

Loss Landscapes Part 2

Edge of Stability

10/15

Make sure to go to your tutorial!

How to carry out assignments

- It is extremely tempting to go “Chat, do this report for me”. Unfortunately, you will likely get suboptimal experiments and retain less content. Instead, plan out the experiments of interest in natural language, specifying hyperparameters as well as what relationships you’d like to see plotted. Then, you can leave implementation and job monitoring to models
- Always pre-register an experiment outcome. Hypothesis generation is one of the most important parts of the scientific process, and you won’t learn anything from your experiment if you don’t have a perspective coming. Plus, every experiment you run is now free practice!

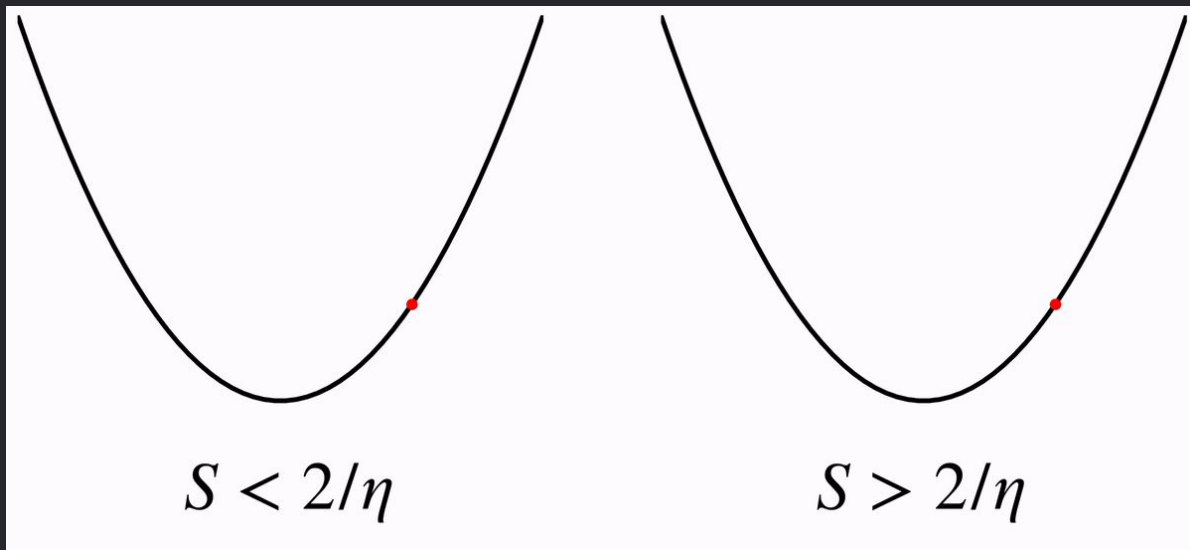
Overview

Today, we will study sharpness

- 1) Edge of Stability: How does the sharpness evolve over training?
 - a) Full batch plain GD followed by modern optimizer components
 - b) We will be closely following the tutorial at <https://centralflows.github.io> by Jeremy Cohen and Alex Damian
- 2) How does sharpness influence model performance when limited by data?
 - a) Note that this will touch on concepts from the Generalization unit
- 3) Optimizing Sharpness

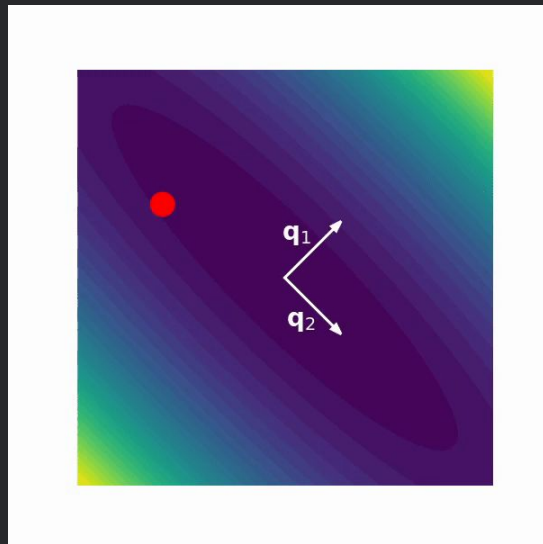
Traditional quadratic intuition

Suppose we were performing SGD on a quadratic $(1/2)Sx^2$. In this case, we must set the learning rate so that $S < 2/\eta$. Otherwise, training will diverge



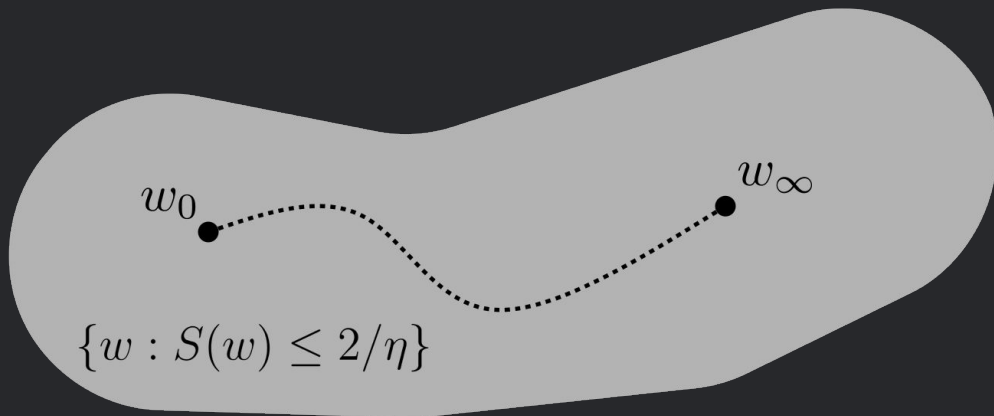
Traditional quadratic intuition

In multiple dimensions, this condition is equivalent to all the eigenvalues of the Hessian staying under $2/\eta$. If not, one of the directions will oscillate. Recall that we had the same conditions for learning rate in the NQM with h_1 .



