22-1-2740 and the MIT Quest for Intelligence. TL was supported by a Simons Junior Fellowship. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

We are grateful to the anonymous reviewers for their helpful and constructive feedback on this manuscript. Sadly, we have not had time to integrate much of their feedback into this camera-ready version of the paper. We will integrate the feedback into the arXiv version of the paper.

Contribution Statement

All authors were involved in project conception and discussions, which were initiated by JB. TL and JB developed the theory. MH and YL made core experimental observations. YL, MH, JB, and HB ran experiments. TL and JB did most of the writing, while JB, MH and YL made the figures. PI contributed guidance and helpful feedback throughout the course of the project. JB wrote the Modula package with help from MH.

References

[1] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. Cited on page 1.

[2] Sashank J. Reddi, Satyen Kale and Sanjiv Kumar. On the convergence of Adam and beyond. In *International Conference on Learning Representations*, 2018. Cited on page 1.

[3] Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu et al. Tuning large neural networks via zero-shot hyperparameter transfer. In *Neural Information Processing Systems*, 2021. Cited on pages 1, 2, 9, and 10.

[4] Greg Yang, Dingli Yu, Chen Zhu and Soufiane Hayou. Tensor programs VI: Feature learning in infinite depth neural networks. In *International Conference on Learning Representations*, 2024. Cited on pages 1, 2, 9, and 21.

[5] Blake Bordelon, Lorenzo Noci, Mufan Bill Li, Boris Hanin and Cengiz Pehlevan. Depthwise hyperparameter transfer in residual networks: Dynamics and scaling limit. In *International Conference on Learning Representations*, 2024. Cited on pages 1 and 2.

[6] Samy Jelassi, Boris Hanin, Ziwei Ji, Sashank J. Reddi, Srinadh Bhojanapalli et al. Depth dependence of μP learning rates in ReLU MLPs. *arXiv:2305.07810*, 2023. Cited on page 1.

[7] Lucas Lingle. A large-scale exploration of μ-transfer. *arXiv:2404.05728*, 2024. Cited on pages 1 and 2.

[8] Hamza Fawzi. Topics in convex optimisation. University of Cambridge, Lent 2023. Lecture 3. Cited on pages 2 and 8.

[9] Jeremy Cohen, Simran Kaur, Yuanzhi Li, J Zico Kolter and Ameet Talwalkar. Gradient descent on neural networks typically occurs at the edge of stability. In *International Conference on Learning Representations*, 2021. Cited on page 2.

[10] Haochuan Li, Jian Qian, Yi Tian, Alexander Rakhlin and Ali Jadbabaie. Convex and non-convex optimization under generalized smoothness. In *Neural Information Processing Systems*, 2023. Cited on page 2.

[11] Jingzhao Zhang, Tianxing He, Suvrit Sra and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity. In *International Conference on Learning Representations*, 2020. Cited on page 2.

[12] Jeremy Bernstein, Arash Vahdat, Yisong Yue and Ming-Yu Liu. On the distance between two neural networks and the stability of learning. In *Neural Information Processing Systems*, 2020. Cited on pages 2 and 9.

[13] Michael Vernon Nelson. Gradient conditioning in deep neural networks. Master's thesis, Brigham Young University, 2022. Cited on page 2.

[14] Greg Yang, James B. Simon and Jeremy Bernstein. A spectral condition for feature learning. *arXiv:2310.17813*, 2023. Cited on pages 2, 3, 9, and 16.

[15] Nikita Dhawan, Sicong Huang, Juhan Bae and Roger Grosse. Efficient parametric approximations of neural network function space distance. In *International Conference on Machine Learning*, 2023. Cited on page 2.

[16] Ari Benjamin, David Rolnick and Konrad Kording. Measuring and regularizing networks in function space. In *International Conference on Learning Representations*, 2019. Cited on page 2.

[17] Behnam Neyshabur, Ruslan Salakhutdinov and Nathan Srebro. Path-SGD: Path-normalized optimization in deep neural networks. *Neural Information Processing Systems*, 2015. Cited on page 2.

[18] Greg Yang and J. Edward Hu. Tensor programs IV: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, 2021. Cited on pages 2, 7, and 9.

[19] Jaehoon Lee, Jascha Sohl-Dickstein, Jeffrey Pennington, Roman Novak, Sam Schoenholz et al. Deep neural networks as Gaussian processes. In *International Conference on Learning Representations*, 2018. Cited on page 2.

[20] Radford M. Neal. *Bayesian Learning for Neural Networks*. Ph.D. thesis, Department of Computer Science, University of Toronto, 1994. Cited on page 2.

[21] Arthur Jacot, Franck Gabriel and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Neural Information Processing Systems*, 2018. Cited on page 2.

[22] Mufan Bill Li, Mihai Nica and Daniel M. Roy. The neural covariance SDE: Shaped infinite depth-and-width networks at initialization. In *Advances in Neural Information Processing Systems*, 2022. Cited on page 2.

[23] Aitor Lewkowycz, Yasaman Bahri, Ethan Dyer, Jascha Sohl-Dickstein and Guy Gur-Ari. The large learning rate phase of deep learning, 2021. Cited on page 2.

[24] Daniel A. Roberts, Sho Yaida and Boris Hanin. *The Principles of Deep Learning Theory*. Cambridge University Press, 2022. Cited on page 2.

[25] Chaoyue Liu, Libin Zhu and Mikhail Belkin. On the linearity of large non-linear models: When and why the tangent kernel is constant. *Neural Information Processing Systems*, 2020. Cited on page 2.

[26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury et al. PyTorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems*, 2019. Cited on pages 2, 5, and 9.

[27] Matthew J. Streeter and Joshua V. Dillon. Automatically bounding the Taylor remainder series: Tighter bounds and new applications. *arXiv:2212.11429*, 2022. Cited on pages 3 and 9.

[28] Matthew J. Streeter. Universal majorization-minimization algorithms. *arXiv:2308.00190*, 2023. Cited on pages 3 and 9.

[29] Matthew Streeter. Beyond automatic differentiation, 2023. URL `https://research.google/blog/beyond-automatic-differentiation/`. Cited on page 3.

[30] Namhoon Cho and Hyo-Sang Shin. Automatic optimisation of normalised neural networks. *arXiv:2312.10672*, 2023. Cited on page 3.

[31] Jeremy Bernstein, Chris Mingard, Kevin Huang, Navid Azizan and Yisong Yue. Automatic gradient descent: Deep learning without hyperparameters. *arXiv:2304.05187*, 2023. Cited on pages 3, 9, 10, and 16.

[32] Dung T. Tran, Nobutaka Ono and Emmanuel Vincent. Fast DNN training based on auxiliary function technique. *International Conference on Acoustics, Speech and Signal Processing*, 2015. Cited on page 3.

[33] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary et al. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`. Cited on page 9.

[34] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau et al. Theano: A Python framework for fast computation of mathematical expressions. *arXiv:1605.02688*, 2016. Cited on page 9.

[35] Andrej Karpathy. The spelled-out intro to neural networks and backpropagation: Building micrograd, 2018. URL `https://www.youtube.com/watch?v=VMj-3S1tku0`. Cited on page 9.

[36] Takeru Miyato, Toshiki Kataoka, Masanori Koyama and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018. Cited on page 9.

[37] Konstantin Mishchenko and Aaron Defazio. Prodigy: An expeditiously adaptive parameter-free learner. *arXiv:2306.06101*, 2024. Cited on page 10.

[38] Aaron Defazio and Konstantin Mishchenko. Learning-rate-free learning by D-adaptation. In *International Conference on Machine Learning*, 2023. Cited on page 10.

[39] Maor Ivgi, Oliver Hinder and Yair Carmon. DoG is SGD's best friend: A parameter-free dynamic step size schedule. In *International Conference on Machine Learning*, 2023. Cited on page 10.

[40] Kaiming He, X. Zhang, Shaoqing Ren and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *International Conference on Computer Vision*, 2015. Cited on page 19.

[41] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv:1606.08415*, 2016. Cited on page 19.

[42] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *arXiv:2205.14135*, 2022. Cited on page 22.

[43] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei et al. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. Cited on pages 23 and 27.

[44] Andrej Karpathy. nanoGPT code repository, 2022. URL `https://github.com/karpathy/nanoGPT`. Cited on pages 23, 27, and 28.

[45] Kaiming He, X. Zhang, Shaoqing Ren and Jian Sun. Deep residual learning for image recognition. *Computer Vision and Pattern Recognition*, 2015. Cited on page 27.

[46] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. Cited on page 27.

[47] Andrej Karpathy. Tiny Shakespeare. `https://huggingface.co/datasets/karpathy/tiny_shakespeare`, 2022. Cited on page 27.

[48] Ronen Eldan and Yuanzhi Li. TinyStories: How small can language models be and still speak coherent English? *arXiv:2305.07759*, 2023. Cited on page 27.

[49] Aaron Gokaslan and Vanya Cohen. OpenWebText corpus. `http://Skylion007.github.io/OpenWebTextCorpus`, 2019. Cited on page 27.

Contents of the Appendices

Appendix A The Modula Package

We created a Python package called Modula that realizes our module framework in code. Modula supplements PyTorch's `Tensor` class with two new classes: `Vector` and `Module`.

A.1 The `Vector` class

The `Vector` class is used to store the weights of a module. It allows for basic algebraic operations to be performed on module weights without needing to write `for` loops over lists of tensors. For example, if `v_1` and `v_2` are vectors with the same sub-structure, then one may write expressions such as `v_1 + v_2` for the vector sum, or `v_1 * v_2` for the elementwise product. Internally, a `Vector` stores a list of tensors and implements operations using efficient PyTorch `foreach` primitives.

A.2 The `Module` class

The most significant aspect of the Modula package is the `Module` class. A `Module` must have six attributes: two `float` attributes, namely `mass` and `sensitivity`. And four methods:

- `forward(w: Vector, x: Tensor) -> Tensor`   `# returns an output tensor`
- `initialize() -> Vector`   `# randomly samples a weight vector`
- `normalize(w: Vector)`   `# normalizes w to have unit modular norm`
- `regularize(w: Vector, strength: float)`   `# regularizes w in-place`

The norm of a module is not specifically implemented, instead we use the normalize method which is how the norm is directly used in optimization.

We refer to modules with hand-specified attributes as *bonds* if they have no weights and *atoms* if they have weights. Modules formed by combining existing modules are called *compounds*. Modula automatically constructs the attributes of compound modules. We provide reference implementations for many common modules—see Appendix B. We equip atoms with their natural operator norm, and compute spectral norms via online power iteration. Reference modules may be imported as follows:

```
from modula.bond      import Identity, ReLU, Abs, FunctionalAttention
from modula.atom      import Linear, Embed, Conv2D
from modula.compound  import ResMLP, ResCNN, Attention, GPT
```

To make building new compounds easier, Modula overloads the following operations on modules:

- `M_2 @ M_1`   `# composes module M_2 with module M_1`
- `(M_1, M_2)`   `# acts as a tuple module in any further composition`
- `M_1 + M_2`   `# returns the module sum`
- `a * M`   `# multiplies module M by scalar a`
- `M ** L`   `# returns the Lth iterate of module M`

For example, the compound

```
(L/(L-1) * Identity() + 1/L * M()) ** L
```

builds an `L`-layer residual network from base module `M`. Comparing with Equation (3.10), we see that Modula expressions closely resemble their mathematical counterparts.

Finally, all modules come with a convenience method `tare(m: float)`, which resets the module mass to `m`, with default `m=1`.

A.3 Normalization in Modula

We can normalize any base optimizer in the modular norm using the following pattern:

```
    delta_w = optim(w.grad())                # get update from base optimizer
    net.normalize(delta_w)                 # normalize update in the modular norm
    w -= lr * delta_w                                  # apply update to weights
```

Computation of `net.normalize(delta_w)` requires an efficient estimation of the spectral matrix norm, in the last two dimensions, of the constituent tensors of `delta_w`; this can be done very quickly to reasonable accuracy using power iteration. We implement this by storing a running estimate of the top singular vector `u` for each constituent tensor of `delta_w`. At initialization, `u` is sampled Gaussian, and each time we normalize a weight update, the previous update's estimated singular vector is used as the starting value for the power iteration. This enables us to use just two steps of power iteration per weight update. Indeed, for any base optimizer with momentum, successive weight updates should be fairly close; for training without momentum more steps of power iteration may be required.

A.4 Overhead

To test the overhead of normalization in the modular norm, we trained a width 64 ResMLP with 8 blocks and block-depth 2 for 10k steps on the CIFAR-10 dataset. We repeated the experiment with and without normalization, and in each case with three different random seeds. Without normalization, the training took 101 ± 1 seconds, and with normalization the training took 124 ± 1 seconds. So in this experiment, the overhead of modular normalization was around 23%.

We note that the user of the Modula package is free to write new atomic modules with cheaper or more efficient normalize functions. For instance, the Frobenius norm can be used as a proxy for the spectral norm whenever the weight updates have low stable rank [14, 31]. And we note in §5 that one could explore more exotic norms such as the L^∞–L^∞ operator norm, which is cheaper to compute than the standard spectral norm. Beyond these suggestions, one could explore CUDA-level optimizations to spectral norm computation, which is something that we have not explored.

Module M	M.forward	M.mass	M.sensitivity	M.norm
Linear	$\boldsymbol{W}, \boldsymbol{x} \mapsto \sqrt{\frac{d_{\text{out}}}{d_{\text{in}}}} \boldsymbol{W}\boldsymbol{x}$	1	1	$\boldsymbol{W} \mapsto \\|\boldsymbol{W}\\|_*$
Embed	$\boldsymbol{E}, \boldsymbol{x} \mapsto \sqrt{d}\, \boldsymbol{E}\boldsymbol{x}$	1	1	$\boldsymbol{E} \mapsto \max_i \\|\boldsymbol{E}_{\cdot i}\\|_2$
Conv2D	$\boldsymbol{C}, \boldsymbol{x} \mapsto \frac{1}{K^2} \sqrt{\frac{d_{\text{out}}}{d_{\text{in}}}} \boldsymbol{C} \circledast \boldsymbol{x}$	1	1	$\boldsymbol{C} \mapsto \max_{ij} \\|\boldsymbol{C}_{\cdot\cdot ij}\\|_*$

Table 2: Three atomic modules. These are the three atoms implemented in `Modula`—enough to build ResNet and GPT networks. By including explicit dimensional scale factors in the forward functions, we are able to use the standard spectral norm $\|\cdot\|_*$ and Euclidean norm $\|\cdot\|_2$, rather than their rescaled versions. d_{in} and d_{out} denote the input and output dimension of the Linear module. d denotes the embedding dimension of the Embed module. K denotes the kernel size of a Conv2D module with d_{out} output channels and d_{in} input channels. $\circledast$ denotes convolution.

Appendix B Module and Network Design

In this appendix, we list the basic, hand-declared modules that serve as building blocks for more complicated neural networks. Then we go on to show how these modules may be combined to yield interesting neural networks. This includes discussion of module broadcasting (Appendix B.3) and mass taring (Appendix B.4). The appendix culminates with case studies on attention (Appendix B.6) and transformers (Appendix B.7).

B.1 Atomic modules

An *atomic module* or *atom* for short is a module with *nonzero mass and nonzero parameter space*, whose attributes are specifically declared rather than derived. Setting an atom's mass to zero has the effect of freezing its weights under normed optimization.

Atom 1 (Linear)**.** For positive integers d_{out} and d_{in}, the *linear module* $\mathsf{Linear}(d_{\text{out}}, d_{\text{in}})$ corresponds to the standard linear layer with d_{in} input features and d_{out} output features. As a module, it has input space $\mathcal{X} = \mathbb{R}^{d_{\text{in}}}$, output space $\mathbb{R}^{d_{\text{out}}}$ and weights $\mathcal{W} = \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ the space of $d_{\text{out}} \times d_{\text{in}}$ matrices.

Its four attributes (forward function, mass, sensitivity, norm) are given in Table 2. Note the presence of the $\sqrt{d_{\text{out}}/d_{\text{in}}}$ factor in the forward function: this convention means that we can work with the *standard* L^2 operator norm $\|\cdot\|_*$ rather than the RMS-RMS operator norm.

Writing $\boldsymbol{f} = \mathsf{Linear}(d_{\text{out}}, d_{\text{in}}).\mathsf{forward}$, its derivative and second derivative at $(\boldsymbol{W}, \boldsymbol{x})$ are given by:

$$\nabla \boldsymbol{f} \diamond (\Delta \boldsymbol{W}, \Delta \boldsymbol{x}) = \sqrt{d_{\text{out}}/d_{\text{in}}}\, ((\Delta \boldsymbol{W})\, \boldsymbol{x} + \boldsymbol{W}\, (\Delta \boldsymbol{x}))\,, \tag{B.1}$$

$$(\Delta \boldsymbol{W}, \Delta \boldsymbol{x}) \diamond \nabla^2 \boldsymbol{f} \diamond (\Delta \widetilde{\boldsymbol{W}}, \Delta \widetilde{\boldsymbol{x}}) = \sqrt{d_{\text{out}}/d_{\text{in}}} \left((\Delta \boldsymbol{W})(\Delta \widetilde{\boldsymbol{x}}) + (\Delta \widetilde{\boldsymbol{W}})(\Delta \boldsymbol{x}) \right). \tag{B.2}$$

from which we conclude that $\mathsf{Linear}(d_{\text{out}}, d_{\text{in}})$ is well-normed, using the RMS norms on its input and output, so long as its arguments satisfy:

$$\|\boldsymbol{W}\|_*, \|\boldsymbol{x}\|_{\mathsf{RMS}} \leq 1. \tag{B.3}$$

These conditions will be automatically satisfied for many neural networks under *orthogonal initialization* of the weights, and especially if a linear module is immediately preceded by something like a LayerNorm module. Moreover, orthogonal initialization guarantees that the well-normed inequality

$$\|\nabla \boldsymbol{f} \diamond \Delta \boldsymbol{x}\|_{\mathsf{RMS}} \leq \|\boldsymbol{x}\|_{\mathsf{RMS}} \tag{B.4}$$

holds tightly in nearly-square matrices at initialization, which is important for getting good signal propagation through the whole network.

Moreover, inspection of the second derivative formula above shows it is always $(0, 1, 0)$-sharp with respect to the RMS norms on the input and output spaces.

Atom 2 (Embed)**.** For positive integers n and d, the *embedding module* $\mathsf{Embed}(n, d)$ corresponds to n class, token, or positional embeddings in a d-dimensional embedding space. As a module, it has input space $\mathbb{R}^n$, output space $\mathbb{R}^d$ and weights $\mathcal{W} = \mathbb{R}^{d\times n}$ the space of $d \times n$ matrices.

Its attributes are listed in Table 2.