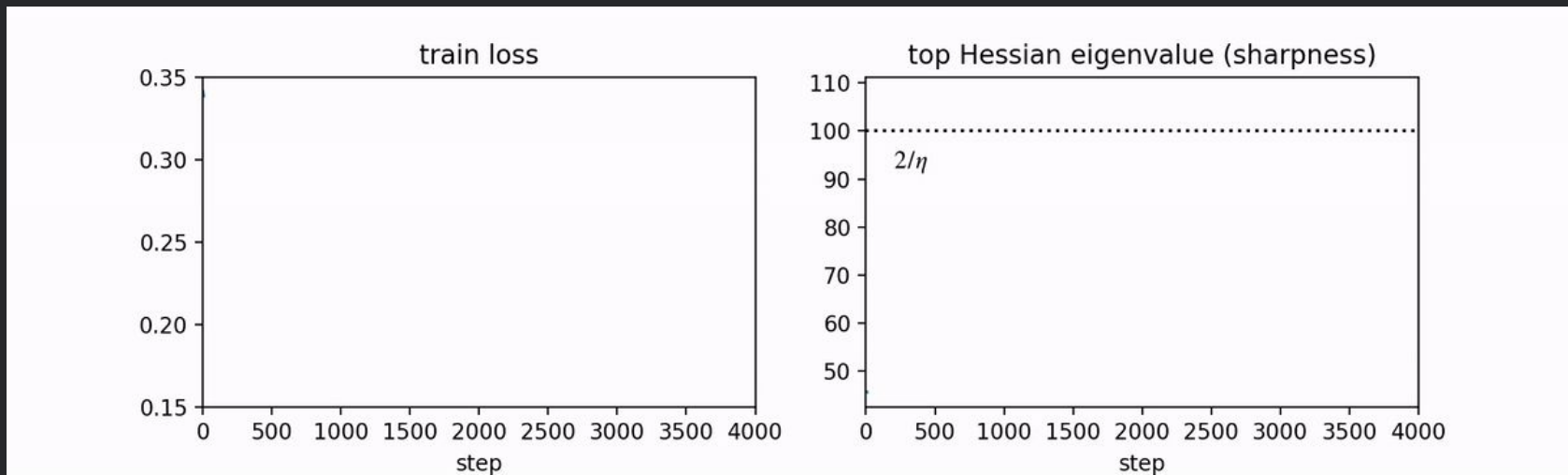
Sharpness in deep networks

Under this perspective, perhaps gradient descent for deep learning also requires this stability and always maintains sharpness under $2/\eta$



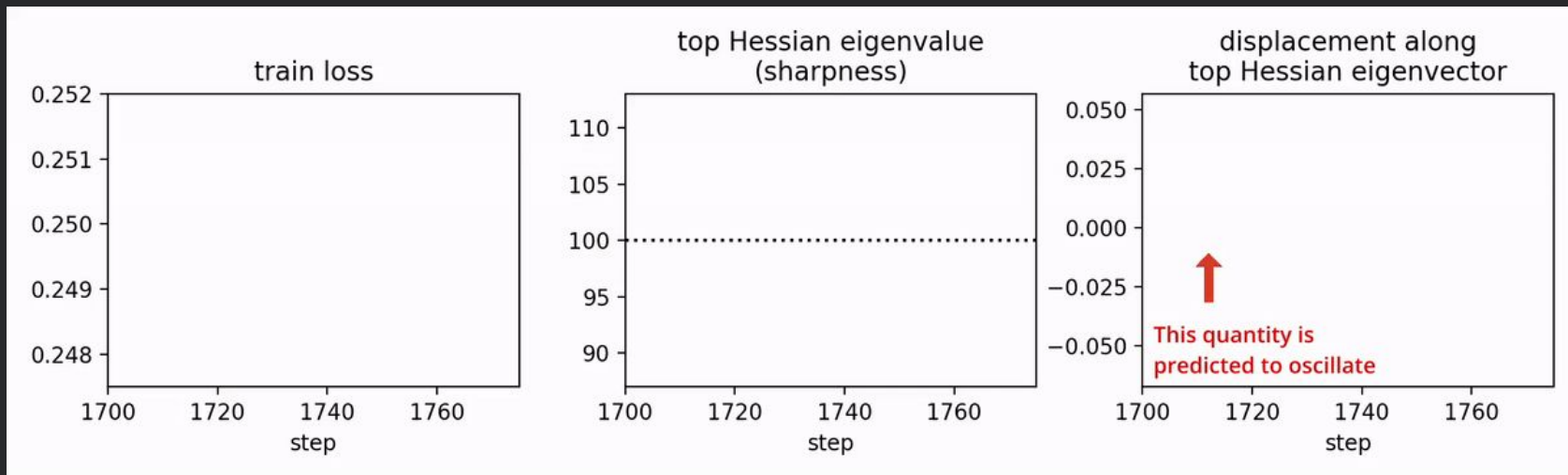
Sharpness in deep networks

This mental model would predict that if the sharpness ever exceeds $2/\eta$, training would diverge



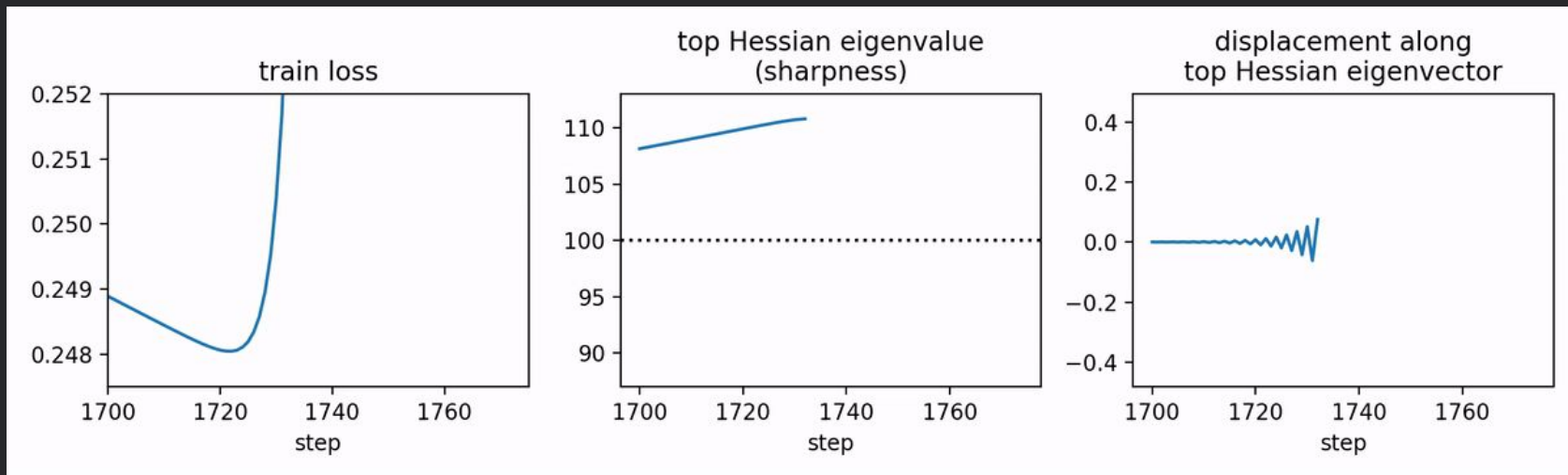
Sharpness in deep networks

Similar to quadratics, the the loss starts to blow up and the model starts to oscillate around the top eigenvalue of the Hessian