This is at first sight similar to the linear module, the key difference being that in applications *we expect the inputs of* $\mathsf{Embed}(n, d)$ *to be one-hot vectors*; as such we consider its input space to carry the L^1-norm.

As for the linear module, $\mathsf{Embed}(n, d)$ is well-normed and $(0, 1, 0)$-sharp with respect to the L^1-norm on the input space $\mathbb{R}^n$ and the RMS norm on the output space $\mathbb{R}^d$.

Atom 3 (Conv2D)**.** For positive integers $d_{\text{out}}, d_{\text{in}}, K$ as well as H, W, the *2D-convolution module* $\mathsf{Conv2D}(d_{\text{out}}, d_{\text{in}}, K)$ corresponds to a convolutional layer with a $K \times K$ kernel; d_{in} and d_{out} are the number of channels for the input and output respectively (we suppress optional stride and padding arguments here for simplicity). Its input space is $\mathcal{X} = \mathbb{R}^{d_{\text{in}}\times H\times W}$, its output space is $\mathcal{Y} = \mathbb{R}^{d_{\text{out}}\times H\times W}$ and its weights are $\mathcal{W} = \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}\times K\times K}$.

Its attributes are listed in Table 2.

In fact, one could alternatively build $\mathsf{Conv2D}(d_{\text{out}}, d_{\text{in}}, K)$ starting from K^2 different $\mathsf{Linear}(d_{\text{out}}, d_{\text{in}})$ modules (of mass $1/K^2$ each) and concatenating them, and composing with a (parameter-less) convolution module. As such, Conv2D is well-normed and $(0, 1, 0)$-sharp. However, in our Modula package, we choose to explicitly declare Conv2D so as to take advantage of Pytorch's efficient implementation of convolution; the presentation here reflects this.

B.2 Bond modules

A *bond module* or *bond* is a module with zero mass and zero parameter space. They are the "glue" between the atomic modules, needed to construct complex neural networks.

Note that *we need not specify a weight space, or mass or norm arguments* for a bond module. Moreover, when discussing whether a bond module is (α, β, γ)-sharp, the inequalities for α and β are vacuous; thus for bond modules we will abbreviate this notion to γ-sharp.

To begin, we need two bond modules that are essentially "utility", as they are crucial for defining basic secondary module operations. These modules are also "type polymorphic" in the sense that they work with any underlying vector space.

Bond 1 (Add)**.** For any vector space $\mathcal{Y}$, the *adder module* Add has inputs $\mathcal{Y} \times \mathcal{Y}$ and outputs $\mathcal{Y}$. It has forward function

$$\mathsf{Add.forward} : (\boldsymbol{y}_1, \boldsymbol{y}_2) \mapsto \boldsymbol{y}_1 + \boldsymbol{y}_2 \tag{B.5}$$

and sensitivity 1. Its significance is that it allows for *concatenable modules to be added*:

$$\mathsf{M}_1 + \mathsf{M}_2 := \mathsf{Add} \circ (\mathsf{M}_1, \mathsf{M}_2). \tag{B.6}$$

For any norm $\|\cdot\|_{\mathcal{Y}}$ on the vector space $\mathcal{Y}$, Add is well-normed with respect to the L^1 combination norm $\|(\boldsymbol{y}_1, \boldsymbol{y}_2)\|_{\mathcal{Y}\times\mathcal{Y}} := \|\boldsymbol{y}_1\|_{\mathcal{Y}} + \|\boldsymbol{y}_2\|_{\mathcal{Y}}$ on its input space. Furthermore, Add is 0-sharp.

Bond 2 (Mul_λ)**.** For any normed vector space $\mathcal{Y}$ and real number λ the *scalar multiplier module* Mul_λ has inputs $\mathcal{Y}$ and outputs $\mathcal{Y}$. Its forward function is:

$$\mathsf{Mul}_\lambda\mathsf{.forward} : \boldsymbol{y} \mapsto \lambda * \boldsymbol{y} \tag{B.7}$$

and its sensitivity is $|\lambda|$. Its significance is that it allows for *scalar multiplication of modules*:

$$\lambda * \mathsf{M} := \mathsf{Mul}_\lambda \circ \mathsf{M}. \tag{B.8}$$

It is well-normed with respect to any norm on $\mathcal{Y}$, and 0-sharp. When $\lambda = 1$, we call this the *identity module* $\mathsf{Identity} = \mathsf{Mul}_1$. Note that $\lambda * \mathsf{Identity} = \mathsf{Mul}_\lambda$ for any λ.

The remaining bond modules are used explicitly as non-linearities in neural networks.

Bond 3 (Abs)**.** In any dimension d, the absolute value bond module Abs has inputs and outputs $\mathbb{R}^d$, forward function

$$\mathsf{Abs.forward} : (x_1, \ldots, x_d) \mapsto (|x_1|, \ldots, |x_d|) \tag{B.9}$$

and sensitivity 1. It is well-normed for any norm on $\mathbb{R}^d$.

Bond 4 (ReLU and $\mathsf{ScaledReLU}$)**.** In any dimension d, we define the "rectified linear unit" bond module ReLU to have input space $\mathcal{X} \subset \mathbb{R}^d$, output space $\mathcal{Y} = \mathbb{R}^d$, forward function

$$\mathsf{ReLU.forward} : (x_1, \dots, x_d) \mapsto (\max(0, x_i))_{i=1,\dots,d}. \tag{B.10}$$

and sensitivity $1/\sqrt{2}$. For this choice of sensitivity, ReLU is not well-normed with input space $\mathcal{X}$ set to the full $\mathbb{R}^d$. However, it is well-normed if the input space is, informally, a set of dense vectors with balanced signs. For illustration, ReLU is rigorously well-normed with respect to the input space

$$\mathcal{X} = \{\operatorname{sign} \boldsymbol{t} : \text{for } \boldsymbol{t} \in \mathbb{R}^d \text{ with } \# \text{ positive entries } = \# \text{ negative entries}\}, \tag{B.11}$$

and RMS norm on inputs and ouputs. For more on this design decision, see [40]. We also define $\mathsf{ScaledReLU} := \sqrt{2} * \mathsf{ReLU}$ to be the unit sensitivity counterpart to ReLU.

Bond 5 (GELU and $\mathsf{ScaledGeLU}$)**.** The "Gaussian error linear unit" bond module GELU [41] is essentially a smoothed version of ReLU. In any dimension d, GELU has inputs $\mathcal{X} \subset \mathbb{R}^d$, outputs $\mathcal{Y} = \mathbb{R}^d$ and forward function

$$\mathsf{GELU.forward} : (x_1, \dots, x_d) \mapsto (x_i \Phi(x_i))_{i=1,\dots,x_d} \tag{B.12}$$

where $\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt$ is the cumulative distribution function of the standard Gaussian.

GELU is well-normed in the same sense as ReLU. We similarly set $\mathsf{ScaledGeLU} = \sqrt{2} * \mathsf{GELU}$.

Bond 6 ($\mathsf{MeanSubtract}$)**.** For any dimension d, the mean subtraction module $\mathsf{MeanSubtract}$ has inputs and outputs $\mathbb{R}^d$. It centers its input to have mean zero. The forward function is given by:

$$\mathsf{MeanSubtract.forward} : (x_1, \dots, x_d) \mapsto (x_1 - \bar{\boldsymbol{x}}, \dots, x_d - \bar{\boldsymbol{x}}) \tag{B.13}$$

and has sensitivity 1. It is well-normed, and since it is a linear mapping, it is 0-sharp.

Bond 7 ($\mathsf{RMSDivide}$)**.** For any dimension d, the RMS division bond module $\mathsf{RMSDivide}$ has inputs and outputs $\mathbb{R}^d$. It normalizes its input to have unit RMS norm. The forward function is given by:

$$\mathsf{RMSDivide.forward} : \boldsymbol{x} \mapsto \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_{\mathsf{RMS}}} = \frac{\sqrt{d}\,\boldsymbol{x}}{\|\boldsymbol{x}\|_2}. \tag{B.14}$$

and has sensitivity 1. While it is not automatically well-normed, as long as its inputs have $\|\boldsymbol{x}\|_{\mathsf{RMS}} \approx 1$, the required inequality is not very far off. Similarly, it is approximately 1-sharp.

Bond 8 ($\mathsf{LayerNorm}$)**.** For any positive integer d, the layer normalization bond module $\mathsf{LayerNorm}$ has inputs and outputs $\mathbb{R}^d$, and is just defined as the composition of modules

$$\mathsf{LayerNorm} = \mathsf{RMSDivide} \circ \mathsf{MeanSubtract}. \tag{B.15}$$

As with $\mathsf{RMSDivide}$, it is approximately well-normed and approximately 1-sharp.

B.3 Module broadcasting

Let us briefly discuss a supplementary module operation, which we refer to as *module broadcasting*.

Definition 6. *Suppose* M *is a module with inputs* $\mathcal{X}$*, outputs* $\mathcal{Y}$ *and weights* $\mathcal{W}$*. Then for any* $h \geq 1$*, the* h*-times-broadcast of* M *is the module* $\mathsf{M}^{(h)}$ *with the same weight space* $\mathcal{W}$*, mass, sensitivity and norm as* M*, but inputs the Cartesian power* $\mathcal{X}^h = \mathcal{X} \times \ldots \times \mathcal{X}$ *and outputs* $\mathcal{Y}^h = \mathcal{Y} \times \ldots \times \mathcal{Y}$*, and forward function*

$$(\boldsymbol{w}, (\boldsymbol{x}_1, \dots, \boldsymbol{x}_h)) \mapsto (\mathsf{M.forward}(\boldsymbol{w}, \boldsymbol{x}_1), \dots, \mathsf{M.forward}(\boldsymbol{w}, \boldsymbol{x}_h)). \tag{B.16}$$

Since this is not defining a module with a new set of weights, we will usually just refer to the broadcast module by the same name M*, and consider this as just an extension of its forward function.*

For example, this allows us to define the action of linear modules $\mathsf{Linear}(d_{\text{out}}, d_{\text{in}})$ on inputs $\boldsymbol{x} \in \mathbb{R}^{\ell \times d_{\text{in}}}$ to give outputs $\boldsymbol{y} \in \mathbb{R}^{\ell \times d_{\text{out}}}$, where ℓ is the context length parameter for a transformer (see Appendix B.6, Appendix B.7, where it is also crucial for the construction of multi-headed attention). Additionally, one can view the basic Abs, ReLU and GELU modules as being broadcasts of the usual one-variable functions to take inputs and outputs in $\mathbb{R}^d$.

Let us briefly note:

Proposition 6. *If M is well-normed, then so is any broadcast of M taking $\mathcal{X}^h$ to $\mathcal{Y}^h$, as long as the norms on $\mathcal{X}^h$ and $\mathcal{Y}^h$ are taken to be either the "mean L^p" norms*

$$\|(\boldsymbol{x}_1, \dots, \boldsymbol{x}_h)\|_{\mathcal{X}^h} = \left(\frac{1}{h}(\|\boldsymbol{x}_1\|_{\mathcal{X}}^p + \ldots + \|\boldsymbol{x}_h\|_{\mathcal{X}}^p)\right)^{1/p} \tag{B.17}$$

$$\|(\boldsymbol{y}_1, \dots, \boldsymbol{y}_h)\|_{\mathcal{Y}^h} = \left(\frac{1}{h}(\|\boldsymbol{y}_1\|_{\mathcal{Y}}^p + \ldots + \|\boldsymbol{y}_h\|_{\mathcal{Y}}^p)\right)^{1/p} \tag{B.18}$$

for $1 \leq p \leq \infty$; when $p = \infty$ this is just the max norm. In the case that M is a bond module (so $\mathcal{W} = 0$, any scalar multiple of the mean L^p norm can be used (including the standard L^p norm).

The situation for sharpness is a bit more complicated; we discuss this in C.3.

B.4 Mass taring

In order to make working with the mass parameter of modules a bit easier, let us introduce an auxiliary operation:

Definition 7 (Tare)**.** *For any module M and positive real number m_{new}, the module $\mathsf{tare}(\mathsf{M}, m_{\text{new}})$ has the exact same inputs, outputs and weights as M; the same forward function, the same sensitivity and the same norm; but has mass*

$$\mathsf{tare}(\mathsf{M}, m_{\text{new}}).\mathsf{mass} = m_{\text{new}}. \tag{B.19}$$

This resets the mass of M. If M is a compound module, one could also reset the masses of all its submodules, by taking $\mathsf{tare}(\mathsf{M}_k, m_{\text{new}} * \frac{\mathsf{M}_k.\mathsf{mass}}{\mathsf{M}.\mathsf{mass}})$ for every submodule M_k, to "reconstruct" the computation graph for $\mathsf{tare}(\mathsf{M}, m_{\text{new}})$.

This way, one can build complex modules starting from atomic modules with unit masses, and then using tare later to reset their masses to desired quantities for better feature learning with normed descent as in Proposition 3.

B.5 Compound modules and neural networks

Composition, concatenation and the secondary operations of addition, scalar multiplication and iterated concatenation allow us to build a wide variety of neural networks which thus come automatically endowed with the modular norm.

Deep neural networks are typically built as long series of compositions. Let us introduce some terminology:

Definition 8 (Blocks and deep networks)**.** *A* deep neural network *is a module M formed by a composition*

$$\mathsf{M} = \mathsf{OutputLayer} \circ \mathsf{Block}_L \circ \ldots \circ \mathsf{Block}_1 \circ \mathsf{InputLayer} \tag{B.20}$$

where $\mathsf{InputLayer}, \mathsf{Block}_1, \ldots, \mathsf{Block}_L, \mathsf{OutputLayer}$ are modules; the number of blocks $L \geq 1$ is the depth *of the network.*

Typically, each of $\mathsf{Block}_1, \ldots, \mathsf{Block}_L$ will be copies of the same module (allowing them to take different weight values, of course), so that the network can be written as an iterated composition

$$\mathsf{M} = \mathsf{OutputLayer} \circ \mathsf{Block}^L \circ \mathsf{InputLayer}. \tag{B.21}$$

$\mathsf{InputLayer}, \mathsf{Block}, \mathsf{OutputLayer}$ can be principle be any module one likes, but usually $\mathsf{InputLayer}$ is often some form of embedding module, and $\mathsf{OutputLayer}$ is usually a linear module.

As for the form of Block, we found the following design principle to be quite useful in practice:

Arrange so that each Block *has unit sensitivity.*

This ensures that the sensitivity of the whole network stays bounded as $L \to \infty$ (this will also be the case if we ensure that $\mathsf{Block}.\mathsf{sensitivity} = 1 + O(1/L)$, but unit sensitivity has the advantage that the modular norm becomes very explicit). With this in mind:

Compound 1 (Residual network)**.** Suppose that M is a module of unit sensitivity whose inputs and outputs are the same space $\mathcal{X}$. For any $L \geq 1$, consider the *residual block*

$$\mathsf{Block} = \tfrac{L-1}{L} * \mathsf{Identity} + \tfrac{1}{L} * \mathsf{M} \tag{B.22}$$

and write $\mathsf{Res}_L(\mathsf{M}) = \mathsf{Block}^L$. This is of unit sensitivity, well-normed if M is, and moreover by Proposition 4 is sharp with O(1) sharpness if M is.

A general *residual network with residue* M is any neural network of the form

$$\mathsf{OutputLayer} \circ \mathsf{Res}_L(\mathsf{M}) \circ \mathsf{InputLayer}. \tag{B.23}$$

In practice, we will want to apply one more operation: we will want to *tare the mass of the residual blocks*. To this end, the *residual network with residue* M*, depth L and total block mass* $m > 0$ *is*

$$\mathsf{OutputLayer} \circ \mathsf{tare}(\mathsf{Res}_L(\mathsf{M}), m) \circ \mathsf{InputLayer}. \tag{B.24}$$

Let us give two basic example of residual networks.

Compound 2 (ResMLP)**.** This is a simple residual variation on the multi-layer perceptron. For a width $d \geq 1$, consider the unit sensitivity module

$$\mathsf{M}(d) = \mathsf{MeanSubtract} \circ \mathsf{Abs} \circ \mathsf{Linear}(d, d) \circ \mathsf{RMSDivide}. \tag{B.25}$$

This particular order of operations is inspired by a reecent paper of Yang et al. [4].

We invite the reader to compare this to something like $\mathsf{ReLU} \circ \mathsf{Linear}(d, d) \circ \mathsf{LayerNorm}$: three core operations are being performed (but in a different order in both cases): the inputs are being normalized; the inputs are being centered; and the inputs are passed through a nonlinearity that mutates just the negative coordinates.

The ResMLP network has as its residue an iterated composition of $\mathsf{M}(d)$, where the number of copies of $\mathsf{M}(d)$ in each residue is called the *block depth* and denoted B. It also has just linear initial and final modules. Thus the ResMLP network of depth L, width d, block depth B and total block mass $m > 0$ is

$$\mathsf{ResMLP} = \mathsf{Linear}(d_{\text{out}}, d) \circ \mathsf{tare}(\mathsf{Res}_L(\mathsf{M}(d)^B), m) \circ \mathsf{Linear}(d, d_{\text{in}}) \tag{B.26}$$

where d_{in} is the number of features of the data, and d_{out} the desired number of output features of the network.

Usually we suggest taking $B = 1$ or 2, and $m \sim 1$.

Compound 3 (ResNet)**.** This is a version of ResNet for image classification tasks. For a width $d \geq 1$ and kernel size K, consider similarly to above the unit sensitivity module

$$\mathsf{M}(d, K) = \mathsf{MeanSubtract} \circ \mathsf{Abs} \circ \mathsf{Conv2D}(d, d, K) \circ \mathsf{RMSDivide}. \tag{B.27}$$

As in the ResMLP, the ResNet network is a residual network with as its residue an iterated composition of B copies of $\mathsf{M}(d, K)$ where B is the *block depth.* Its initial and final modules are given by

$$\mathsf{InputLayer} = \mathsf{Conv2D}(d, c_{\text{in}}, K) \tag{B.28}$$

$$\mathsf{OutputLayer} = \mathsf{Linear}(d_{\text{out}}, d_{\text{total}}) \circ \mathsf{AvgPool} \tag{B.29}$$

where AvgPool is an additional bond module implementing adaptative average pooling. Here, c_{in} is the number of channel dimensions of the input image, $d_{\text{total}} = d * H * W$ is the total dimension of the hidden representation, and d_{out} is the desired number of output features (note that in Modula we include an additional dummy module `Flatten` to change the tensor shape before passing through the final layer). The ResNet network of depth L, width d, block depth B, kernel size K and total block mass m is thus:

$$\mathsf{ResNet} = \mathsf{OutputLayer} \circ \mathsf{tare}(\mathsf{Res}_L(\mathsf{M}(d, K)^B), m) \circ \mathsf{InputLayer}. \tag{B.30}$$

As defaults, we suggest taking $B = 2, K = 3$ and $m \sim 20$.

B.6 Case study I: Attention

Let us now focus on the construction of a single *multi-headed attention module* in this framework. The attention module should have, as both inputs and outputs, $\mathcal{X} = \mathbb{R}^{\ell \times d}$ where ℓ is the context length and d is the embedding dimension. The attention module itself will depend on three additional dimensional arguments:

- h, the number of heads;
- d_Q, the key/query dimension;
- d_V, the value dimension;

as well as an $\ell \times \ell$ matrix $\mathbf{mask}$, which we usually take to be either

$$\mathbf{mask}_{ij} = \begin{cases} 0 & \text{if } i \geq j \\ -\infty & \text{otherwise} \end{cases} \tag{B.31}$$

for causal attention, and $\mathbf{mask} = 0$ for non-casual attention.

The core of the attention module is a bond module which we call *functional attention*.

Bond 9 (FuncAttention)**.** Take positive integers ℓ, d_Q, d_V and mask matrix $\mathbf{mask}$. The corresponding *functional attention* is the bond module of unit sensitivity, inputs $\mathcal{X} = \mathbb{R}^{\ell \times d_Q} \times \mathbb{R}^{\ell \times d_Q} \times \mathbb{R}^{\ell \times d_V}$, outputs $\mathcal{Y} = \mathbb{R}^{\ell \times d_V}$, and forward function

$$\mathsf{FuncAttention.forward}(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}) = \operatorname{softmax}\left(\frac{\boldsymbol{q}\boldsymbol{k}^\top}{d_Q} + \mathbf{mask}\right)\boldsymbol{v}. \tag{B.32}$$

Moreover, we set $\mathsf{FuncAttention.sensitivity} = 1$.

In theory, one could try break up attention further into constituent more basic modules (such as scaled dot product, softmax, etc), but keeping FuncAttention as the basic unit one to leverage efficient implementations of attention such as FlashAttention [42].

In fact, a perhaps surprising result is that with the above $\frac{1}{d_Q}$ scaling of the dot product, we can estimate the sensitivity and sharpness of FuncAttention. This relies on giving norms for the input and output spaces; these norms are chosen to be

$$\|(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v})\|_{\mathcal{X}} = \|\boldsymbol{q}\|_{\infty\mathsf{RMS}} + \|\boldsymbol{k}\|_{\infty\mathsf{RMS}} + \|\boldsymbol{v}\|_{\infty\mathsf{RMS}}, \quad \|\boldsymbol{y}\|_{\mathcal{Y}} = \|\boldsymbol{y}\|_{\infty\mathsf{RMS}} \tag{B.33}$$

where $\|\cdot\|_{\infty\mathsf{RMS}}$ is the *infinity-RMS-norm* on $\mathbb{R}^{\ell \times d}$ defined from the standard root-mean-square norm $\|\cdot\|_{\mathsf{RMS}}$ on $\mathbb{R}^d$ by

$$\|\boldsymbol{x}\|_{\infty\mathsf{RMS}} := \max_{i=1,\dots,\ell} \|\boldsymbol{x}_{i*}\|_{\mathsf{RMS}}. \tag{B.34}$$

Proposition 7. *Over the space of inputs $\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}$ with each $\|\boldsymbol{q}\|_{\infty\mathsf{RMS}}, \|\boldsymbol{k}\|_{\infty\mathsf{RMS}}, \|\boldsymbol{v}\|_{\infty\mathsf{RMS}} \leq 1$, the functional attention module* FuncAttention *is well-normed, and moreover is sharp with sharpness constant $\gamma = 3$.*

The proof is given in Appendix E. We thus choose to adopt a $\frac{1}{d_Q}$-dot-product scaling in our implementation of attention– a rigorous bound as above is not possible for $\frac{1}{\sqrt{d_Q}}$-scaling, for instance.

We can then immediately define a single head of attention.

Compound 4 (Single-headed attention)**.** For positive integers ℓ, d, d_Q, d_V and a choice of $\mathbf{mask}$, take four instances of the linear module, for the query, key, value and exit parameters:

$$\mathsf{Query} = \mathsf{Linear}(d_Q, d) \tag{B.35}$$

$$\mathsf{Key} = \mathsf{Linear}(d_Q, d) \tag{B.36}$$

$$\mathsf{Value} = \mathsf{Linear}(d_V, d) \tag{B.37}$$

$$\mathsf{Exit} = \mathsf{Linear}(d, d_V) \tag{B.38}$$

which by broadcasting we consider to have inputs of shape $\mathbb{R}^{\ell \times d}$. The single-headed attention Attention module is then the composition

$$\mathsf{Attention} = \mathsf{Exit} \circ \frac{1}{3} * \mathsf{FuncAttention} \circ (\mathsf{Query}, \mathsf{Key}, \mathsf{Value}). \tag{B.39}$$

The scalar multiplication factor of $\frac{1}{3}$ ensures that Attention has unit sensitivity.

For multiple heads of attention, we simply take advantage of module broadcasting (Definition 6):

Compound 5 (Multi-headed attention)**.** For positive integers ℓ, d, h, d_Q, d_V and a choice of $\mathbf{mask}$, take four instances of the linear module:

$$\mathsf{Query} = \mathsf{Linear}(h * d_Q, d) \tag{B.40}$$

$$\mathsf{Key} = \mathsf{Linear}(h * d_Q, d) \tag{B.41}$$

$$\mathsf{Value} = \mathsf{Linear}(h * d_V, d) \tag{B.42}$$

$$\mathsf{Exit} = \mathsf{Linear}(d, h * d_V) \tag{B.43}$$

which by broadcasting we consider to have inputs of shape $\mathbb{R}^{\ell \times d}$. The multi-headed attention $\mathsf{MultiHeadAttention}$ module is then the composition:

$$\mathsf{MultiHeadAttention} = \mathsf{Exit} \circ \frac{1}{3} * \mathsf{FuncAttention}^{(h)} \circ (\mathsf{Query}, \mathsf{Key}, \mathsf{Value}) \tag{B.44}$$

where $\mathsf{FuncAttention}$ is broadcast over the heads dimension. Note that in Modula, we do this by creating dummy bond modules called `AddHeads` and `RemoveHeads` to reshape the tensors and create/remove the explicit head dimension.

As in the single-headed case, the scalar multiplication factor of $\frac{1}{3}$ ensures unit sensitivity.

B.7 Case study II: GPT

Let us now build an auto-regressive transformer similar to GPT-2 [43] or nanoGPT [44] in this framework. Fix positive integers ℓ, d, h, d_Q, d_V (usually h divides d and $d_Q = d_V = d/h$). In addition to Compound 5 from the earlier, consider the 2-layer MLP:

$$\mathsf{MLP} = \mathsf{Linear}(d, 4d) \circ \sqrt{2} * \mathsf{GELU} \circ \mathsf{Linear}(4d, d) \tag{B.45}$$

where we are using the scalar correction so that GELU has unit sensitivity, and using module broadcasting so that it can take inputs and outputs $\mathbb{R}^{\ell \times d}$. Fix a depth $L \geq 1$, and consider the following two modules, whose input and output spaces are $\mathbb{R}^{\ell \times d}$:

$$\mathsf{Block}_{\mathsf{MLP}} := \tfrac{2L-1}{2L} * \mathsf{Identity} + \tfrac{1}{2L} * \mathsf{MLP} \circ \mathsf{LayerNorm}_d \tag{B.46}$$

$$\mathsf{Block}_{\mathsf{Attn}} := \tfrac{2L-1}{2L} * \mathsf{Identity} + \tfrac{1}{2L} * \mathsf{MultiHeadAttention} \circ \mathsf{LayerNorm}_d \tag{B.47}$$

where $\mathsf{LayerNorm}_d$ refers to taking $\mathsf{LayerNorm}$ in the embedding dimension (i.e. the rows of matrices in $\mathbb{R}^{\ell \times d}$, as distinct from normalizing all $\ell \times d$ coordinates together). This can alternately be thought of as just taking the usual $\mathsf{LayerNorm}$ on $\mathbb{R}^d$ and broadcasting it to take inputs and outputs $\mathbb{R}^{\ell \times d}$.

Suppose that N is the number of tokens. For the initial module, take two embeddings of the N tokens and ℓ context positions

$$\mathsf{Embed}_{\mathrm{tok}} = \mathsf{Embed}(N, d), \qquad \mathsf{Embed}_{\mathrm{pos}} = \mathsf{Embed}(\ell, d) \tag{B.48}$$

and form the mass one, sensitivity one module

$$\mathsf{InputLayer} = \mathsf{tare}(\tfrac{1}{2} * \mathsf{Embed}_{\mathrm{tok}} + \tfrac{1}{2} * \mathsf{Embed}_{\mathrm{pos}}, 1). \tag{B.49}$$

The final module is just

$$\mathsf{OutputLayer} = \mathsf{Linear}(N, d) \circ \mathsf{LayerNorm}_d. \tag{B.50}$$

The depth $L \geq 1$, width d, total block mass $m > 0$ GPT module is thus

$$\mathsf{GPT} = \mathsf{OutputLayer} \circ \mathsf{tare}((\mathsf{Block}_{\mathsf{MLP}} \circ \mathsf{Block}_{\mathsf{Attn}})^L, m) \circ \mathsf{InputLayer}. \tag{B.51}$$

We suggest, as a default value, a total block mass of $m \sim 5$.

Appendix C More on Smoothness and Sharpness

C.1 Underlying every estimate: The Gauss-Newton decomposition

All our estimates of sharpness for compound modules, as well as the smoothness estimate Proposition 5 for loss functions, depend on an application of the chain rule to compute second derivatives which in the optimization context is sometimes called the Gauss-Newton decomposition.

Namely, if $\boldsymbol{f} : \mathbb{R}^{d_0} \to \mathbb{R}^{d_1}$ and $\boldsymbol{g} : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$, then the second derivative of their composite $\boldsymbol{h} = \boldsymbol{g} \circ \boldsymbol{f}$ is computed by

$$\boldsymbol{v} \diamond \nabla^2 \boldsymbol{h} \diamond \boldsymbol{w} = (\nabla \boldsymbol{f} \diamond \boldsymbol{v}) \diamond \nabla^2 \boldsymbol{g} \diamond (\nabla \boldsymbol{f} \diamond \boldsymbol{w}) + \nabla \boldsymbol{g} \diamond (\boldsymbol{v} \diamond \nabla^2 \boldsymbol{f} \diamond \boldsymbol{w}) \tag{C.1}$$

for any $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^{d_0}$, or for short

$$\nabla^2 \boldsymbol{h}(\cdot, \cdot) = \nabla^2 \boldsymbol{g}(\nabla \boldsymbol{f}(\cdot), \nabla \boldsymbol{f}(\cdot)) + \nabla \boldsymbol{g}(\nabla^2 \boldsymbol{f}(\cdot, \cdot)). \tag{C.2}$$

Indeed, this amounts to simply the following expression for partial derivatives:

$$\frac{\partial^2 \boldsymbol{h}}{\partial x_i \partial x_j} = \sum_{k,l} \frac{\partial^2 \boldsymbol{g}}{\partial y_k \partial y_l} \frac{\partial f_k}{\partial x_i} \frac{\partial f_l}{\partial x_j} + \sum_k \frac{\partial \boldsymbol{g}}{\partial y_k} \frac{\partial^2 f_k}{\partial x_i \partial x_j}. \tag{C.3}$$

C.2 Sharpness under composition and concatenation

Here, we state the two formulae for computing the sharpness of a composition and a concatenation of two modules. The proofs are given in Appendix E.

Proposition 8 (Sharpness under composition)**.** *Suppose that* M_2 *and* M_1 *are well-normed, composable modules that are respectively* $(\alpha_2, \beta_2, \gamma_2)$*-sharp and* $(\alpha_1, \beta_1, \gamma_1)$*-sharp. Under the shorthand that* $p_k \equiv \frac{\mathsf{M}_k.\mathsf{mass}}{\mathsf{M}_1.\mathsf{mass}+\mathsf{M}_2.\mathsf{mass}}$ *and* $\mu_k \equiv \mathsf{M}_k.\mathsf{sensitivity}$*, the composite* $\mathsf{M}_2 \circ \mathsf{M}_1$ *is* (α, β, γ)*-sharp for:*

$$\alpha = \tfrac{1}{\mu_2} p_1^2 \alpha_1 + p_2^2 \alpha_2 + \tfrac{2}{\mu_2} p_1 p_2 \beta_2 + \tfrac{1}{\mu_2^2} p_1^2 \gamma_2, \tag{C.4}$$

$$\beta = p_1 \beta_1 + \mu_1 p_2 \beta_2 + \tfrac{\mu_1}{\mu_2} p_1 \gamma_2, \tag{C.5}$$

$$\gamma = \mu_2 \gamma_1 + \mu_1^2 \gamma_2. \tag{C.6}$$

Proposition 9 (Sharpness under concatenation)**.** *Suppose that* M_1 *and* M_2 *are well-normed, concatenatable modules that are respectively* $(\alpha_1, \beta_1, \gamma_1)$*-sharp and* $(\alpha_2, \beta_2, \gamma_2)$*-sharp. Under the shorthand that* $p_k \equiv \frac{\mathsf{M}_k.\mathsf{mass}}{\mathsf{M}_1.\mathsf{mass}+\mathsf{M}_2.\mathsf{mass}}$ *and* $\mu_k \equiv \mathsf{M}_k.\mathsf{sensitivity}$*, the tuple* $(\mathsf{M}_1, \mathsf{M}_2)$ *is* (α, β, γ)*-sharp for:*

$$\alpha = p_1^2 \alpha_1 + p_2^2 \alpha_2, \tag{C.7}$$

$$\beta = p_1 \beta_1 + p_2 \beta_2, \tag{C.8}$$

$$\gamma = \gamma_1 + \gamma_2. \tag{C.9}$$

Taken together, Propositions 8 and 9 specify a recursive procedure for computing the sharpness of any compound module that is built from a set of well-normed modules of known sharpness.

Remark 1. These two sets of formulas are actually *associative*, as the reader may verify using their favorite computer algebra package. This means, for instance, that if $\mathsf{M}_1, \mathsf{M}_2, \mathsf{M}_3$ are successively composable, well-normed and each $(\alpha_k, \beta_k, \gamma_k)$-sharp, then the two sets of sharpness estimates coming from applying the above formulas for $\mathsf{M}_3 \circ (\mathsf{M}_2 \circ \mathsf{M}_1)$ and $(\mathsf{M}_3 \circ \mathsf{M}_2) \circ \mathsf{M}_1$ actually coincide.

C.3 Sharpness under module broadcasting

Suppose M is a well-normed module with inputs $\mathcal{X}$, outputs $\mathcal{Y}$ and weights $\mathcal{W}$, and suppose moreover that it is (α, β, γ)-sharp. The broadcast module $\mathsf{M}^{(h)}$ has the same weights, mass, sensitivity and norm, but takes $\mathcal{X}^h$ to $\mathcal{Y}^h$.

By Proposition 6, $\mathsf{M}^{(h)}$ is well-normed, as long as the norms on $\mathcal{X}^h$ and $\mathcal{Y}^h$ are taken to be

$$\|(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_h)\|_{\mathcal{X}^h} = S * (\|\boldsymbol{x}_1\|_{\mathcal{X}}^p + \ldots + \|\boldsymbol{x}_h\|_{\mathcal{X}}^p)^{1/p} \tag{C.10}$$

$$\|(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_h)\|_{\mathcal{Y}^h} = S * \left(\|\boldsymbol{y}_1\|_{\mathcal{Y}}^p + \ldots + \|\boldsymbol{y}_h\|_{\mathcal{Y}}^p\right)^{1/p} \tag{C.11}$$

for $1 \leq p \leq \infty$; unless M is a bond module (and thus weight-less), we must take $S = h^{-1/p}$, otherwise S can be any positive scalar.

A natural question is whether $\mathsf{M}^{(h)}$ is also sharp, and if so what its sharpness constants are, with respect to these norms. More or less the same proof as for Proposition 6 shows that the α and β bounds for sharpness are always true, with the same α, β. The γ bound is trickier however, and depends subtly on the chosen S, p. We highlight three cases where one can say something interesting.

Case 1. $p = \infty, S = 1$. For the L^∞ norm, we have that $\mathsf{M}^{(h)}$ is (α, β, γ)-sharp with the same α, β, γ by a more or less immediate proof.

Case 2. $p < \infty, S = 1$. For the "standard" L^p-norms, we have that $\mathsf{M}^{(h)}$ is (α, β, γ)-sharp with the same α, β, γ. The proof is direct, using the inequality

$$(x_1^p \widetilde{x}_1^p + \ldots + x_h^p \widetilde{x}_h^p)^{1/p} \leq (x_1^p + \ldots + x_h^p)^{1/p} (\widetilde{x}_1^p + \ldots + \widetilde{x}_h^p)^{1/p} \tag{C.12}$$

for any positive reals $x_1, \ldots, x_h, \tilde{x}_1, \ldots, \tilde{x}_h$; however this is *a very weak inequality* and so leads to very pessimistic sharpness estimates for large h.

Case 3. $p = 2, S = 1/\sqrt{h}$. This is the "RMS norm" case. As in Case 2, one could use a very weak inequality to obtain the pessimistic result that $\mathsf{M}^{(h)}$ is $(\alpha, \beta, \sqrt{h} * \gamma)$-sharp. However, one could also make the following observation: if h is large, and $x_1, \ldots, x_h$ are sampled from any normal distribution $N(\mu, \sigma^2)$, then

$$\left(\frac{1}{h}(x_1^4 + \ldots + x_h^4) \right)^{1/2} \approx \sqrt{3} \left(\frac{1}{h}(x_1^2 + \ldots + x_h^2) \right). \tag{C.13}$$

In particular, this justifies the statement that "for large h, the broadcast module $\mathsf{M}^{(h)}$ is approximately $(\alpha, \beta, \sqrt{3} * \gamma)$-sharp". While in actual deep learning contexts, the assumption that $x_1, \ldots, x_h$ are sampled from a normal distribution may not be valid, one should still expect the ratio between the two sides of Equation (C.13) to stay O(1) as $h \to \infty$, and so even if the "$\sqrt{3}$ rule" is insufficient, the effective sharpness of the broadcast module should not blow up as $h \to \infty$.