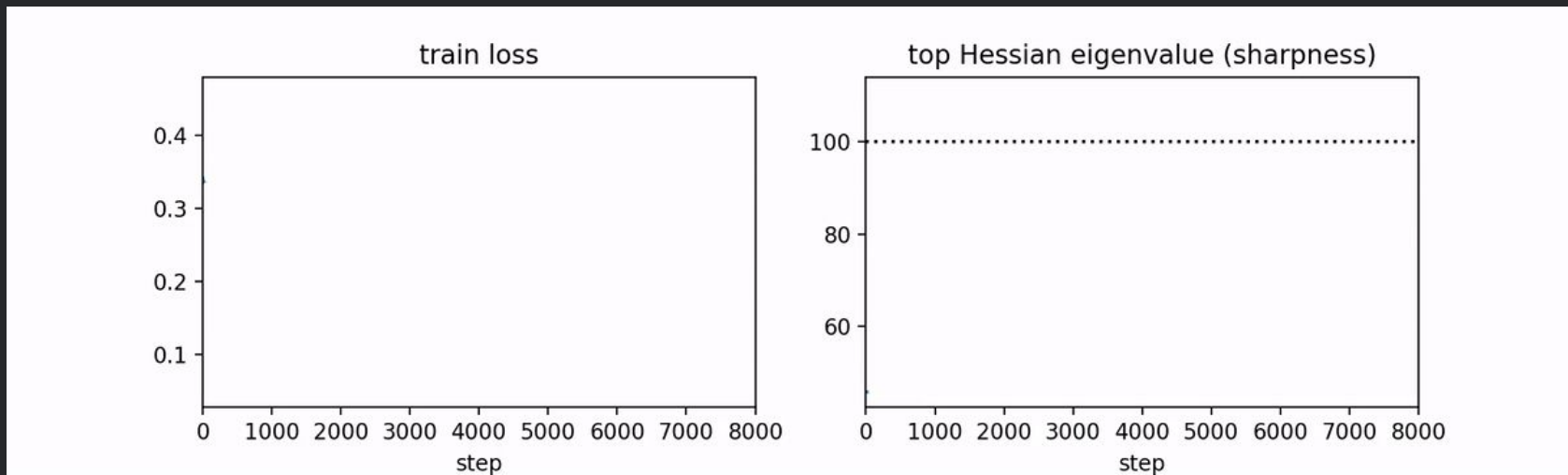
Sharpness in deep networks

Interestingly, with enough oscillation the sharpness drops back down! This deviates from quadratics



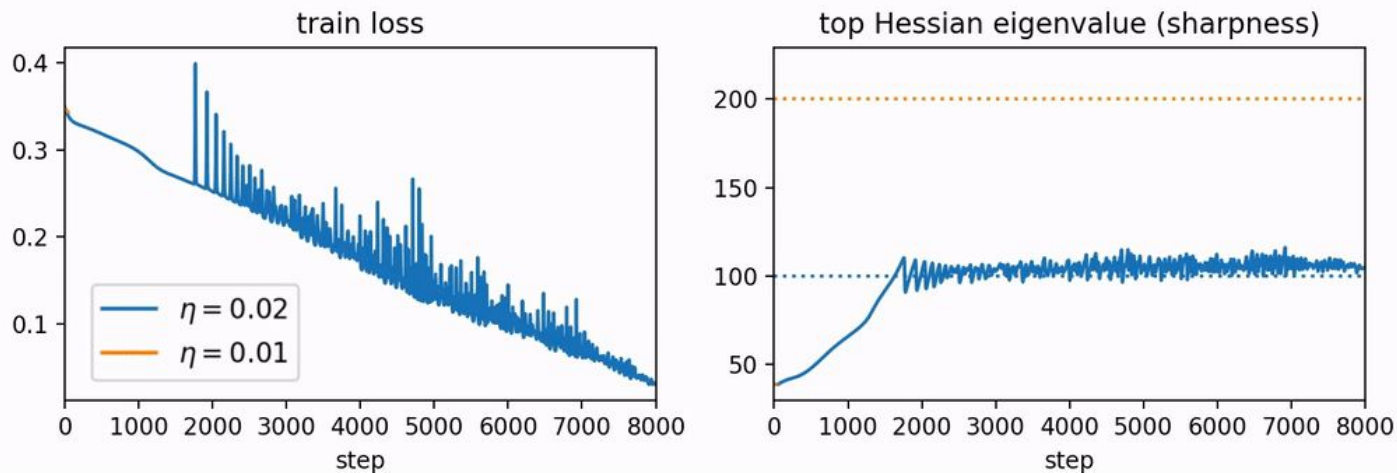
Sharpness in deep networks

We can now look at the full run to see how loss (inconsistently) goes down while the sharpness equilibrates around $2/\eta$



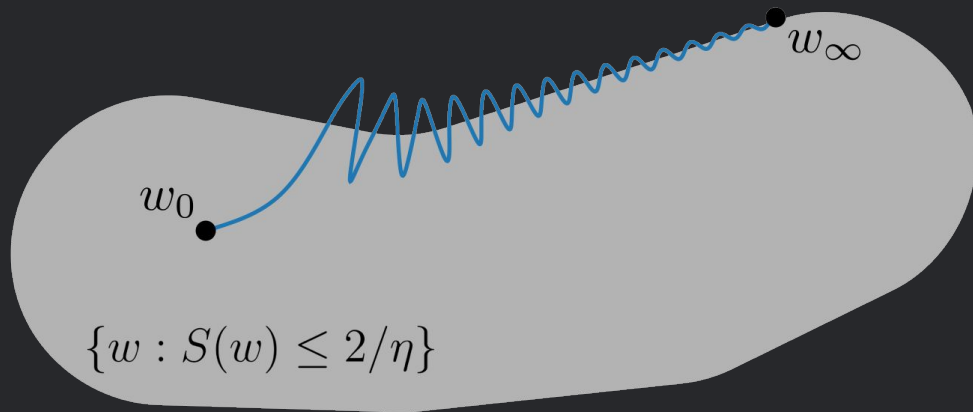
Sharpness in deep networks

The same phenomenon occurs at a different learning rate with the sharpness equilibrating at a correspondingly different threshold