C.4 Smoothness estimates for common error measures

Suppose $\ell : \mathcal{Y} \times \mathcal{T} \to \mathbb{R}$ is an error measure. In Proposition 5, we showed that smoothness estimates on ℓ together with sharpness of a neural network imply smoothness of the corresponding average error loss function. The precise estimates are that ℓ is σ-Lipschitz and τ-smooth in the module output, in the sense that:

$$|\nabla_{\boldsymbol{y}} \ell(\boldsymbol{y}, \boldsymbol{t}) \diamond \Delta \boldsymbol{y}| \leq \sigma \, \|\Delta \boldsymbol{y}\|_{\mathcal{Y}} \qquad \text{for all } \Delta \boldsymbol{y} \in \mathcal{Y} \text{ and } \boldsymbol{t} \in \mathcal{T}; \tag{C.14}$$

$$|\Delta \boldsymbol{y} \diamond \nabla^2_{\boldsymbol{y}\boldsymbol{y}} \ell(\boldsymbol{y}, \boldsymbol{t}) \diamond \Delta \widetilde{\boldsymbol{y}}| \leq \tau \, \|\Delta \boldsymbol{y}\|_{\mathcal{Y}} \, \|\Delta \widetilde{\boldsymbol{y}}\|_{\mathcal{Y}} \qquad \text{for all } \Delta \boldsymbol{y}, \Delta \widetilde{\boldsymbol{y}} \in \mathcal{Y} \text{ and } \boldsymbol{t} \in \mathcal{T}. \tag{C.15}$$

We now present estimates on σ and τ for square and cross-entropy error. Both estimates will be in terms of *the value of the average loss function $\mathcal{L}$ itself*, rather than being truly global over the entire output space $\mathcal{Y}$. Thus, to apply them to real learning problems, one should *measure* the average loss $\mathcal{L}$ at initialization, and use this for estimates for σ and τ; we are implicitly making the assumption that under gradient descent the loss decreases.

Square error

Consider square error for a d-class classification problem. Thus, $\mathcal{Y} = \mathbb{R}^d$ and $\mathcal{T} = \{1, \ldots, d\}$. Consider the RMS norm on $\mathcal{Y}$, and define the error function

$$\ell(\boldsymbol{y}, t) = \frac{1}{2d} \left(y_1^2 + \ldots + (y_t - \sqrt{d})^2 + \ldots + y_d^2 \right) \qquad \text{for } \boldsymbol{y}, t \in \mathcal{Y} \times \mathcal{T}. \tag{C.16}$$

(the slightly non-standard scalings are due to the choice of RMS norm on $\mathcal{Y}$). The first and second partial derivatives of ℓ are given by

$$\frac{\partial \ell}{\partial y_i}(\boldsymbol{y}, t) = \frac{1}{d}(y_i - \delta_{it}\sqrt{d}), \qquad \frac{\partial^2 \ell}{\partial y_i \partial y_j}(\boldsymbol{y}, t) = \frac{1}{d}\delta_{ij} \tag{C.17}$$

The desired constants σ, τ can then be computed as maxima:

$$\sigma = \max_{\|\boldsymbol{z}\|_{\mathsf{RMS}}=1} \sum_i \frac{\partial \ell}{\partial y_i} z_i, \qquad \tau = \max_{\|\boldsymbol{z}\|_{\mathsf{RMS}}=1} \sum_{i,j} \frac{\partial^2 \ell}{\partial y_i \partial y_j} z_i z_j \tag{C.18}$$

which from the above formulas amounts exactly to

$$\sigma = \sqrt{\ell(\boldsymbol{y}, t)}, \qquad \tau = 1. \tag{C.19}$$

To translate this into a bound for the average loss function $\mathcal{L}$, note that square root is a *concave* function. Thus if we have outputs $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_B$ with true classes $t_1, \ldots, t_B$, Jensen's inequality yields

$$\frac{1}{B} \sum \sqrt{\ell(\boldsymbol{y}_b, t_b)} \leq \sqrt{\tfrac{1}{B} \sum \ell(\boldsymbol{y}_b, t_b)} = \sqrt{\mathcal{L}} \tag{C.20}$$

allowing us to use $\sigma = \sqrt{\mathcal{L}}$ as our estimate for Proposition 5.

Cross-entropy error

Consider cross-entropy error for a d-class classification problem. Thus, $\mathcal{Y} = \mathbb{R}^d$ and $\mathcal{T} = 1, \ldots, d$. For $\boldsymbol{y} \in \mathbb{R}^d$ and $t \in \mathcal{T}$, write

$$p_t(\boldsymbol{y}) = \frac{e^{y_t}}{\Sigma_j e^{y_j}} \tag{C.21}$$

and consider the error function

$$\ell(\boldsymbol{y}, t) = -\log(p_t(\boldsymbol{y})). \tag{C.22}$$

The first and second partial derivatives of ℓ are given by

$$\frac{\partial \ell}{\partial y_i}(\boldsymbol{y}, t) = p_i - \delta_{it}, \qquad \frac{\partial^2 \ell}{\partial y_i y_j}(\boldsymbol{y}, t) = \delta_{ij} p_i - p_i p_j. \tag{C.23}$$

Consider again the RMS norm on $\mathcal{Y}$. An estimate on σ can thus be computed as

$$\max_{\|\boldsymbol{z}\|_{\mathsf{RMS}}=1} \sum_i \frac{\partial \ell}{\partial y_i} z_i = \sqrt{d} \left(\sum_i (p_i - \delta_{it}) \right)^{1/2} \leq \sqrt{d} * \sqrt{\ell} \tag{C.24}$$

using the basic fact that if $p_1, \ldots, p_d$ are non-negative numbers that sum to 1, then

$$p_1^2 + \ldots + (p_t - 1)^2 + \ldots + p_d^2 \leq -\log(p_t). \tag{C.25}$$

(Indeed, for fixed p_t, the left hand side is maximized at $p_1 = 1 - p_t$ and all other p_i = 0; one then easily checked that $2(p-1)^2 \leq -\log(p)$ for all $0 < p \leq 1$.)

A similar concavity argument to the square error case then enables us to use $\sigma = \sqrt{d} * \sqrt{\mathcal{L}}$ as the first derivative bound for average cross-entropy loss.

The second derivative bound depends on more subtle information geometry. Indeed, τ can be computed to be

$$\tau = d * \lambda \tag{C.26}$$

where λ is the largest eigenvalue of the matrix $\operatorname{diag}(\boldsymbol{p}) - \boldsymbol{p}\boldsymbol{p}^T$. It is possible for this eigenvalue to be quite large (for instance, if $p_1 = p_2 = 1/2$ and all other $p_i = 0$, then $\lambda = 1/2$). However, the average eigenvalue is

$$\frac{1}{d}\left(1 - \sum p_i^2\right) \leq \frac{d-1}{d^2} < \frac{1}{d}. \tag{C.27}$$

If we presumed that, in the course of a gradient descent optimizing the weights of a module M, the output perturbations $\nabla \mathsf{M} \diamond \Delta \boldsymbol{w}$ are only generically aligned with the eigenvectors of $\operatorname{diag}(\boldsymbol{p}) - \boldsymbol{p}\boldsymbol{p}^T$, then we could use the "effective" smoothness bound $\tau = 1$.

Perhaps this is a dubious assumption however. A more conservative, but perhaps still dubious, assumption comes from assuming that the logits $\boldsymbol{y}$ have roughly $N(0, 1)$ entries—at least this could be more or less true at initialization. In this case, the largest eigenvalue λ is with high probability bounded as

$$\lambda \leq 1/\sqrt{d} \tag{C.28}$$

justifying "approximate" smoothness bound of $\tau = \sqrt{d}$.

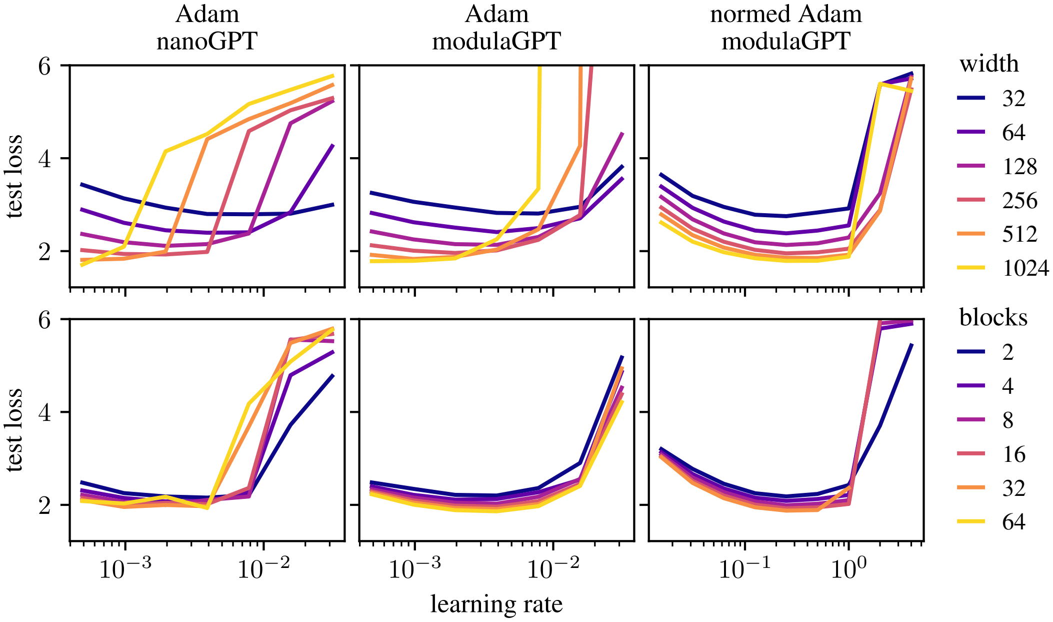

Figure 5: Comparing to a standard transformer implementation. Since we used our own well-normed GPT implementation for the experiments in this paper (here referred to as modulaGPT) we wanted to check its performance was on par with a standard nanoGPT implementation. These plots show learning rate sweeps for varying width and depth for Adam on nanoGPT, as well as Adam and normed Adam on modulaGPT. Even without normed updates, the architectural changes and orthogonal initialization used in Modula seem to already improve transfer compared to nanoGPT.

Appendix D Experimental Details

D.1 Datasets

All experiments with ResMLP and ResNet [45] are done with the CIFAR-10 [46] image dataset with standard train and test splits. For data augmentation on the training set, we use random crop, random horizontal flip and PyTorch AutoAugment.

For the GPT [43] transformer experiments, we compared three different datasets:

(a) The Shakespeare corpus, using character-level tokens [47];

(b) The TinyStories database [48] using sub-word level tokenization;

(c) OpenWebText using sub-word level tokenization [49].

No data augmentation was used on the language data. We used data splitting code from [44].

D.2 Architectures

Full details of the ResMLP, ResNet and GPT architectures we used are detailed in Appendices B.5 and Appendix B.7. In every experiment, we used:

(a) cross-entropy loss with no weight decay;

(b) block depth $B = 2$ for ResMLP and ResNet;

(c) kernel size $K = 3$ for ResNet;

(d) $h = 8$ heads for GPT, with query and value dimensions $d_Q = d_V = d/h$ where d is the embedding dimension (width);

(e) context length 128 for GPT, except for the experiment in Appendix D.7.

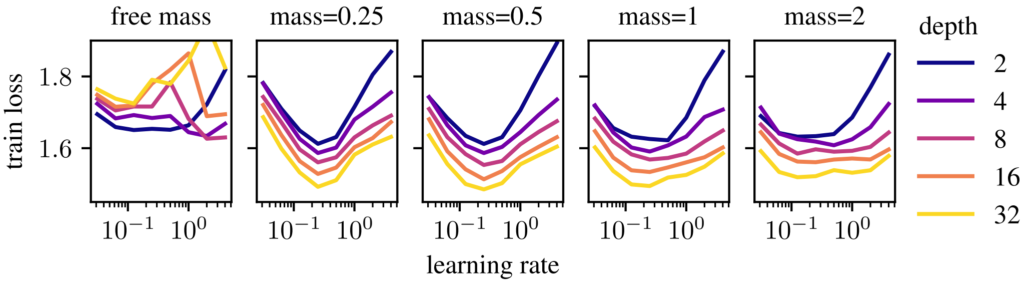

Figure 6: Comparing mass allocation strategies. We train a ResMLP with width 64 and 2 layers per block on CIFAR-10. In the first sub-plot titled "free mass", we set every atomic module to have unit mass, so that as depth is scaled the masses of the input and output layer become insignificant relative to the total mass of the hidden layers. In the other four subplots, we tare the total mass of the hidden layers to the value indicated in the subplot title. As can be seen, the taring strategy seems to work much better than the free mass strategy. So, at least in this experiment, it is good to keep a constant fraction of learning in the input and output layers even as depth is scaled.

D.3 Hardware

All experiments were run on NVIDIA GPUs using `float32`-precision. We used a combination of `TITAN-RTX`, `RTX-3090`, `V100`, `Ada6000`, and `H100` devices. Each data point in the experiments takes up to 5 hours, depending on the computing device used. We ran over 1000 training runs in total.

D.4 Comparing to standard nanoGPT architecture

Our implementation of GPT in `Modula` has certain differences from off-the-shelf architectures such as nanoGPT [44]. We would summarize the overall changes to transformer architecture and training the following three points:

(I) the mathematical *architecture has slightly different coefficients*;

(II) we initialize weight matrices to be *orthogonal rather than Gaussian*;

(III) we train using *normalized weight updates*.

The architectural choices we made were entirely informed by the desire for the network to be well-normed and have unit sensitivity: in particular this means that the network enjoys favorable signal propagation properties. In the language of modules, these architectural changes can be summarized as:

(a) Each residual block in our architecture is of the form

$$\tfrac{2L-1}{2L} * \mathsf{Identity} + \tfrac{1}{2L} * \mathsf{Block} \tag{D.1}$$

where $\mathsf{Block} = \mathsf{Block}_{\mathsf{MLP}}$ or $\mathsf{Block}_{\mathsf{Attn}}$, compared to $\mathsf{Identity} + \frac{1}{\sqrt{L}}\mathsf{Block}$ suggested for nanoGPT;

(b) We use a scaled dot product attention with $\frac{1}{d_Q}$ scaling, rather than $\frac{1}{\sqrt{d_Q}}$;

(c) The forward function of our Linear and Embed modules (see Appendix B.1) includes scale factors $\sqrt{d_{\text{out}}/d_{\text{in}}}$ and $\sqrt{d}$ respectively.

(d) We use several additional scalar multiplications to keep the network of unit sensitivity:

- Each Attention module (B.44) has a scalar factor of $\frac{1}{3}$;
- Each MLP module (B.45) has a scalar factor of $\sqrt{2}$;
- The token and position embeddings (B.49) have a scalar factor of $\frac{1}{2}$.

In Figure 5, we ran a comparison of the performance of the standard (unnormed) Adam optimizer trained on OpenWebText with:

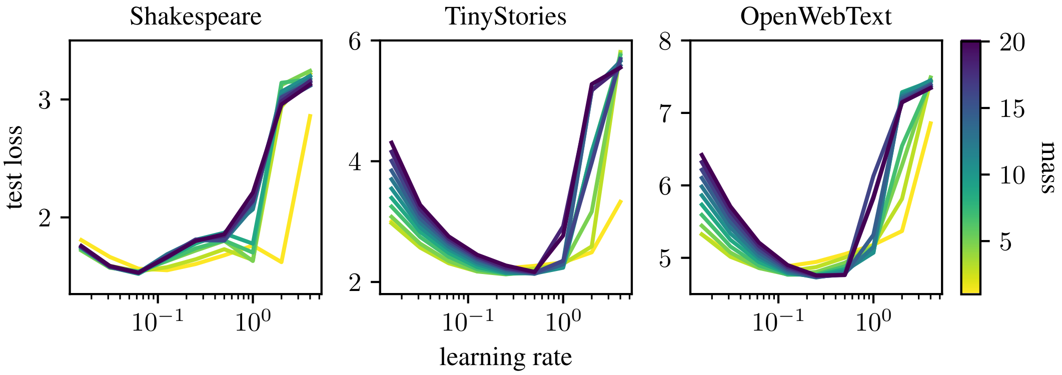

Figure 7: Mass and learning rate sweeps across datasets of increasing difficulty. A small GPT architecture of width 128 and 3 transformer blocks was trained on the Shakespeare, TinyStores and OpenWebText datasets. We varied the learning rate as well as the total mass of the blocks. Optimal mass and learning rate seem to transfer reasonably well from TinyStories to OpenWebText, and less well from the much smaller Shakespeare dataset.

1. the nanoGPT architecture with Gaussian initialization;
2. our implementation of GPT with orthogonal initialization.

We found that even without using the normed optimizer, our implementation with orthogonal initialization *transferred learning rate better*. We suggest that even the base Adam optimizer benefits from the above architectural changes.

D.5 Full sweeps

In Figures 9 to 12, at the end of this Appendix, we report on full learning rate sweep experiments, across width and depth, for GPT on OpenWebText and TinyStories, and ResMLP, ResNet on CIFAR-10.

We consistently find that the normed Adam optimizer matches or outperforms unnormed Adam in both test and training loss, all the while exhibiting significantly better transfer across width. The difference in depth transfer is less stark, however we posit that, in part, unnormed Adam is already benefiting from architectural changes we made to improve depth scaling.

Notice too that normed SGD consistently significantly outperforms ordinary SGD, often coming close to or matching the performance of Adam. We would like to highlight this, since SGD has a significantly lower memory requirement than Adam, and does not require any tuning of β_2.

D.6 Mass allocation

A novel feature of our normed optimization framework is the need to choose a *mass* parameter for each atomic module. In the context of networks of the form

$$\mathsf{Network} = \mathsf{OutputLayer} \circ \mathsf{HiddenLayers} \circ \mathsf{InputLayer} \tag{D.2}$$

where $\mathsf{HiddenLayers} = \mathsf{Block}^L$. We typically do this by assuming that $\mathsf{InputLayer}, \mathsf{OutputLayer}$ have mass 1, and by hand resetting the mass of $\mathsf{HiddenLayers}$ to be a fixed total mass $m > 0$, by calling $\mathsf{tare}(\mathsf{HiddenLayers}, m)$.

In this Appendix, we explore some different aspects the choice of m.

First, we tested whether or not calling tare is necessary in the first place. Not using tare would leave the “free mass” of $\mathsf{HiddenLayers.mass} = L * \mathsf{Block.mass}$; accordingly as L grows large, the feature learning allotment (see Proposition 3) for $\mathsf{InputLayer}$ and $\mathsf{OutputLayer}$ would grow smaller. Indeed, as the reader can see in Figure 6, this “free mass” arrangement for a ResMLP network on CIFAR-10, allowing the mass of $\mathsf{HiddenLayers}$ to grow with L is very undesirable, and for good learning rate transfer with depth we must fix a mass.

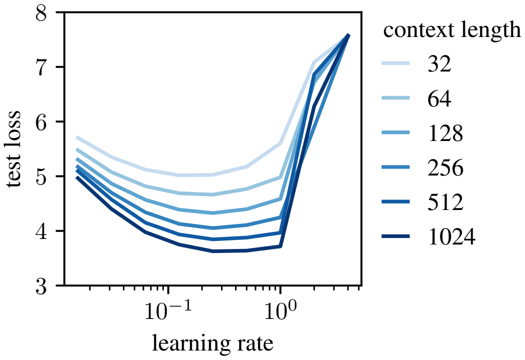

Figure 8: Context length transfer. We trained GPTs of various context lengths using normed Adam. As can be seen, learning rate transferred quite well across context length.

The mass m is thus left as a tunable parameter. We then tested the transferability of mass tuning. Specifically, we wanted to know:

1. whether one can tune m on a network of small width/depth, and expect that same m to be close to optimal on a larger network;
2. whether learning rate transfer across width/depth is itself dependent on selecting a good mass m;
3. how sensitive the tuning for m is: if there is a broad range of acceptable masses, or certain precise values lead to big improvements in train or test loss.

Figures 3 and 6 answer Question 1 above in the affirmative, in the context of ResMLP on CIFAR-10 and GPT on OpenWebText. Moreover, in the context of ResMLP on CIFAR-10, they give an answer of Question 2 and Question 3: learning rate transfer occurs at a range of values of m.

Figure 7 address Question 3 in the context of transformers, on three different datasets. Across all three datasets, a mass in the region $m \sim 5$ to 10 is reasonable.

D.7 Context length

Additionally, we also tested the dependence of the optimal learning rate for GPT training on OpenWebText on *the context length*; the results are in Figure 8 Interestingly, we report good transfer of the optimal learning rate from small contexts to long contexts.

D.8 Full sweep results

The next four pages of the Appendix list results of our full learning rate sweeps over width/depth for GPT on OpenWebText and TinyStories, and ResMLP, ResNet on CIFAR-10.

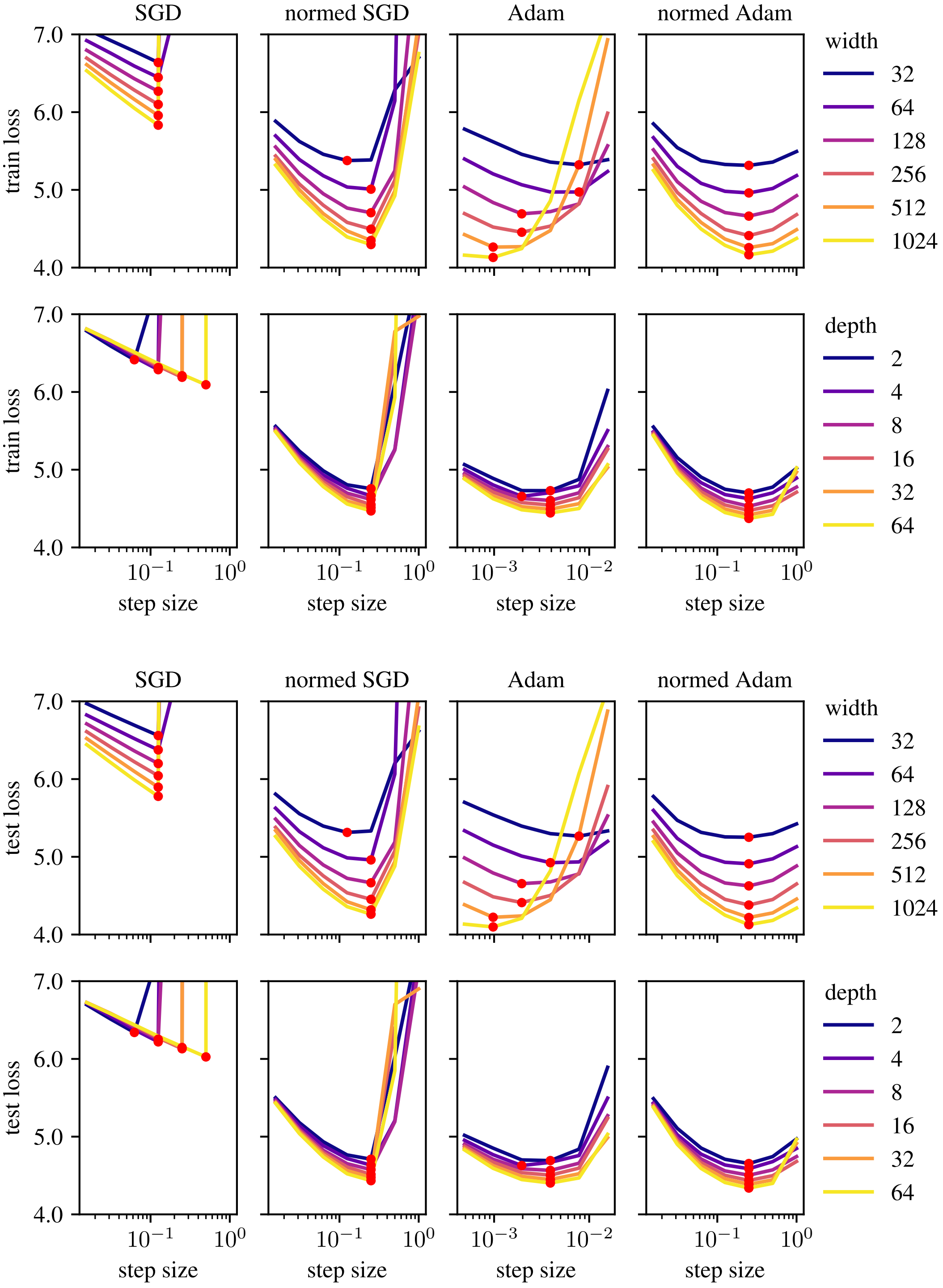

Figure 9: Learning rate transfer for GPT on OpenWebText. Training is done for 10k steps, at batch size 128, with SGD, Adam, and their normed versions. The total block mass for normed SGD/Adam is $m = 5$. Width scaling experiments are done at fixed depth 3, and depth scaling experiments are done at fixed width 128.

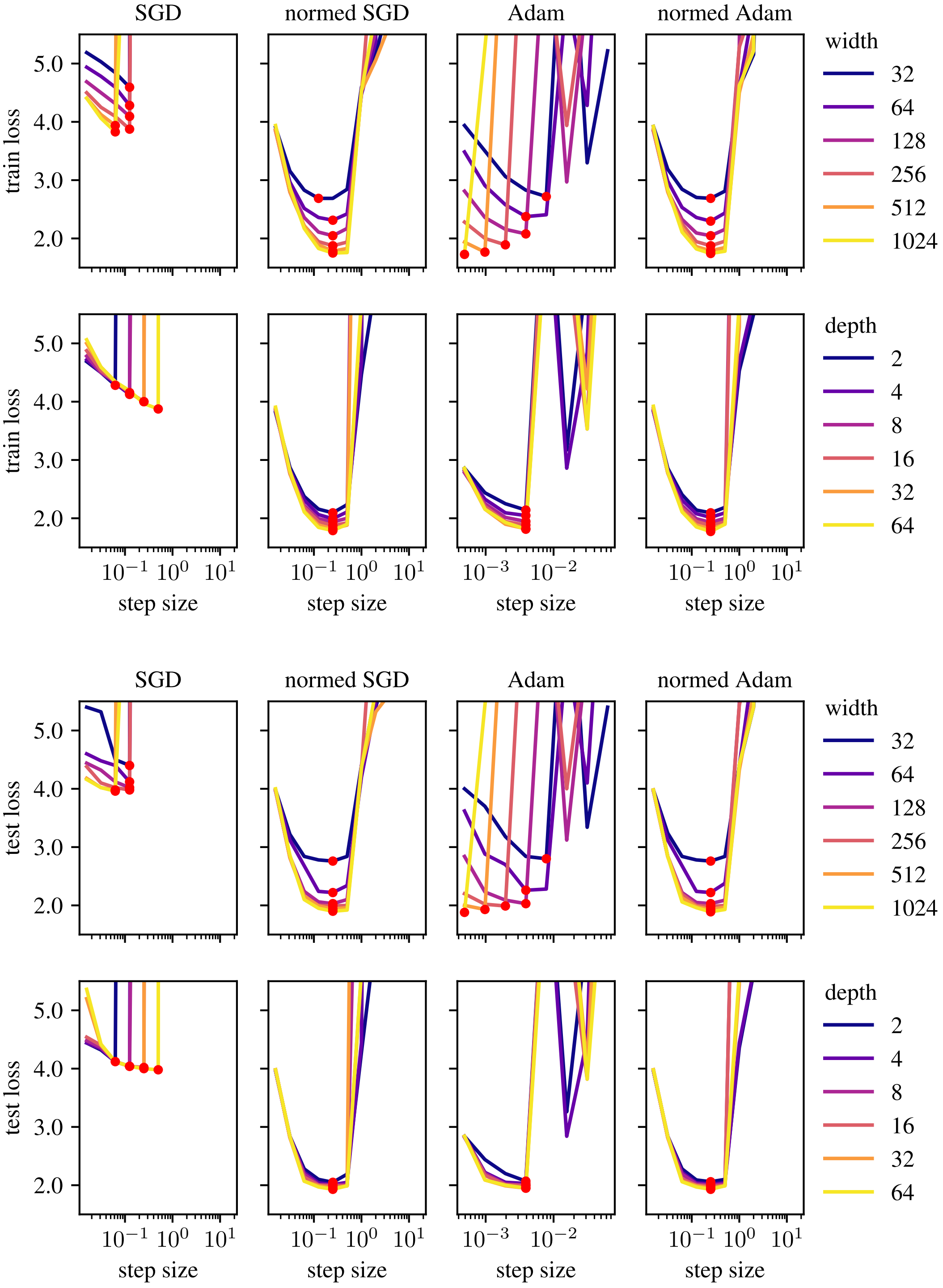

Figure 10: Learning rate transfer for GPT on TinyStories. Training is done for 10k steps, at batch size 128, with SGD, Adam, and their normed versions. The total block mass for normed SGD/Adam is $m = 5$. Width scaling experiments are done at fixed depth 3, and depth scaling experiments are done at fixed width 128.

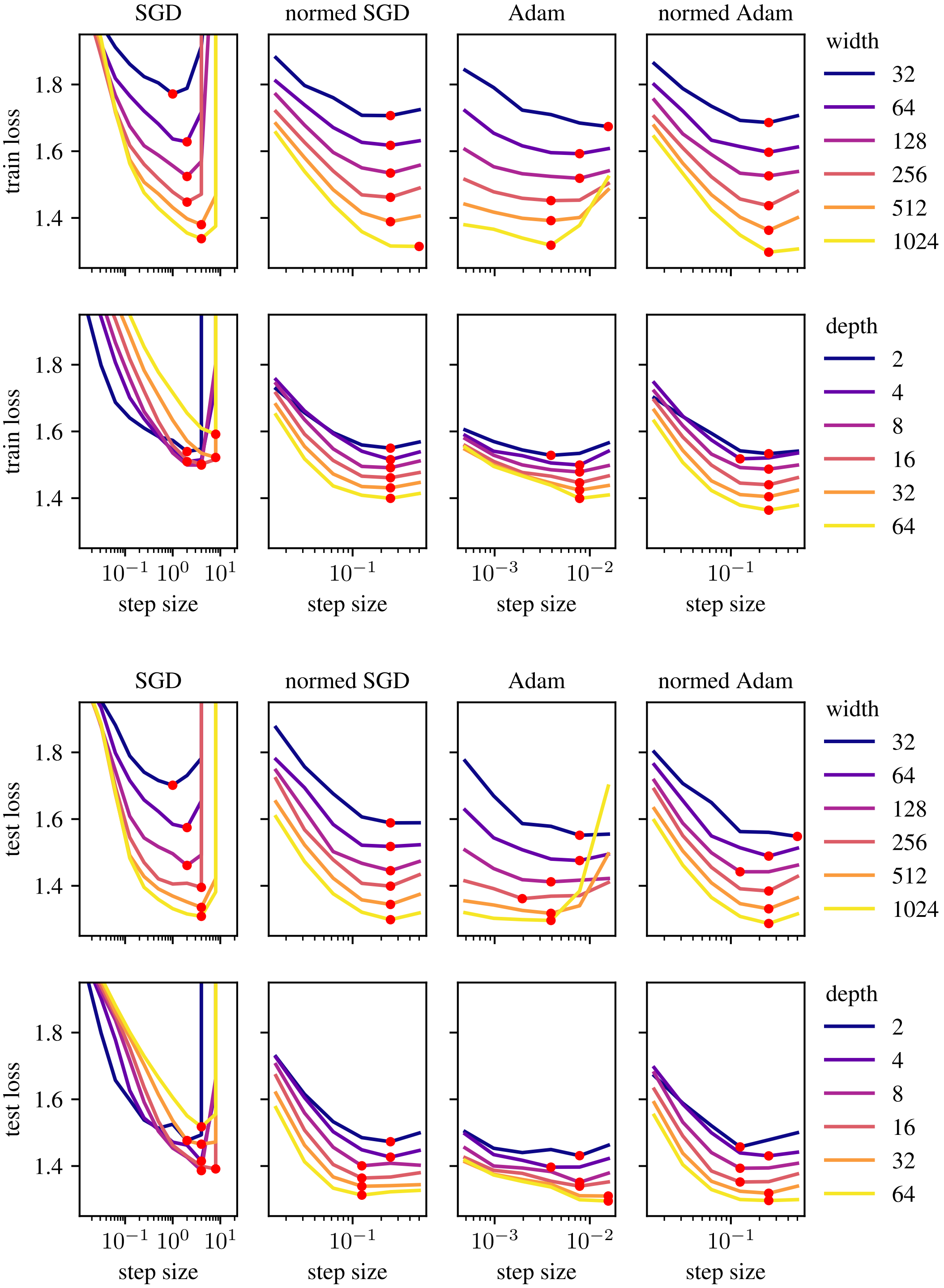

Figure 11: Learning rate transfer for ResMLP. ResMLP architectures on CIFAR-10 are trained for 10k steps, at batch size 128, with SGD, Adam, and their normed versions. The total block mass for normed SGD/Adam is $m = 1$. Width scaling experiments are done at fixed depth 3, and depth scaling experiments are done at fixed width 128.

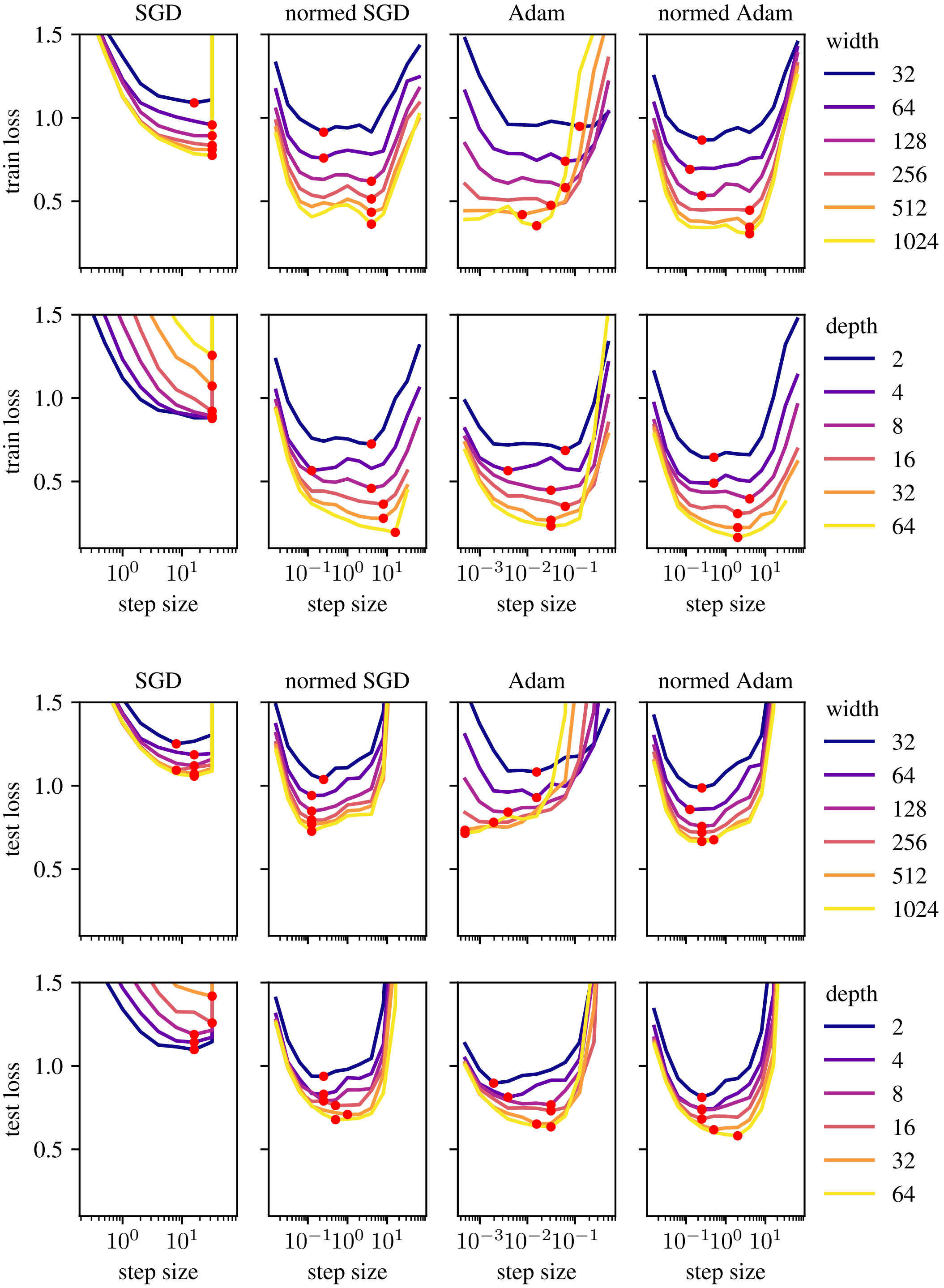

Figure 12: Learning rate transfer for ResNet. ResNet architectures on CIFAR-10 are trained for 10k steps, at batch size 128, with SGD, Adam, and their normed versions. The total block mass for normed SGD/Adam is $m = 20$. Width scaling experiments are done at fixed depth 3, and depth scaling experiments are done at fixed width 128.

Appendix E Proofs

Proposition 3: Feature learning is apportioned by mass

To prove Proposition 3, it suffices to induct on the construction of a compound module M by composition and concatenation, with the atomic modules (where the inequality is just part of well-normed-ness) as the base case.

Indeed, suppose either $\mathsf{M} = \mathsf{M}_2 \circ \mathsf{M}_1$ or $\mathsf{M} = (\mathsf{M}_1, \mathsf{M}_2)$. Suppose $\boldsymbol{w}_k$ is a weight for one of the atomic modules of M, and write m for the mass of this atomic module. Then $\boldsymbol{w}_k$ is must be a weight of either M_1 or M_2; the inductive assumption is that

$$\|\nabla_{\boldsymbol{w}_k} \mathsf{M}_i \diamond \Delta \boldsymbol{w}_k\| \leq \frac{m}{\mathsf{M}_i.\mathsf{mass}} * \|\Delta \boldsymbol{w}\|_{\mathsf{M}_i} \tag{E.1}$$

where $i = 1$ or 2 accordingly.