Sharpness in deep networks

Therefore, the traditional picture is somewhat flawed and gradient descent hovers around the “Edge of Stability”



GD vs SGD

The loss curves in the preceding plot have unusually exaggerated loss spikes due to the use of full-batch GD. Nonetheless, similar behavior also exists for SGD with a different threshold [[Jastrzebski et al, 2018](#)]

In your assignment, you will spend the majority of Problem 3 exploring the Edge of Stability in the typical minibatch regime

Self-stabilization (informal)

Oscillation around the top eigenvector can be written as $w = \bar{w} + xu$

Take a Taylor expansion of the loss around this weight. Keeping the first and second term yields a quadratic approximation

$$\nabla L(\bar{w} + xu) = \overbrace{\nabla L(\bar{w})}^{\text{first term}} + \underbrace{xH(\bar{w})u}_{= xS(\bar{w})u} + \mathcal{O}(x^2)$$

This approximation behaves like a quadratic model where we will move in the opposite direction of u , causing oscillation for large sharpness

Self-stabilization (informal)

The magic comes from a third-order Taylor expansion

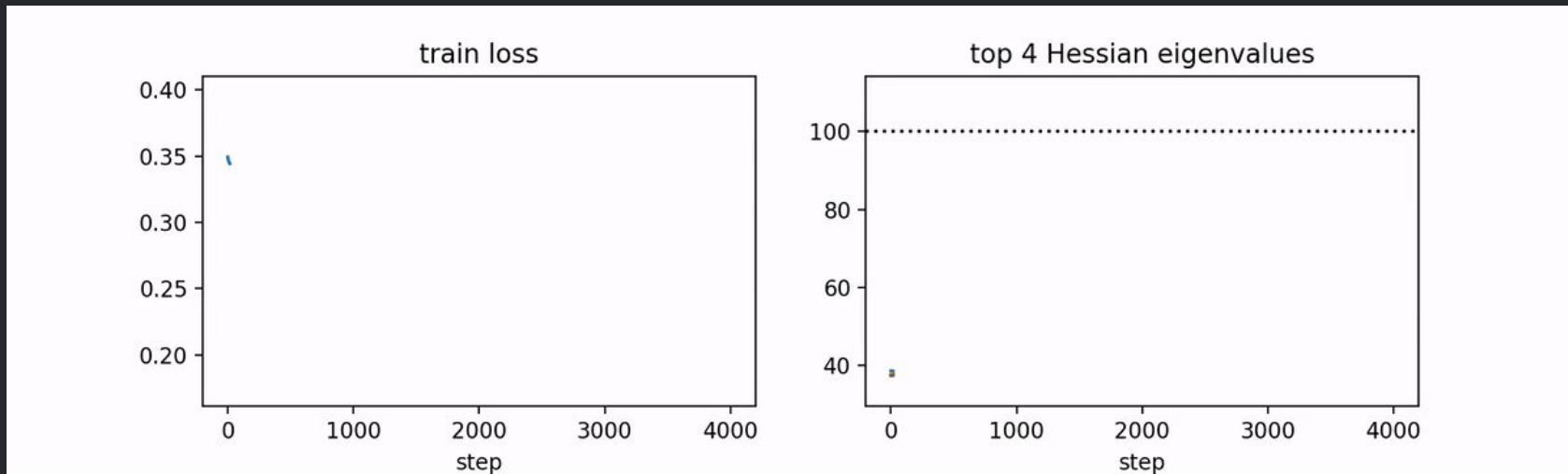
$$\nabla L(\bar{w} + xu) = \overbrace{\nabla L(\bar{w})}^{\text{first term}} + \overbrace{xS(\bar{w})u}^{\text{second term}} + \underbrace{\frac{1}{2}x^2 \nabla_{\bar{w}} [u^T H(\bar{w}) u]}_{\text{third term}} + \mathcal{O}(x^3)$$

$= \frac{1}{2}x^2 \nabla S(\bar{w})$

The inner quantity is the curvature in the u direction, equivalent to the eigenvalue in the u direction. Therefore, this expression takes the gradient of the sharpness and minimizes it with effective step size $(1/2)\eta x^2$

Multiple eigenvalues

We've analyzed only the top eigenvalue. When there are multiple eigenvalues, it can be observed that multiple of them eventually reach the edge of stability and oscillate at this threshold



Gradient Flow

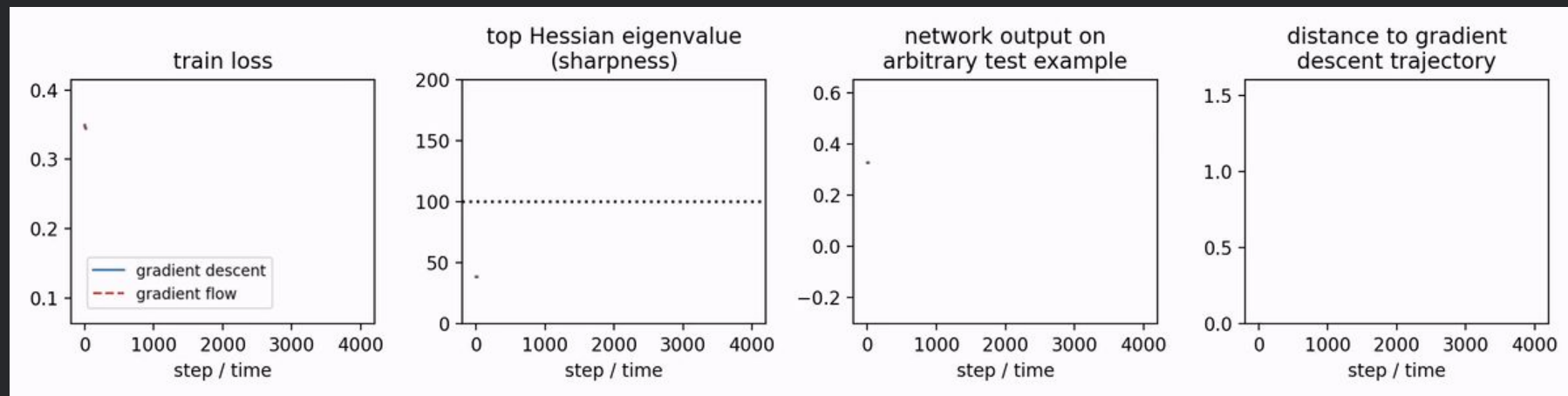
Though the exact dynamics are difficult to analyze, is there a theoretical construct that can capture the macroscopic properties of training?