Case 1. $\mathsf{M} = \mathsf{M}_2 \circ \mathsf{M}_1$ and $\boldsymbol{w}_k$ is a weight of M_1. From the chain rule we then must have:

$$\|\nabla_{\boldsymbol{w}_k} \mathsf{M} \diamond \Delta \boldsymbol{w}_k\| = \|\nabla_{\boldsymbol{x}} \mathsf{M}_2 \diamond \nabla_{\boldsymbol{w}_k} \mathsf{M}_1 \diamond \Delta \boldsymbol{w}_k\| \tag{E.2}$$

$$\leq \mathsf{M}_2.\mathsf{sensitivity} * \|\nabla_{\boldsymbol{w}_k} \mathsf{M}_1 \diamond \Delta \boldsymbol{w}_k\| \quad \text{by well-normed-ness} \tag{E.3}$$

$$\leq \mathsf{M}_2.\mathsf{sensitivity} * \frac{m}{\mathsf{M}_1.\mathsf{mass}} * \|\Delta \boldsymbol{w}\|_{\mathsf{M}_1} \quad \text{by assumption} \tag{E.4}$$

$$\leq \frac{m}{\mathsf{M}.\mathsf{mass}} \|\Delta \boldsymbol{w}\|_{\mathsf{M}} \tag{E.5}$$

where the last line is by the definition of the norm of module composition.

Case 2. $\mathsf{M}_2 \circ \mathsf{M}_1$ and $\boldsymbol{w}_k$ is a weight of M_2. The chain rule is not needed in this case, and we proceed straight from the inductive assumption:

$$\|\nabla_{\boldsymbol{w}_k} \mathsf{M} \diamond \Delta \boldsymbol{w}_k\| = \|\nabla_{\boldsymbol{w}_k} \mathsf{M}_2 \diamond \Delta \boldsymbol{w}_k\| \tag{E.6}$$

$$\leq \frac{m}{\mathsf{M}_2.\mathsf{mass}} * \|\Delta \boldsymbol{w}\|_{\mathsf{M}_2} \tag{E.7}$$

$$\leq \frac{m}{\mathsf{M}.\mathsf{mass}} \|\Delta \boldsymbol{w}\|_{\mathsf{M}}. \tag{E.8}$$

Case 3. $\mathsf{M} = (\mathsf{M}_1, \mathsf{M}_2)$. Given the symmetric roles of $\mathsf{M}_1, \mathsf{M}_2$, without loss of generality assume $\boldsymbol{w}_k$ is a weight of M_1. Then,

$$\|\nabla_{\boldsymbol{w}_k} \mathsf{M} \diamond \Delta \boldsymbol{w}_k\| = \|\nabla_{\boldsymbol{w}_k} \mathsf{M}_1 \diamond \Delta \boldsymbol{w}_k\| \tag{E.9}$$

$$\leq \frac{m}{\mathsf{M}_1.\mathsf{mass}} * \|\Delta \boldsymbol{w}\|_{\mathsf{M}_1} \tag{E.10}$$

$$\leq \frac{m}{\mathsf{M}.\mathsf{mass}} \|\Delta \boldsymbol{w}\|_{\mathsf{M}}. \tag{E.11}$$

This completes the proof.

Proposition 4: Sharpness of residual networks

Suppose M is a well-normed module of unit sensitivity on $(\mathcal{X}, \mathcal{X}, \mathcal{W})$ and is (α, β, γ)-sharp. Then, by Proposition 8 for any $L \geq 1$, the module $\frac{1}{L} * \mathsf{M}$ is well-normed, sensitivity $\frac{1}{L}$, and $(L\alpha, \beta, \frac{1}{L}\gamma)$-sharp.

The module $\frac{L-1}{L} * \mathsf{Identity}$ is also well-normed, sensitivity $\frac{L-1}{L}$, and $(0, 0, 0)$-sharp. In particular, the sum

$$\mathsf{M}_{res} = \tfrac{L-1}{L} * \mathsf{Identity} + \tfrac{1}{L} * \mathsf{M} \tag{E.12}$$

is well-normed, unit sensitivity, and $(L\alpha, \beta, \frac{1}{L}\gamma)$-sharp; it has the same mass as the original module.

We induct on the statement for $k = 1, 2, \ldots$ that M_{res}^k is $(\alpha_k, \beta_k, \gamma_k)$-sharp where

$$\alpha_k = \frac{L}{k}\alpha + \frac{2\,(1 + 2 + \ldots + (k-1))}{k^2}\beta + \frac{\left(1^2 + 2^2 + \ldots + (k-1)^2\right)}{Lk^2}\gamma \tag{E.13}$$

$$\beta_k = \beta + \frac{(1 + 2 + \ldots + (k-1))}{Lk}\gamma \tag{E.14}$$

$$\gamma_k = \frac{k}{L}\gamma. \tag{E.15}$$

The base case is clearly true, and given the statement for M_{res}^k, which has exactly k times the mass as M_{res}, we see that $\mathsf{M}_{res}^{k+1} = \mathsf{M}_{res} \circ \mathsf{M}_{res}^k$ is $(\alpha_{k+1}, \beta_{k+1}, \gamma_{k+1})$-sharp by applying Proposition 8 with $p_1 = \frac{k}{k+1}$ and $p_2 = \frac{1}{k+1}$, where

$$\alpha_{k+1} = \frac{k^2}{(k+1)^2}\alpha_k + \frac{1}{(k+1)^2}L\alpha + \frac{2k}{(k+1)^2}\beta + \frac{k^2}{L(k+1)^2}\gamma \tag{E.16}$$

$$\beta_{k+1} = \frac{k}{k+1}\beta_k + \frac{1}{k+1}\beta + \frac{k}{L(k+1)}\gamma \tag{E.17}$$

$$\gamma_{k+1} = \gamma_k + \frac{1}{L}\gamma. \tag{E.18}$$

which yields the induction.

Setting $k = L$, observe that $1 + 2 + \ldots + (L-1) = \frac{1}{2}L(L-1)$ and $1^2 + 2^2 + \ldots + (L-1)^2 = \frac{1}{6}L(L-1)(2L-1)$, giving

$$\alpha_L = \alpha + \frac{L-1}{L}\beta + \frac{L(L-1)(2L-1)}{6L^3}\gamma \leq \alpha + \beta + \tfrac{1}{3}\gamma \tag{E.19}$$

$$\beta_L = \beta + \frac{L-1}{2L}\gamma \leq \beta + \tfrac{1}{2}\gamma \tag{E.20}$$

$$\gamma_L = \gamma \tag{E.21}$$

which proves the result.

Proposition 5: Smoothness in the modular norm

To establish the first inequality, we start by applying the Gauss-Newton decomposition (C.1) of the Hessian, followed by the fact that the error ℓ is σ-Lipschitz and τ-smooth, followed by the well-normedness and (α, β, γ)-sharpness of the module M:

$$|\Delta \boldsymbol{w} \diamond \nabla^2_{\boldsymbol{w}\boldsymbol{w}}\mathcal{L} \diamond \Delta\widetilde{\boldsymbol{w}}| \tag{E.22}$$

$$= \left|\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\mathcal{D}}\left[\nabla_{\boldsymbol{y}}\ell \diamond \left(\Delta\boldsymbol{w} \diamond \nabla^2_{\boldsymbol{w}\boldsymbol{w}}\mathsf{M} \diamond \Delta\widetilde{\boldsymbol{w}}\right) + (\nabla_{\boldsymbol{w}}\mathsf{M} \diamond \Delta\boldsymbol{w}) \diamond \nabla^2_{\boldsymbol{y}\boldsymbol{y}}\ell \diamond (\nabla_{\boldsymbol{w}}\mathsf{M} \diamond \Delta\widetilde{\boldsymbol{w}})\right]\right| \tag{E.23}$$

$$\leq \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\mathcal{D}}\left[\sigma \left\|\Delta\boldsymbol{w} \diamond \nabla^2_{\boldsymbol{w}\boldsymbol{w}}\mathsf{M} \diamond \Delta\widetilde{\boldsymbol{w}}\right\|_{\mathcal{Y}} + \tau \left\|\nabla_{\boldsymbol{w}}\mathsf{M} \diamond \Delta\boldsymbol{w}\right\|_{\mathcal{Y}} \left\|\nabla_{\boldsymbol{w}}\mathsf{M} \diamond \Delta\widetilde{\boldsymbol{w}}\right\|_{\mathcal{Y}}\right] \tag{E.24}$$

$$\leq (\sigma\alpha + \tau) \left\|\Delta\boldsymbol{w}\right\|_{\mathsf{M}} \left\|\Delta\widetilde{\boldsymbol{w}}\right\|_{\mathsf{M}}. \tag{E.25}$$

The second inequality follows from the first via the fundamental theorem of calculus:

$$\left\|\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w} + \Delta\boldsymbol{w}) - \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})\right\|_{\mathsf{M}}^* = \max_{\|\Delta\widetilde{\boldsymbol{w}}\|_{\mathsf{M}}=1} \left|\left[\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w} + \Delta\boldsymbol{w}) - \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})\right] \diamond \Delta\widetilde{\boldsymbol{w}}\right| \tag{E.26}$$

$$\leq \max_{\|\Delta\widetilde{\boldsymbol{w}}\|_{\mathsf{M}}=1} \int_0^1 \left|\Delta\boldsymbol{w} \diamond \nabla^2_{\boldsymbol{w}\boldsymbol{w}}\mathcal{L}(\boldsymbol{w} + t\Delta\boldsymbol{w}) \diamond \Delta\widetilde{\boldsymbol{w}}\right| \mathrm{d}t \tag{E.27}$$

$$\leq \max_{\|\Delta\widetilde{\boldsymbol{w}}\|_{\mathsf{M}}=1} (\sigma\alpha + \tau) \left\|\Delta\boldsymbol{w}\right\|_{\mathsf{M}} \left\|\Delta\widetilde{\boldsymbol{w}}\right\|_{\mathsf{M}} \int_0^1 \mathrm{d}t \tag{E.28}$$

$$= (\sigma\alpha + \tau) \left\|\Delta\boldsymbol{w}\right\|_{\mathsf{M}}. \tag{E.29}$$

The third inequality follows from the second by again applying the fundamental theorem of calculus, followed by the Cauchy-Schwarz inequality:

$$\left|\mathcal{L}(\boldsymbol{w} + \Delta\boldsymbol{w}) - \left[\mathcal{L}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}) \diamond \Delta\boldsymbol{w}\right]\right| \tag{E.30}$$

$$= \left|\int_0^1 \left[\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w} + t\Delta\boldsymbol{w}) - \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})\right] \diamond \Delta\boldsymbol{w} \, \mathrm{d}t\right| \tag{E.31}$$

$$\leq \int_0^1 \left\|\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w} + t\Delta\boldsymbol{w}) - \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})\right\|_{\mathsf{M}}^* \left\|\Delta\boldsymbol{w}\right\|_{\mathsf{M}} \mathrm{d}t \tag{E.32}$$

$$\leq (\sigma\alpha + \tau) \left\|\Delta\boldsymbol{w}\right\|_{\mathsf{M}}^2 \int_0^1 t \, \mathrm{d}t \tag{E.33}$$

$$= \tfrac{1}{2} (\sigma\alpha + \tau) \left\|\Delta\boldsymbol{w}\right\|_{\mathsf{M}}^2. \tag{E.34}$$

This completes the proof.

Proposition 6: Broadcast modules are well-normed

Suppose M is a module with inputs $\mathcal{X}$, outputs $\mathcal{Y}$ and weights $\mathcal{W}$, broadcast to take $\mathcal{X}^h$ to $\mathcal{Y}^h$. We take norms on these spaces to be

$$\|(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_h)\|_{\mathcal{X}^h} = S * (\|\boldsymbol{x}_1\|_{\mathcal{X}}^p + \ldots + \|\boldsymbol{x}_h\|_{\mathcal{X}}^p)^{1/p} \tag{E.35}$$

$$\|(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_h)\|_{\mathcal{Y}^h} = S * \left(\|\boldsymbol{y}_1\|_{\mathcal{Y}}^p + \ldots + \|\boldsymbol{y}_h\|_{\mathcal{Y}}^p\right)^{1/p} \tag{E.36}$$

where $S = h^{-1/p}$ unless M is a bond module. Write $\mu = \mathsf{M.sensitivity}$. Then, for perturbations in the weight direction, which only occur if M is not a bond module:

$$\begin{aligned}
\|\nabla_{\boldsymbol{w}} \mathsf{M}(\boldsymbol{w}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_h) \diamond \Delta \boldsymbol{w}\|_{\mathcal{Y}^h} &= \left(\frac{1}{h} \sum_j \|\nabla_{\boldsymbol{w}} \mathsf{M}(\boldsymbol{w}, \boldsymbol{x}_j) \diamond \Delta \boldsymbol{w}\|_{\mathcal{Y}}^p\right)^{1/p} && \text{(E.37)} \\
&\leq \|\Delta \boldsymbol{w}\|_{\mathsf{M}} \quad \text{applying well-normed-ness.} && \text{(E.38)}
\end{aligned}$$

For perturbations in the input direction, we have:

$$\begin{aligned}
\|\nabla_{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_h} \mathsf{M} \diamond (\Delta \boldsymbol{x}_1, \ldots, \Delta \boldsymbol{x}_h)\|_{\mathcal{Y}^h} &= S * \left(\sum_j \|\nabla_{\boldsymbol{x}_j} \mathsf{M} \diamond \Delta \boldsymbol{x}_j\|_{\mathcal{Y}}^p\right)^{1/p} && \text{(E.39)} \\
&\leq S * \left(\sum_j \mu^p \|\Delta \boldsymbol{x}_j\|_{\mathcal{Y}}^p\right)^{1/p} && \text{(E.40)} \\
&= \mu * \|(\Delta \boldsymbol{x}_1, \ldots, \Delta \boldsymbol{x}_h)\|_{\mathcal{Y}^h} && \text{(E.41)}
\end{aligned}$$

which proves the proposition.

Proposition 7: Sensitivity of attention

We prove that the functional attention module $\mathsf{FuncAttention}$ of Bond 9 is well-normed and of unit sensitivity.

Recall we use the following norms on the inputs $\mathcal{X} = \mathbb{R}^{\ell \times d_Q} \times \mathbb{R}^{\ell \times d_Q} \times \mathbb{R}^{\ell \times d_V}$ and outputs $\mathcal{Y} = \mathbb{R}^{\ell \times d_V}$:

$$\|(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v})\|_{\mathcal{X}} = \|\boldsymbol{q}\|_{\infty\mathsf{RMS}} + \|\boldsymbol{k}\|_{\infty\mathsf{RMS}} + \|\boldsymbol{v}\|_{\infty\mathsf{RMS}}, \quad \|\boldsymbol{y}\|_{\mathcal{Y}} = \|\boldsymbol{y}\|_{\infty\mathsf{RMS}}. \tag{E.42}$$

We will also make use of the L^∞-operator norm for $\ell \times \ell$ matrices, which we write as

$$\|\boldsymbol{B}\|_{\infty-\mathsf{op}} = \max_{i=1,\ldots,\ell} \left(\sum_{j=1}^{\ell} |\boldsymbol{B}_{ij}|\right); \tag{E.43}$$

observe that for $\boldsymbol{B} \in \mathbb{R}^{\ell \times \ell}$ and $\boldsymbol{x} \in \mathbb{R}^{\ell \times d}$ we have

$$\|\boldsymbol{B}\,\boldsymbol{x}\|_{\infty\mathsf{RMS}} \leq \|\boldsymbol{B}\|_{\infty-\mathsf{op}} \|\boldsymbol{x}\|_{\infty\mathsf{RMS}}. \tag{E.44}$$

Writing $F = \mathsf{FuncAttention.forward}$ for short, recall that

$$F(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}) = \operatorname{softmax}(\tfrac{1}{d_Q} \boldsymbol{q}\boldsymbol{k}^T + \boldsymbol{M})\boldsymbol{v} \tag{E.45}$$

where $\boldsymbol{M}$ is the mask (our proof will apply equally for the standard causal mask and also the non-causal $\boldsymbol{M} \equiv 0$).

We will prove that at any $(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v})$ satisfying $\|\boldsymbol{q}\|_{\infty\mathsf{RMS}}, \|\boldsymbol{k}\|_{\infty\mathsf{RMS}}, \|\boldsymbol{v}\|_{\infty\mathsf{RMS}} \leq 1$, for any $(\Delta \boldsymbol{q}, \Delta \boldsymbol{k}, \Delta \boldsymbol{v})$ we have

$$\|\nabla F(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}) \diamond (\Delta \boldsymbol{q}, \Delta \boldsymbol{k}, \Delta \boldsymbol{v})\|_{\mathcal{Y}} \leq \|(\Delta \boldsymbol{q}, \Delta \boldsymbol{k}, \Delta \boldsymbol{v})\|_{\mathcal{X}}. \tag{E.46}$$

For short, write $\boldsymbol{A} = \operatorname{softmax}(\frac{1}{d_Q} \boldsymbol{q}\boldsymbol{k}^T + \boldsymbol{M})$ for the attention matrix and its derivative as

$$\Delta \boldsymbol{A} = \nabla_{(\boldsymbol{q}, \boldsymbol{k})} \operatorname{softmax}(\tfrac{1}{d_Q} \boldsymbol{q}\boldsymbol{k}^T + \boldsymbol{M}) \diamond (\Delta \boldsymbol{q}, \Delta \boldsymbol{k}). \tag{E.47}$$

Now, the derivative of F splits into two terms

$$\nabla F \diamond (\Delta \boldsymbol{q}, \Delta \boldsymbol{k}, \Delta \boldsymbol{v}) = \boldsymbol{A}(\Delta \boldsymbol{v}) + (\Delta \boldsymbol{A})\boldsymbol{v}. \tag{E.48}$$

To complete the proof, we claim that

$$\|\boldsymbol{A}\|_{\infty-\mathsf{op}} = 1 \quad \text{and} \quad \|\Delta \boldsymbol{A}\|_{\infty-\mathsf{op}} \leq \|\Delta \boldsymbol{q}\|_{\infty\mathsf{RMS}} + \|\Delta \boldsymbol{k}\|_{\infty\mathsf{RMS}}. \tag{E.49}$$

The calculation of the norm of $\boldsymbol{A}$ follows by definition from its construction by softmax. For the calculation of the norm of $\Delta \boldsymbol{A}$, a direct calculation yields that

$$\Delta \boldsymbol{A}_{ij} = \tfrac{1}{d_Q} \boldsymbol{A}_{ij} \langle \Delta \boldsymbol{q}_i,\ \boldsymbol{k}_j - \Sigma_m \boldsymbol{A}_{im} \boldsymbol{k}_m \rangle + \tfrac{1}{d_Q} \boldsymbol{A}_{ij} \langle \boldsymbol{q}_i,\ \Delta \boldsymbol{k}_j - \Sigma_m \boldsymbol{A}_{im} \Delta \boldsymbol{k}_m \rangle \tag{E.50}$$

where we are writing $\boldsymbol{q}_i = \boldsymbol{q}_{i*}$ and so on.

Taking absolute values, applying the Cauchy-Schwarz inequality and summing over j we have

$$\begin{aligned} \Sigma_j |\Delta \boldsymbol{A}_{ij}| \leq & \|\Delta \boldsymbol{q}_i\|_{\mathsf{RMS}} \left(\Sigma_j \boldsymbol{A}_{ij} \|\boldsymbol{k}_j - \Sigma_m \boldsymbol{A}_{im} \boldsymbol{k}_m\|_{\mathsf{RMS}} \right) & \text{(E.51)} \\ & + \|\boldsymbol{q}_i\|_{\mathsf{RMS}} \left(\Sigma_j \boldsymbol{A}_{ij} \|\Delta \boldsymbol{k}_j - \Sigma_m \boldsymbol{A}_{im} \Delta \boldsymbol{k}_m\|_{\mathsf{RMS}} \right). & \text{(E.52)} \end{aligned}$$

We now use the following inequality: given any non-negative reals $p_1, \ldots, p_\ell$ which sum to 1, and any vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell$ in an inner product space with norm $\|\cdot\|$, we have by Jensen's inequality

$$\begin{aligned} \Sigma_j p_j \|\boldsymbol{x}_j - \Sigma_m p_m \boldsymbol{x}_m\| &\leq \left(\Sigma_j p_j \|\boldsymbol{x}_j - \Sigma_m p_m \boldsymbol{x}_m\|^2 \right)^{\frac{1}{2}} & \text{(E.53)} \\ &= \left(\Sigma_j p_j \|\boldsymbol{x}_j\|^2 - \|\Sigma_j p_j \boldsymbol{x}_j\|^2 \right)^{\frac{1}{2}} & \text{(E.54)} \\ &\leq \left(\Sigma_j p_j \|\boldsymbol{x}_j\|^2 \right)^{\frac{1}{2}} & \text{(E.55)} \\ &\leq \max_j \|\boldsymbol{x}_j\|. & \text{(E.56)} \end{aligned}$$

Applying to the matrix $\Delta \boldsymbol{A}$, we thus have

$$\Sigma_j |\boldsymbol{A}_{ij}| \leq \|\Delta \boldsymbol{q}_i\|_{\mathsf{RMS}} \max_j \|\boldsymbol{k}_j\|_{\mathsf{RMS}} + \|\boldsymbol{q}_i\|_{\mathsf{RMS}} \max_j \|\Delta \boldsymbol{k}_j\|_{\mathsf{RMS}}. \tag{E.57}$$

Taking the max over i, this shows the L^∞-operator-norm of $\Delta \boldsymbol{A}$ is at most

$$\|\Delta \boldsymbol{q}\|_{\infty\mathsf{RMS}} \|\boldsymbol{k}\|_{\infty\mathsf{RMS}} + \|\boldsymbol{q}\|_{\infty\mathsf{RMS}} \|\Delta \boldsymbol{k}\|_{\infty\mathsf{RMS}} \tag{E.58}$$

which, since $\|\boldsymbol{q}\|_{\infty\mathsf{RMS}}, \|\boldsymbol{k}\|_{\infty\mathsf{RMS}} \leq 1$, completes the proof.

Proposition 7: Sharpness of functional attention

In this section, we estimate the second derivative of the forward function F of functional attention at $(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v})$ in perturbation directions $(\Delta \boldsymbol{q}, \Delta \boldsymbol{k}, \Delta \boldsymbol{v})$ and $(\Delta \widetilde{\boldsymbol{q}}, \Delta \widetilde{\boldsymbol{k}}, \Delta \widetilde{\boldsymbol{v}})$:

$$\Delta^2 F := (\Delta \widetilde{\boldsymbol{q}}, \Delta \widetilde{\boldsymbol{k}}, \Delta \widetilde{\boldsymbol{q}}) \diamond \nabla^2 F \diamond (\Delta \boldsymbol{q}, \Delta \boldsymbol{k}, \Delta \boldsymbol{v}). \tag{E.59}$$

We will prove that functional attention is γ-sharp where in fact $\gamma = 3$; this amounts to proving that

$$\|\Delta^2 F\| \leq 3 \|(\Delta \boldsymbol{q}, \Delta \boldsymbol{k}, \Delta \boldsymbol{v})\|_{\mathcal{X}} \|(\Delta \widetilde{\boldsymbol{q}}, \Delta \widetilde{\boldsymbol{k}}, \Delta \widetilde{\boldsymbol{v}})\|_{\mathcal{X}}. \tag{E.60}$$

We continue with all the notation of the previous section. Moreover, to simplify the calculation, we suppress all factors of $\frac{1}{d_Q}$ (indeed, one can absorb them as a rescaled inner product $\langle \cdot,\ \cdot \rangle$). We also, in addition to the shorthand $\boldsymbol{x}_i = \boldsymbol{x}_{i*}$ for $\ell \times d$ matrices $\boldsymbol{x}$, we adopt the shorthand for an $\ell \times \ell$ matrix $\boldsymbol{B}$ and a $\ell \times d$ matrix $\boldsymbol{x}$, and any $i, j = 1, \ldots, \ell$:

$$[\boldsymbol{B}, \boldsymbol{x}]_{ij} := \boldsymbol{x}_j - \Sigma_m \boldsymbol{B}_{im} \boldsymbol{x}_m. \tag{E.61}$$

We note three crucial inequalities regarding $[\boldsymbol{B}, \boldsymbol{x}]$, *for any $\ell \times \ell$ matrix $\boldsymbol{B}$ with non-negative entries whose rows sum to 1, and $\ell \times d$ matrices $\boldsymbol{x}, \boldsymbol{y}$:*:

$$\Sigma_j \boldsymbol{B}_{ij} \|[\boldsymbol{B}, \boldsymbol{x}]_{ij}\| \leq \max_j \|\boldsymbol{x}_j\|; \tag{E.62}$$

$$\Sigma_j \boldsymbol{B}_{ij} \|[\boldsymbol{B}, \boldsymbol{x}]_{ij}\|^2 \leq \max_j \|\boldsymbol{x}_j\|^2; \tag{E.63}$$

$$\Sigma_j \boldsymbol{B}_{ij} \|[\boldsymbol{B}, \boldsymbol{x}]_{ij}\| \|[\boldsymbol{B}, \boldsymbol{y}]_{ij}\| \leq (\max_j \|\boldsymbol{x}_j\|)(\max_j \|\boldsymbol{y}_j\|). \tag{E.64}$$

All three inequalities follow from standard expectation/variance inequalities for random variables on the finite set $\{1, \dots, \ell\}$ with distributions given by $\boldsymbol{B}_{i1}, \dots, \boldsymbol{B}_{i\ell}$.

With these conventions, the expression for $\Delta\boldsymbol{A}$ is thus

$$\Delta\boldsymbol{A}_{ij} = \boldsymbol{A}_{ij}\langle\Delta\boldsymbol{q}_i,\ [\boldsymbol{A},\boldsymbol{k}]_{ij}\rangle + \boldsymbol{A}_{ij}\langle\boldsymbol{q}_i,\ [\boldsymbol{A},\Delta\boldsymbol{k}]_{ij}\rangle. \tag{E.65}$$

Let us also write

$$\Delta\widetilde{\boldsymbol{A}} : = \nabla_{(\boldsymbol{q},\boldsymbol{k})}\,\mathrm{softmax}(\tfrac{1}{d_Q}\boldsymbol{q}\boldsymbol{k}^T + \boldsymbol{M}) \diamond (\Delta\widetilde{\boldsymbol{q}}, \Delta\widetilde{\boldsymbol{k}}) \tag{E.66}$$

$$\Delta\widetilde{\boldsymbol{A}}_{ij} = \boldsymbol{A}_{ij}\langle\Delta\widetilde{\boldsymbol{q}}_i,\ [\boldsymbol{A},\boldsymbol{k}]_{ij}\rangle + \boldsymbol{A}_{ij}\langle\boldsymbol{q}_i,\ [\boldsymbol{A},\Delta\widetilde{\boldsymbol{k}}]_{ij}\rangle. \tag{E.67}$$

as well as

$$\Delta^2\boldsymbol{A} := (\Delta\widetilde{\boldsymbol{q}}, \Delta\widetilde{\boldsymbol{k}}) \diamond \nabla^2 F \diamond (\Delta\boldsymbol{q}, \Delta\boldsymbol{k}). \tag{E.68}$$

In these terms, the second derivative $\Delta^2 F$ is just

$$\Delta^2 F = (\Delta\widetilde{\boldsymbol{A}})(\Delta\boldsymbol{v}) + (\Delta\boldsymbol{A})(\Delta\widetilde{\boldsymbol{v}}) + (\Delta^2\boldsymbol{A})\boldsymbol{v}. \tag{E.69}$$

From the estimates of the previous section, we have

$$\|(\Delta\widetilde{\boldsymbol{A}})(\Delta\boldsymbol{v})\|_{\infty\mathsf{RMS}} \le (\|\Delta\widetilde{\boldsymbol{q}}\|_{\infty\mathsf{RMS}} + \|\Delta\widetilde{\boldsymbol{k}}\|_{\infty\mathsf{RMS}})\|\Delta\boldsymbol{v}\|_{\infty\mathsf{RMS}} \tag{E.70}$$

$$\|(\Delta\boldsymbol{A})(\Delta\widetilde{\boldsymbol{v}})\|_{\infty\mathsf{RMS}} \le (\|\Delta\boldsymbol{q}\|_{\infty\mathsf{RMS}} + \|\Delta\boldsymbol{k}\|_{\infty\mathsf{RMS}})\|\Delta\widetilde{\boldsymbol{v}}\|_{\infty\mathsf{RMS}} \tag{E.71}$$

so our task is to estimate the L^∞-operator-norm of $\Delta^2\boldsymbol{A}$. Thus, we calculate $\Delta^2\boldsymbol{A}$:

$$\Delta^2\boldsymbol{A}_{ij} = \boldsymbol{A}_{ij}\langle\Delta\boldsymbol{q}_i,\ [\boldsymbol{A},\Delta\widetilde{\boldsymbol{k}}]_{ij}\rangle \tag{E.72}$$

$$+\ \boldsymbol{A}_{ij}\langle\Delta\widetilde{\boldsymbol{q}}_i,\ [\boldsymbol{A},\Delta\boldsymbol{k}]_{ij}\rangle \tag{E.73}$$

$$+\ \Delta\widetilde{\boldsymbol{A}}_{ij}\langle\Delta\boldsymbol{q}_i,\ [\boldsymbol{A},\boldsymbol{k}]_{ij}\rangle \tag{E.74}$$

$$+\ \Delta\widetilde{\boldsymbol{A}}_{ij}\langle\boldsymbol{q}_i,\ [\boldsymbol{A},\Delta\boldsymbol{k}]_{ij}\rangle \tag{E.75}$$

$$+\ \boldsymbol{A}_{ij}\langle\Delta\boldsymbol{q}_i,\ -\Sigma_m(\Delta\widetilde{\boldsymbol{A}})_{im}\boldsymbol{k}_m\rangle \tag{E.76}$$

$$+\ \boldsymbol{A}_{ij}\langle\boldsymbol{q}_i,\ -\Sigma_m(\Delta\widetilde{\boldsymbol{A}})_{im}\Delta\boldsymbol{k}_m\rangle \tag{E.77}$$

We estimate the L^∞-operator-norm of these six terms one by one. The first (E.72), (E.73) are the simplest, using inequality (E.62):

$$\max_i \Sigma_j|\boldsymbol{A}_{ij}\langle\Delta\boldsymbol{q}_i,\ [\boldsymbol{A},\Delta\widetilde{\boldsymbol{k}}]_{ij}\rangle| \le \max_i \Sigma_j \boldsymbol{A}_{ij}\|\Delta\boldsymbol{q}_i\|\|[\boldsymbol{A},\Delta\widetilde{\boldsymbol{k}}]_{ij}\| \tag{E.78}$$

$$\le \max_i \|\Delta\boldsymbol{q}_i\| \max_j \|\Delta\boldsymbol{k}_j\| \tag{E.79}$$

$$= \|\Delta\boldsymbol{q}\|_{\infty\mathsf{RMS}}\|\Delta\widetilde{\boldsymbol{k}}\|_{\infty\mathsf{RMS}} \tag{E.80}$$

$$\max_i \Sigma_j|\boldsymbol{A}_{ij}\langle\Delta\widetilde{\boldsymbol{q}}_i,\ [\boldsymbol{A},\Delta\boldsymbol{k}]_{ij}\rangle| \le \|\Delta\widetilde{\boldsymbol{q}}\|_{\infty\mathsf{RMS}}\|\Delta\boldsymbol{k}\|_{\infty\mathsf{RMS}} \quad \text{likewise.} \tag{E.81}$$

For the term (E.74), we have

$$\Delta\widetilde{\boldsymbol{A}}_{ij}\langle\Delta\boldsymbol{q}_i,\ [\boldsymbol{A},\boldsymbol{k}]_{ij}\rangle = \Big(\boldsymbol{A}_{ij}\langle\Delta\widetilde{\boldsymbol{q}}_i,\ [\boldsymbol{A},\boldsymbol{k}]_{ij}\rangle + \boldsymbol{A}_{ij}\langle\boldsymbol{q}_i,\ [\boldsymbol{A},\Delta\widetilde{\boldsymbol{k}}]_{ij}\rangle\Big)\langle\Delta\boldsymbol{q}_i,\ [\boldsymbol{A},\boldsymbol{k}]_{ij}\rangle \tag{E.82}$$

Take absolute values, sum over j, and apply Cauchy-Schwarz and inequalities (E.63),(E.64):

$$\Sigma_j|\Delta\widetilde{\boldsymbol{A}}_{ij}\langle\Delta\boldsymbol{q}_i,\ [\boldsymbol{A},\boldsymbol{k}]_{ij}\rangle| \le \Sigma_j\boldsymbol{A}_{ij}\Big(\|\Delta\boldsymbol{q}_i\|\|\Delta\widetilde{\boldsymbol{q}}_i\|\|[\boldsymbol{A},\boldsymbol{k}]_{ij}\|^2 + \|\boldsymbol{q}_i\|\|\Delta\boldsymbol{q}_i\|\|[\boldsymbol{A},\boldsymbol{k}]_{ij}\|\|[\boldsymbol{A},\Delta\widetilde{\boldsymbol{k}}]_{ij}\|\Big) \tag{E.83}$$

$$\le \|\Delta\boldsymbol{q}_i\|\|\Delta\widetilde{\boldsymbol{q}}_i\|\max_j\|\boldsymbol{k}_j\|^2 + \|\boldsymbol{q}_i\|\|\Delta\boldsymbol{q}_i\|(\max_j\|\boldsymbol{k}_j\|)(\max_j\|\Delta\widetilde{\boldsymbol{k}}_j\|). \tag{E.84}$$

Taking the max over i and applying $\|\boldsymbol{q}\|_{\infty\mathsf{RMS}}, \|\boldsymbol{k}\|_{\infty\mathsf{RMS}}, \|\boldsymbol{v}\|_{\infty\mathsf{RMS}} \le 1$:

$$\max_i \Sigma_j|\Delta\widetilde{\boldsymbol{A}}_{ij}\langle\Delta\boldsymbol{q}_i,\ [\boldsymbol{A},\boldsymbol{k}]_{ij}\rangle| \le \|\Delta\boldsymbol{q}\|_{\infty\mathsf{RMS}}\|\Delta\widetilde{\boldsymbol{q}}\|_{\infty\mathsf{RMS}} + \|\Delta\boldsymbol{q}\|_{\infty\mathsf{RMS}}\|\Delta\widetilde{\boldsymbol{k}}\|_{\infty\mathsf{RMS}}. \tag{E.85}$$

The term (E.75) is similar:

$$\max_i \Sigma_j|\Delta\widetilde{\boldsymbol{A}}_{ij}\langle\boldsymbol{q}_i,\ [\boldsymbol{A},\Delta\boldsymbol{k}]_{ij}\rangle| \le \|\Delta\boldsymbol{k}\|_{\infty\mathsf{RMS}}\|\Delta\widetilde{\boldsymbol{q}}\|_{\infty\mathsf{RMS}} + \|\Delta\boldsymbol{k}\|_{\infty\mathsf{RMS}}\|\Delta\widetilde{\boldsymbol{k}}\|_{\infty\mathsf{RMS}} \tag{E.86}$$

For term (E.76), observe that

$$\max_i \|\Sigma_m(\Delta\widetilde{\boldsymbol{A}})_{im}\boldsymbol{k}_m\| \leq \|\Delta\widetilde{\boldsymbol{A}}\|_{\infty-\mathsf{op}}\|\Delta\boldsymbol{k}\|_{\infty\mathsf{RMS}} \tag{E.87}$$

$$\leq \|\Delta\widetilde{\boldsymbol{q}}\|_{\infty\mathsf{RMS}} + \|\Delta\widetilde{\boldsymbol{k}}\|_{\infty\mathsf{RMS}} \tag{E.88}$$

and so by Cauchy-Schwarz and the fact that the rows of $\boldsymbol{A}$ sum to 1:

$$\max_i \Sigma_j |\boldsymbol{A}_{ij}\langle\Delta\boldsymbol{q}_i,\ -\Sigma_m(\Delta\widetilde{\boldsymbol{A}})_{im}\boldsymbol{k}_m\rangle| \leq \max_i \|\Delta\boldsymbol{q}_i\|\|\Sigma_m(\Delta\widetilde{\boldsymbol{A}})_{im}\boldsymbol{k}_m\| \tag{E.89}$$

$$\leq \|\Delta\boldsymbol{q}\|_{\infty\mathsf{RMS}}\|\Delta\widetilde{\boldsymbol{q}}\|_{\infty\mathsf{RMS}} + \|\Delta\boldsymbol{q}\|_{\infty\mathsf{RMS}}\|\Delta\widetilde{\boldsymbol{k}}\|_{\infty\mathsf{RMS}}. \tag{E.90}$$

By a similar argument, for term (E.77) we have:

$$\boldsymbol{A}_{ij}\langle\boldsymbol{q}_i,\ -\Sigma_m(\Delta\widetilde{\boldsymbol{A}})_{im}\Delta\boldsymbol{k}_m\rangle \leq \|\Delta\boldsymbol{k}\|_{\infty\mathsf{RMS}}\|\Delta\widetilde{\boldsymbol{q}}\|_{\infty\mathsf{RMS}} + \|\Delta\boldsymbol{k}\|_{\infty\mathsf{RMS}}\|\Delta\widetilde{\boldsymbol{k}}\|_{\infty\mathsf{RMS}} \tag{E.91}$$

Thus, we have an estimate on the L^∞-operator-norm of $\Delta^2\boldsymbol{A}$:

$$\|\Delta^2\boldsymbol{A}\|_{\infty-\mathsf{op}} \leq 2\|\Delta\boldsymbol{q}\|\|\Delta\widetilde{\boldsymbol{q}}\| + 3\|\Delta\boldsymbol{q}\|\|\Delta\widetilde{\boldsymbol{k}}\| + 3\|\Delta\boldsymbol{k}\|\|\Delta\widetilde{\boldsymbol{q}}\| + 2\|\Delta\boldsymbol{k}\|\|\Delta\widetilde{\boldsymbol{k}}\| \tag{E.92}$$

where all the norms on the right hand side are $\|\cdot\|_{\infty\mathsf{RMS}}$.