The typical approach is to consider the continuous-time approximation to gradient descent, a differential equation called *gradient flow*

$$\frac{dw}{dt} = -\eta \nabla L(w),$$

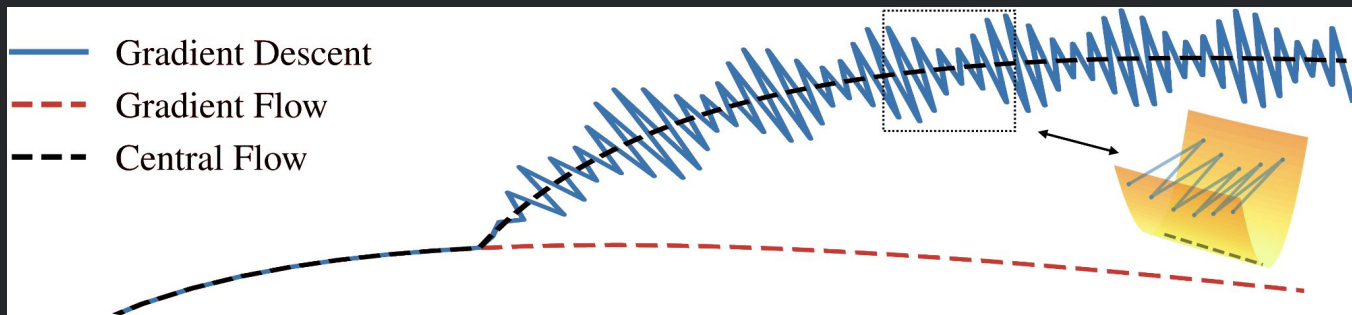
Gradient Flow

Gradient flow is actually pretty good at capturing training before sharpness exceeds the edge of stability. However, past this regime, gradient flow continues to increase sharpness, diverging from empirics



Central Flow

The central flow models gradient descent via its time-averaged trajectory



Informal proof sketch: assume time-averaged trajectory follows a differential equation, show that only one makes sense, and show that it agrees with empirics

Central Flow

When only one eigenvalue is at edge of stability, we can write each iterate as

$$\underset{\text{iterate}}{w_t} = \underset{\text{time-averaged iterate}}{w(t)} + \underset{\text{perturbation along top eigenvector}}{x_t u(t)}.$$

with Taylor expansion

$$\underset{\text{gradient at iterate}}{\nabla L(w_t)} \approx \underset{\text{gradient at time-averaged iterate}}{\nabla L(w(t))} + \underset{\text{oscillation}}{x_t S(w(t)) u(t)} + \underset{\text{sharpness reduction}}{\frac{1}{2} x_t^2 \nabla S(w(t))}.$$

By abusing notation, we can cancel out the oscillation

$$\underset{\text{time-averaged gradient}}{\mathbb{E}[\nabla L(w_t)]} \approx \underset{\text{gradient at time-averaged iterate}}{\nabla L(w(t))} + \cancel{\underset{\text{0 because } \mathbb{E}[x_t]=0}{\mathbb{E}[x_t] S(w(t)) u(t)}} + \underset{\text{implicit sharpness penalty}}{\frac{1}{2} \mathbb{E}[x_t^2] \nabla S(w(t))}.$$

Central Flow

This corresponds to a differential equation with unknown sharpness penalty that depends on the oscillations

$$\frac{dw}{dt} = -\eta \left[\nabla L(w) + \underbrace{\frac{1}{2} \sigma^2(t) \nabla S(w)}_{\text{sharpness penalty}} \right],$$

We know that when we enter the edge of stability, the time-averaged sharpness stops changing. Therefore, we will enforce that the time derivative of the sharpness along the central flow is zero

Central Flow

Interestingly, there is only one such choice that meets this property

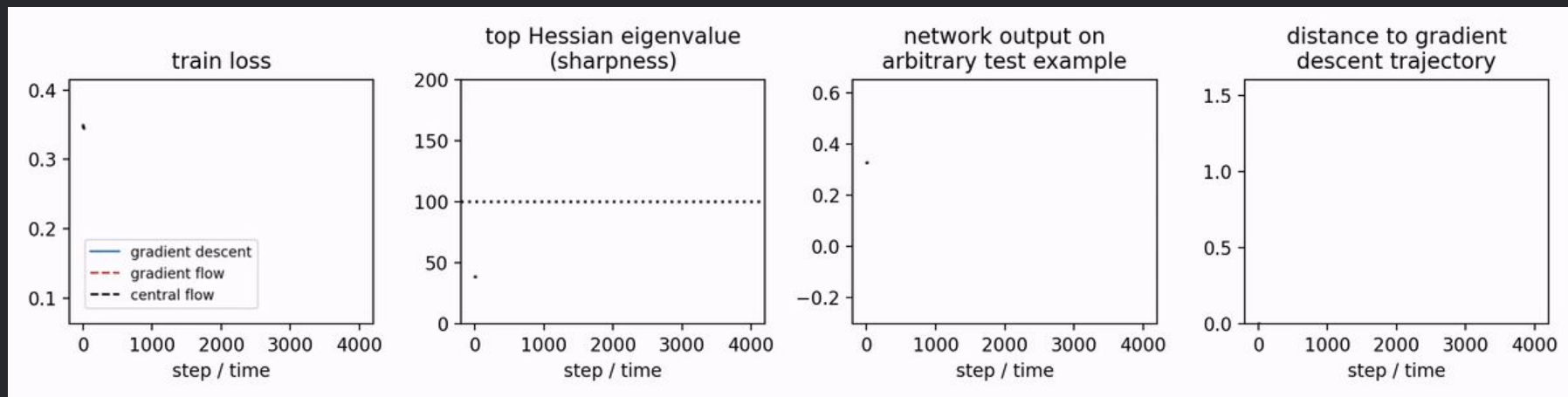
$$\begin{aligned}\frac{dS}{dt} &= \left\langle \nabla S(w), \frac{dw}{dt} \right\rangle && \text{(chain rule)} \\ &= \left\langle \nabla S(w), -\eta \left[\nabla L(w) + \frac{1}{2} \sigma^2(t) \nabla S(w) \right] \right\rangle && \text{(ansatz for } \frac{dw}{dt} \text{)} \\ &= \underbrace{\langle \nabla S(w), -\eta \nabla L(w) \rangle}_{\text{time derivative of sharpness under gradient flow}} - \underbrace{\frac{1}{2} \eta \sigma^2(t) \|\nabla S(w)\|^2}_{\text{sharpness reduction from oscillations}}. && \text{(simplify)}\end{aligned}$$