Adding this together with (E.70) and (E.71), we obtain (all norms being $\|\cdot\|_{\infty\mathsf{RMS}}$:

$$\|\Delta^2 F\| \leq 2\|\Delta\boldsymbol{q}\|\|\Delta\widetilde{\boldsymbol{q}}\| + 3\|\Delta\boldsymbol{q}\|\|\Delta\widetilde{\boldsymbol{k}}\| + 3\|\Delta\boldsymbol{k}\|\|\Delta\widetilde{\boldsymbol{q}}\| + 2\|\Delta\boldsymbol{k}\|\|\Delta\widetilde{\boldsymbol{k}}\| \tag{E.93}$$

$$+ \|\Delta\boldsymbol{v}\|\|\Delta\widetilde{\boldsymbol{q}}\| + \|\Delta\boldsymbol{v}\|\|\Delta\widetilde{\boldsymbol{k}}\| + \|\Delta\boldsymbol{q}\|\|\Delta\widetilde{\boldsymbol{v}}\| + \|\Delta\boldsymbol{k}\|\|\Delta\widetilde{\boldsymbol{v}}\| \tag{E.94}$$

$$\leq 3(\|\Delta\boldsymbol{q}\| + \|\Delta\boldsymbol{k}\| + \|\Delta\boldsymbol{v}\|)(\|\Delta\widetilde{\boldsymbol{q}}\| + \|\Delta\widetilde{\boldsymbol{k}}\| + \|\Delta\widetilde{\boldsymbol{v}}\|) \tag{E.95}$$

which is the desired result.

Proposition 8: Sharpness under composition

Suppose $\mathsf{M} = \mathsf{M}_2 \circ \mathsf{M}_1$ where $\mathsf{M}_1, \mathsf{M}_2$ are well-normed modules on respectively $(\mathcal{X}_k, \mathcal{Y}_k, \mathcal{W}_k)$ and moreover $(\alpha_k, \beta_k, \gamma_k)$-sharp for $k = 1, 2$. If $p_k = \frac{\mathsf{M}_k.\mathsf{mass}}{\mathsf{M}.\mathsf{mass}}$ for $k = 1, 2$, note that by the definition of the modular norm on the composite M, we have for any $\Delta\boldsymbol{w} = (\Delta\boldsymbol{w}_1, \Delta\boldsymbol{w}_2) \in \mathcal{W}_1 \times \mathcal{W}_2$:

$$\|\Delta\boldsymbol{w}_1\|_{\mathsf{M}_1} \leq \tfrac{1}{\mu_2}p_1\|\Delta\boldsymbol{w}\|_{\mathsf{M}} \quad \text{and} \quad \|\Delta\boldsymbol{w}_2\|_{\mathsf{M}_2} \leq p_2\|\Delta\boldsymbol{w}\|_{\mathsf{M}}. \tag{E.96}$$

We must prove that M is (α, β, γ) sharp where:

$$\alpha = \tfrac{1}{\mu_2}p_1^2\alpha_1 + p_2^2\alpha_2 + \tfrac{2}{\mu_2}p_1p_2\beta_2 + \tfrac{1}{\mu_2^2}p_1^2\gamma_2, \tag{E.97}$$

$$\beta = p_1\beta_1 + \mu_1p_2\beta_2 + \tfrac{\mu_1}{\mu_2}p_1\gamma_2, \tag{E.98}$$

$$\gamma = \mu_2\gamma_1 + \mu_1^2\gamma_2. \tag{E.99}$$

Turning to the second derivative of $\mathsf{M}(\cdot,\cdot)$, we prove the first Inequality (E.97). The Gauss-Newton decomposition (C.1) for any $\Delta\boldsymbol{w} = (\Delta\boldsymbol{w}_1, \Delta\boldsymbol{w}_2)$ and $\Delta\widetilde{\boldsymbol{w}} = \Delta\widetilde{\boldsymbol{w}}_1, \Delta\widetilde{\boldsymbol{w}}_2$ yields

$$\Delta\boldsymbol{w} \diamond \nabla^2\mathsf{M} \diamond \Delta\widetilde{\boldsymbol{w}} = \nabla\mathsf{M}_2 \diamond (\Delta\boldsymbol{w}_1 \diamond \nabla^2\mathsf{M}_1 \diamond \Delta\widetilde{\boldsymbol{w}}_1) \tag{E.100}$$

$$+ (\Delta\boldsymbol{w}_2, \nabla\mathsf{M}_1 \diamond \Delta\boldsymbol{w}_1) \diamond \nabla^2\mathsf{M}_2 \diamond (\Delta\widetilde{\boldsymbol{w}}_2, \nabla\mathsf{M}_1 \diamond \Delta\widetilde{\boldsymbol{w}}_1) \tag{E.101}$$

Applying the well-normed and sharpness inequalities, the norm of the first (E.100) of these terms is bounded by

$$\mu_2\|\Delta\boldsymbol{w}_1 \diamond \nabla^2\mathsf{M}_1 \diamond \Delta\widetilde{\boldsymbol{w}}_1\|_{\mathcal{Y}_1} \leq \mu_2\alpha_1\|\Delta\boldsymbol{w}_1\|_{\mathsf{M}_1}\|\Delta\widetilde{\boldsymbol{w}}_1\|_{\mathsf{M}_1} \tag{E.102}$$

$$\leq \tfrac{1}{\mu_2}p_1^2\alpha_1\|\Delta\boldsymbol{w}\|_{\mathsf{M}}\|\Delta\widetilde{\boldsymbol{w}}\|_{\mathsf{M}}. \tag{E.103}$$

The second term (E.101) breaks into four separate terms:

$$\Delta\boldsymbol{w}_2 \diamond \nabla^2_{\boldsymbol{ww}}\mathsf{M}_2 \diamond \Delta\widetilde{\boldsymbol{w}}_2 \tag{E.104}$$

$$+(\nabla\mathsf{M}_1 \diamond \Delta\boldsymbol{w}_1) \diamond \nabla^2_{\boldsymbol{xw}}\mathsf{M}_2 \diamond \Delta\widetilde{\boldsymbol{w}_2} \tag{E.105}$$

$$+\Delta\boldsymbol{w}_2 \diamond \nabla^2_{\boldsymbol{wx}}\mathsf{M}_2 \diamond (\nabla\mathsf{M}_1 \diamond \Delta\widetilde{\boldsymbol{w}}_1) \tag{E.106}$$

$$+(\nabla\mathsf{M}_1 \diamond \Delta\boldsymbol{w}_1) \diamond \nabla^2_{\boldsymbol{xx}}\mathsf{M}_2 \diamond (\nabla\mathsf{M}_1 \diamond \Delta\widetilde{\boldsymbol{w}}_1). \tag{E.107}$$

In particular, applying the well-normed and sharpness inequalities, this is bounded by

$$\begin{aligned}
&\alpha_2\|\Delta\boldsymbol{w}_2\|_{\mathsf{M}_2}\|\Delta\widetilde{\boldsymbol{w}}_2\|_{\mathsf{M}_2} && \text{(E.108)}\\
&+\beta_2\|\Delta\boldsymbol{w}_1\|_{\mathsf{M}_1}\|\Delta\widetilde{\boldsymbol{w}}_2\|_{\mathsf{M}_2} && \text{(E.109)}\\
&+\beta_2\|\Delta\boldsymbol{w}_2\|_{\mathsf{M}_2}\|\Delta\widetilde{\boldsymbol{w}}_1\|_{\mathsf{M}_1} && \text{(E.110)}\\
&+\gamma_2\|\Delta\boldsymbol{w}_1\|_{\mathsf{M}_1}\|\Delta\widetilde{\boldsymbol{w}}_1\|_{\mathsf{M}_1}, && \text{(E.111)}
\end{aligned}$$

which is less than

$$\left(p_2^2\alpha_2 + \tfrac{2}{\mu_2}p_1p_2\beta_2 + \tfrac{1}{\mu_2^2}p_1^2\gamma_2\right)\|\Delta\boldsymbol{w}\|_{\mathsf{M}}\|\Delta\boldsymbol{w}\|_{\mathsf{M}} \tag{E.112}$$

which completes the proof of Inequality (C.4).

Inequalities (E.98) and (E.99) are simpler. For the first of these, note we have

$$\begin{aligned}
\Delta\boldsymbol{w}\diamond\nabla^2\mathsf{M}\diamond\Delta\boldsymbol{x} =&\nabla\mathsf{M}_2\diamond(\Delta\boldsymbol{w}_1\diamond\nabla^2\mathsf{M}_1\diamond\Delta\boldsymbol{x}) && \text{(E.113)}\\
&+(\Delta\boldsymbol{w}_2,\nabla\mathsf{M}_1\diamond\Delta\boldsymbol{w}_1)\diamond\nabla^2\mathsf{M}_2\diamond(\nabla\mathsf{M}_1\diamond\Delta\boldsymbol{x}). && \text{(E.114)}
\end{aligned}$$

Term (E.113) is bounded by

$$\begin{aligned}
\mu_2\|\Delta\boldsymbol{w}_1\diamond\nabla^2\mathsf{M}_1\diamond\Delta\boldsymbol{x}\|_{\mathcal{Y}_1} &\leq \mu_2\beta_1\|\Delta\boldsymbol{w}_1\|_{\mathsf{M}_1}\|\Delta\boldsymbol{x}\|_{\mathcal{X}_1} && \text{(E.115)}\\
&\leq p_1\beta_1\|\Delta\boldsymbol{w}\|_{\mathsf{M}}\|\Delta\boldsymbol{x}\|_{\mathcal{X}_1} && \text{(E.116)}
\end{aligned}$$

Term (E.114) breaks into two separate terms

$$\Delta\boldsymbol{w}_2\diamond\nabla^2_{\boldsymbol{w}\boldsymbol{x}}\mathsf{M}_2\diamond(\nabla\mathsf{M}_1\diamond\Delta\boldsymbol{x}) + (\nabla\mathsf{M}_1\diamond\Delta\boldsymbol{w}_1)\diamond\nabla^2_{\boldsymbol{x}\boldsymbol{x}}\mathsf{M}_2\diamond(\nabla\mathsf{M}_1\diamond\Delta\boldsymbol{x}). \tag{E.117}$$

In particular this is bounded by

$$\beta_2\|\Delta\boldsymbol{w}_2\|_{\mathsf{M}_2}\mu_1\|\Delta\boldsymbol{x}\|_{\mathcal{X}_1} + \gamma_2\|\Delta\boldsymbol{w}_1\|_{\mathsf{M}_1}\mu_1\|\Delta\boldsymbol{x}\|_{\mathcal{X}_1} \leq \left(\mu_1p_2\beta_2 + \tfrac{\mu_1}{\mu_2}p_1\gamma_2\right)\|\Delta\boldsymbol{w}\|_{\mathsf{M}}\|\Delta\boldsymbol{x}\|_{\mathcal{X}_1} \tag{E.118}$$

which completes the proof of Inequality (E.98).

Finally, for (E.99), we have

$$\begin{aligned}
\Delta\boldsymbol{x}\diamond\nabla^2\mathsf{M}\diamond\Delta\widetilde{\boldsymbol{x}} =&\nabla\mathsf{M}_2\diamond(\Delta\boldsymbol{x}\diamond\nabla^2\mathsf{M}_1\diamond\Delta\widetilde{\boldsymbol{x}}) && \text{(E.119)}\\
&+(\nabla\mathsf{M}_1\diamond\Delta\boldsymbol{x})\diamond\Delta^2\mathsf{M}_2\diamond(\nabla\mathsf{M}_1\diamond\Delta\widetilde{\boldsymbol{x}}). && \text{(E.120)}
\end{aligned}$$

Term (E.119) is bounded by

$$\mu_2\|\Delta\boldsymbol{x}\diamond\nabla^2\mathsf{M}_1\diamond\Delta\widetilde{\boldsymbol{x}}\|_{\mathcal{Y}_1} \leq \mu_2\gamma_1\|\Delta\boldsymbol{x}\|_{\mathcal{X}_1}\|\Delta\widetilde{\boldsymbol{x}}\|_{\mathcal{X}_1} \tag{E.121}$$

while Term (E.120) is bounded by

$$\gamma_2\|\nabla\mathsf{M}_1\diamond\Delta\boldsymbol{x}\|_{\mathcal{X}_2}\|\nabla\mathsf{M}_1\diamond\Delta\widetilde{\boldsymbol{x}}\|_{\mathcal{X}_2} \leq \mu_1^2\gamma_2\|\Delta\boldsymbol{x}\|_{\mathcal{X}_1}\|\Delta\widetilde{\boldsymbol{x}}\|_{\mathcal{X}_1} \tag{E.122}$$

which together give Inequality (E.99).

Proposition 9: Sharpness under concatenation

Suppose $\mathsf{M} = (\mathsf{M}_1, \mathsf{M}_2)$ where $\mathsf{M}_1, \mathsf{M}_2$ are well-normed modules on respectively $(\mathcal{X}_k, \mathcal{Y}_k, \mathcal{W}_k)$ and moreover $(\alpha_k, \beta_k, \gamma_k)$-sharp for $k = 1, 2$. If $p_k = \frac{\mathsf{M}_k.\mathsf{mass}}{\mathsf{M}.\mathsf{mass}}$ for $k = 1, 2$, as in the previous proof we have for any $\Delta\boldsymbol{w} = (\Delta\boldsymbol{w}_1, \Delta\boldsymbol{w}_2) \in \mathcal{W}_1\times\mathcal{W}_2$:

$$\|\Delta\boldsymbol{w}_1\|_{\mathsf{M}_1} \leq \tfrac{1}{\mu_2}p_1\|\Delta\boldsymbol{w}\|_{\mathsf{M}} \quad\text{and}\quad \|\Delta\boldsymbol{w}_2\|_{\mathsf{M}_2} \leq p_2\|\Delta\boldsymbol{w}\|_{\mathsf{M}}. \tag{E.123}$$

We must prove that M is (α, β, γ)-sharp where

$$\begin{aligned}
\alpha &= p_1^2\alpha_1 + p_2^2\alpha_2, && \text{(E.124)}\\
\beta &= p_1\beta_1 + p_2\beta_2, && \text{(E.125)}\\
\gamma &= \gamma_1 + \gamma_2. && \text{(E.126)}
\end{aligned}$$

Now, for the first of these identities, we have for $\Delta\boldsymbol{w} = (\Delta\boldsymbol{w}_1, \Delta\boldsymbol{w}_2)$ and $\Delta\widetilde{\boldsymbol{w}} = (\Delta\widetilde{\boldsymbol{w}}_1, \Delta\widetilde{\boldsymbol{w}}_2)$:

$$\begin{aligned}
\|\Delta\boldsymbol{w}\diamond\nabla^2\mathsf{M}\diamond\Delta\widetilde{\boldsymbol{w}}\|_{\mathcal{Y}_1\times\mathcal{Y}_2} &= \|(\Delta\boldsymbol{w}_1\diamond\nabla^2\mathsf{M}_1\diamond\Delta\widetilde{\boldsymbol{w}}_1, \Delta\boldsymbol{w}_2\diamond\nabla^2\mathsf{M}_2\diamond\Delta\widetilde{\boldsymbol{w}}_2)\|_{\mathcal{Y}_1\times\mathcal{Y}_2} && \text{(E.127)}\\
&= \|\Delta\boldsymbol{w}_1\diamond\nabla^2\mathsf{M}_1\diamond\Delta\widetilde{\boldsymbol{w}}_1\|_{\mathcal{Y}_1} + \|\Delta\boldsymbol{w}_2\diamond\nabla^2\mathsf{M}_2\diamond\Delta\widetilde{\boldsymbol{w}}_2)\|_{\mathcal{Y}_2} && \text{(E.128)}\\
&\leq \alpha_1\|\Delta\boldsymbol{w}_1\|^2_{\mathsf{M}_1} + \alpha_2\|\Delta\boldsymbol{w}_2\|^2_{\mathsf{M}_2} && \text{(E.129)}\\
&\leq (p_1^2\alpha_1 + p_2^2\alpha_2)\,\|\Delta\boldsymbol{w}\|^2_{\mathsf{M}} && \text{(E.130)}
\end{aligned}$$

which shows $\alpha = p_1^2\alpha_1 + p_2^2\alpha_2$. The expressions for β, γ follow similarly.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our main claim is that normalizing Adam and SGD updates in the modular norm leads to good learning rate transfer across width and depth. We believe this claim is supported by the experiments in our paper.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss limitations in the discussion section (§5).

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: We state theoretical results as formal propositions and provide their proofs in Appendix E.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: See Appendix A for an overview of our code, Appendix B for the detailed network architectures and Appendix D for the parameters of our experiments. In addition, we provide the source code for our experiments and the Modula package.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We make use of standard datasets and provide our code in the supplemental materials.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: See Appendix D for the full details of our experiments. Also, see our code.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Due to computational resource constraints, we opted to run a large number of experiments to check that trends hold across several distinct architectures and datasets, rather than running repeats on each individual experiment. Each hyperparameter sweep involves on the order of 100 training runs, and we are working under academic resource limits. We believe that the fact the reported trends hold across varied experimental settings supports the significance of our results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We report on this in Appendix D.3.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We believe that no ethics guidelines were violated.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: No societal impact of the work was discussed. Potentially the work could have a positive impact in terms of reducing carbon emissions caused by sweeping hyperparameters for large-scale models.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: We are not releasing any new datasets or pre-trained models.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We only use public and open-source resources. We have cited these works. Licenses were not provided from the original source.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We provide code and instructions on how to use the new modules that we define.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: We did not crowdsource and we did not use human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: We did not use human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.