This gives us the unique solution

$$\sigma^2(t) = \frac{2 \langle -\nabla L(w), \nabla S(w) \rangle}{\|\nabla S(w)\|^2}.$$

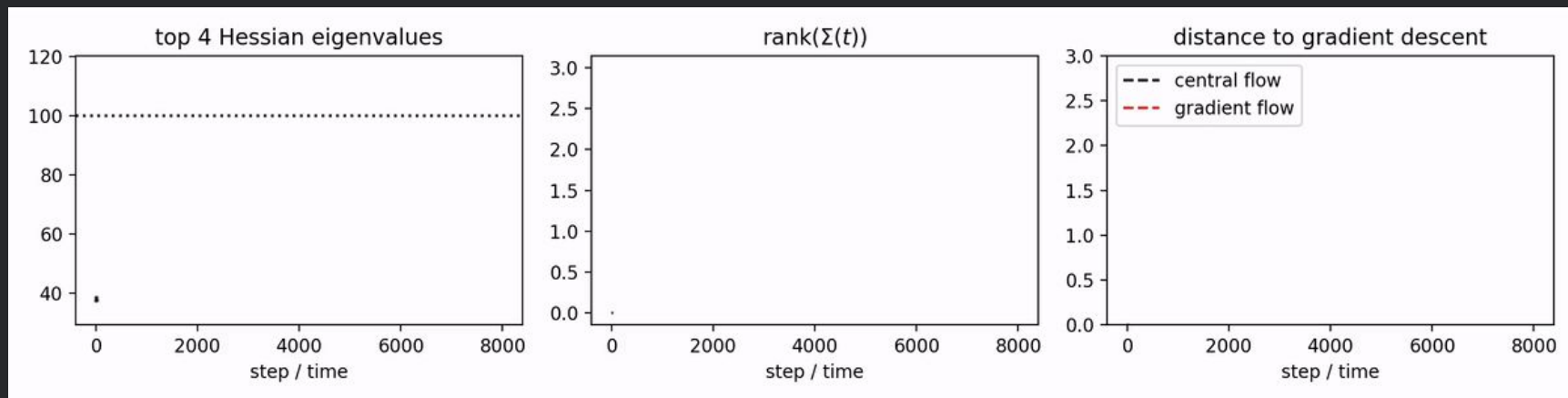
Central Flow

Now we can assess its empirical predictions. We find that it tracks the empirical predictions quite well



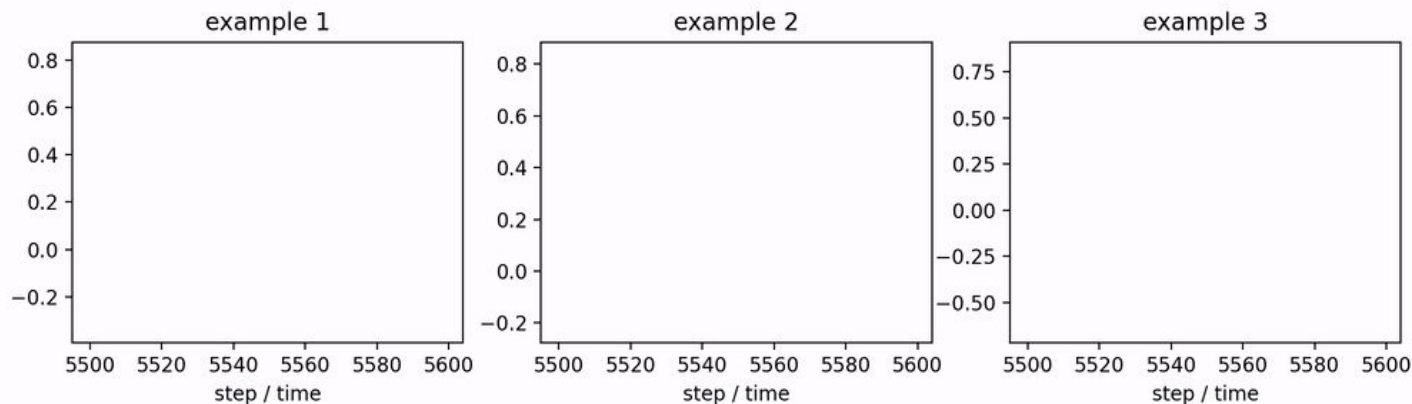
Central Flow

You can also generalize it to multiple eigenvectors and track each separately



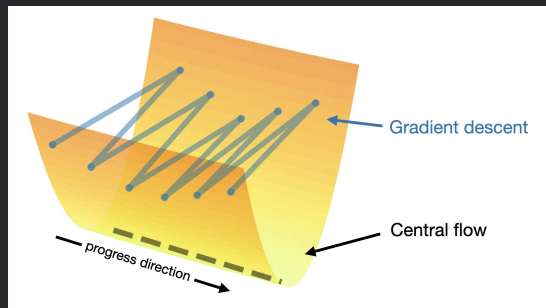
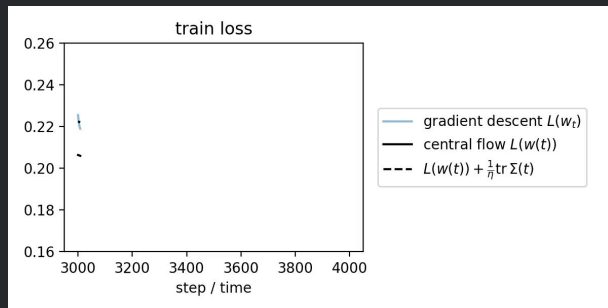
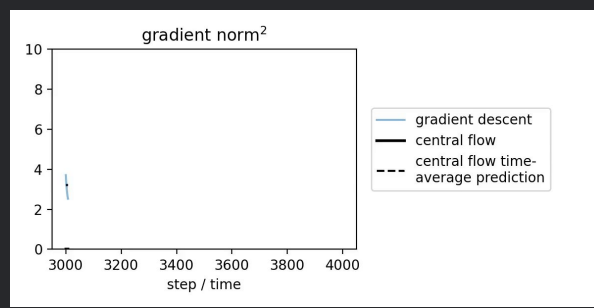
Central Flow

The central flow can further be used to track the variations in individual predictions



River valley predictions

Since the central flow captures the time-averaged dynamics, it intuitively corresponds to the river direction in the river valley mental model, revealing hidden progress made without annealing (and even the loss without annealing)

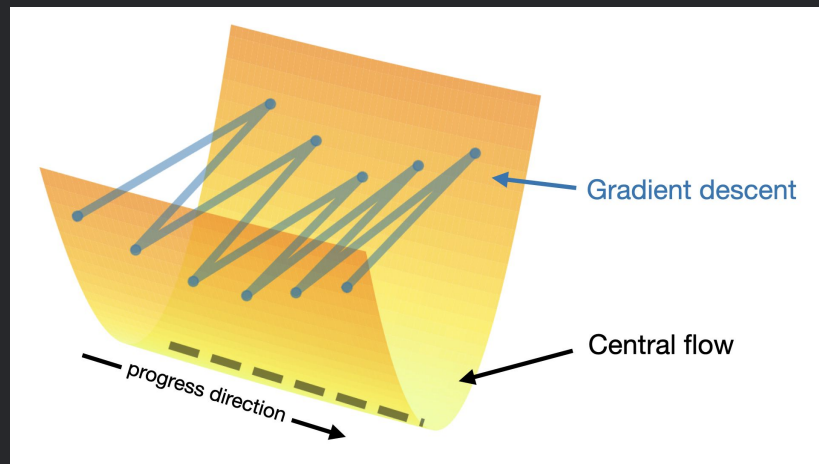


River valley predictions

There are two layers of movement at once

- 1) Oscillations from wall to wall
- 2) Moving vertically from varying sharpness

The first source of movement comes from high sharpness, the second source comes from oscillating sharpness

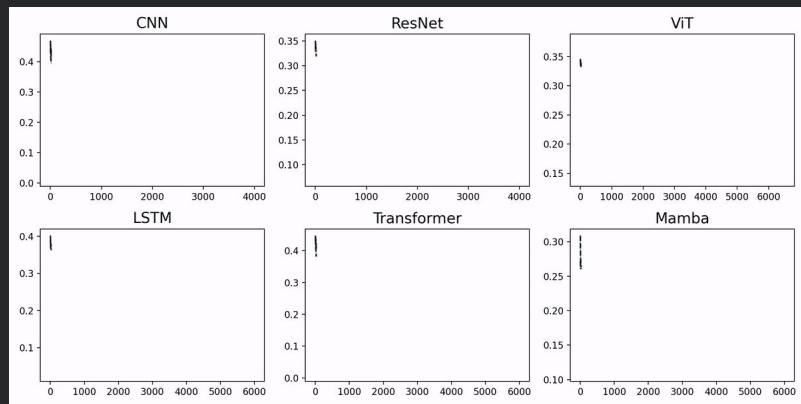


Summary

Quadratics predict that optimization diverges when sharpness exceeds a critical threshold

Modern NNs empirically reduce the sharpness. GD implicitly regularizes sharpness to prevent this from happening

By tracking time-averaged parameters, we can build the Central Flow which makes the correct empirical predictions



Extending EoS to cover modern optimizer modifications

There are three main missing components from EoS

- Momentum
- Minibatch SGD
- Adaptive optimizers

Interaction with momentum

For momentum, we accumulate the gradient over iterations

$$\mathbf{v}_{t+1} = \beta \mathbf{v}_t + \mathbf{g}_t \qquad \mathbf{w}_t = \mathbf{w}_t - \eta \mathbf{v}_{t+1}$$

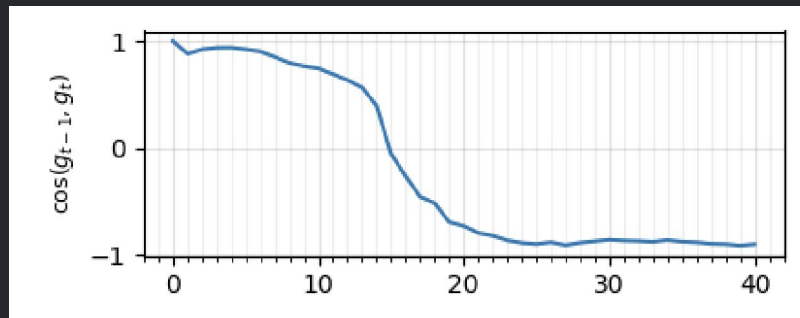
To reason about the sharpness, we need to reason about the maximum stable LR

Interaction with momentum

For quadratics, if we have full batch, the gradients take us from wall to wall. In this regime gradients accumulate like

$$g_t + \beta g_{t-1} + \beta^2 g_{t-2} + \beta^3 g_{t-3} + \dots \approx g - \beta g + \beta^2 g - \beta^3 g + \dots = g / (1 + \beta)$$

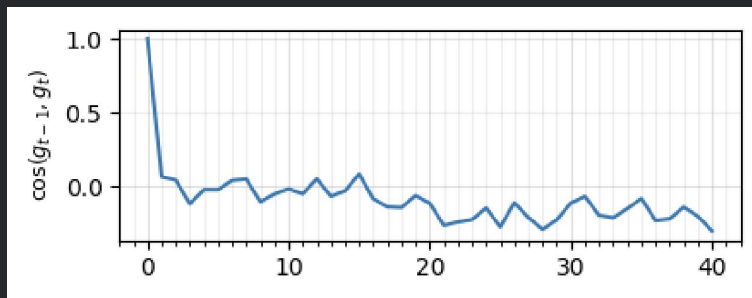
Therefore, for the same learning rate, sharpness increases by a factor of $1 + \beta$



Interaction with batch size

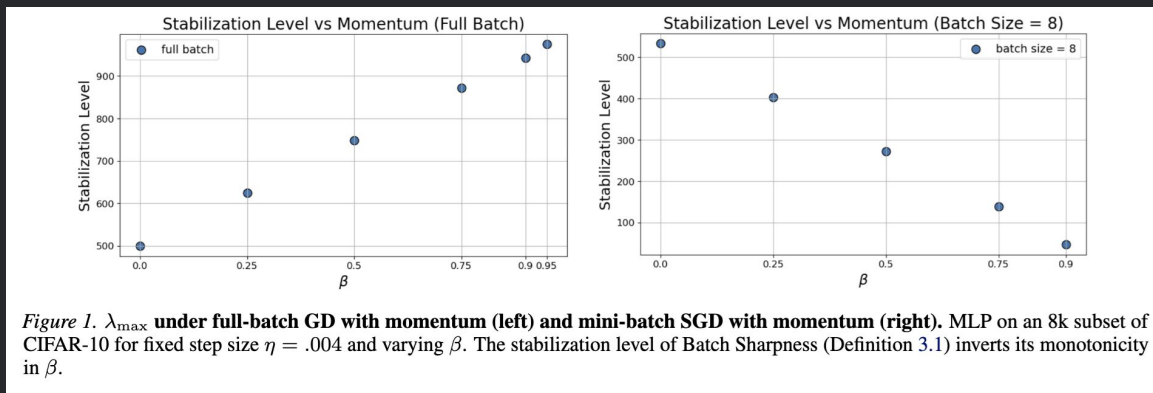
For minibatch, recall from River Valley that the minibatch regime is noise dominated, with gradient directions being close to independent. This results in the gradients accumulating differently

$$\begin{aligned}\text{Var}(\mathbf{g}) &= \text{Var}(\mathbf{g}_t + \beta \mathbf{g}_{t-1} + \beta^2 \mathbf{g}_{t-2} + \beta^3 \mathbf{g}_{t-3} + \dots) \\ &\approx \text{Var}(\mathbf{g}_t) + \text{Var}(\beta \mathbf{g}_{t-1}) + \text{Var}(\beta^2 \mathbf{g}_{t-2}) + \text{Var}(\beta^3 \mathbf{g}_{t-3}) + \dots \\ &= \text{Var}(\mathbf{g}) / (1 - \beta)\end{aligned}$$



Interaction with momentum

Therefore, we have two regimes: large batch where momentum increases sharpness and small batch where momentum decreases sharpness



Refer to [Andreyev et al, 2026](#) for an analysis of momentum and [Song et al, 2026](#) for an analysis of EoS with noise from zeroth order optimization

Scalar RMSProp

Finally, we would like our analysis to extend past SGD

We'll start by studying a simple adaptive optimizer Scalar RMSProp

$$\nu_t = \beta_2 \nu_{t-1} + (1 - \beta_2) \|\nabla L(w_t)\|^2, \quad w_{t+1} = w_t - \frac{\eta}{\sqrt{\nu_t}} \nabla L(w_t).$$

maintain EMA of squared gradient norm take gradient step of size $\eta/\sqrt{\nu}$

This is Adam without momentum and without per-parameter adaptivity. We will see how this update adapts to the loss curvature

Scalar RMSProp

$$\nu_t = \beta_2 \nu_{t-1} + (1 - \beta_2) \|\nabla L(w_t)\|^2, \quad w_{t+1} = w_t - \frac{\eta}{\sqrt{\nu_t}} \nabla L(w_t).$$

maintain EMA of squared gradient norm take gradient step of size $\eta/\sqrt{\nu}$

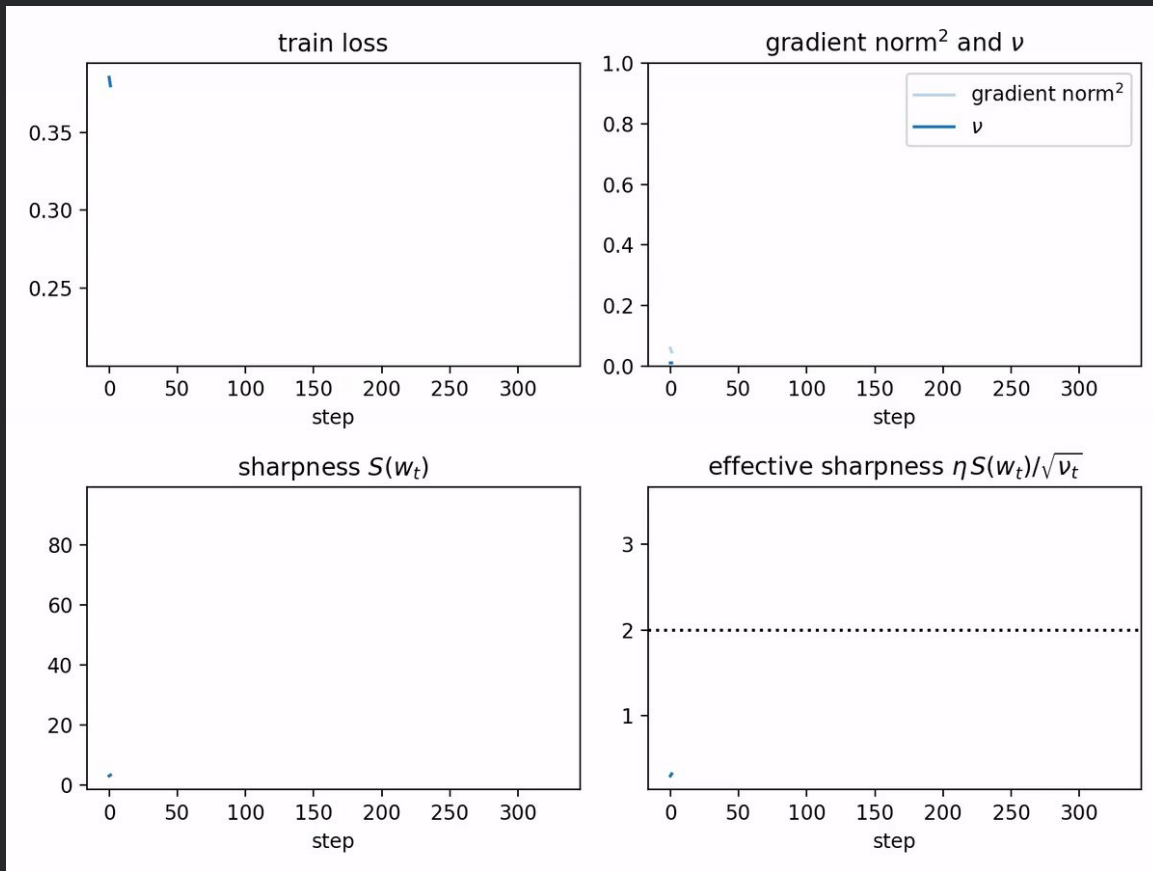
Under Scalar RMSProp, the step size should instead be thought of as $\eta / \sqrt{\nu_t}$

Therefore, we expect oscillation whenever sharpness S exceeds $2\sqrt{\nu_t} / \eta$

This motivates an effective sharpness, $\eta S / \sqrt{\nu_t}$, which should stabilize around 2

Scalar RMSProp

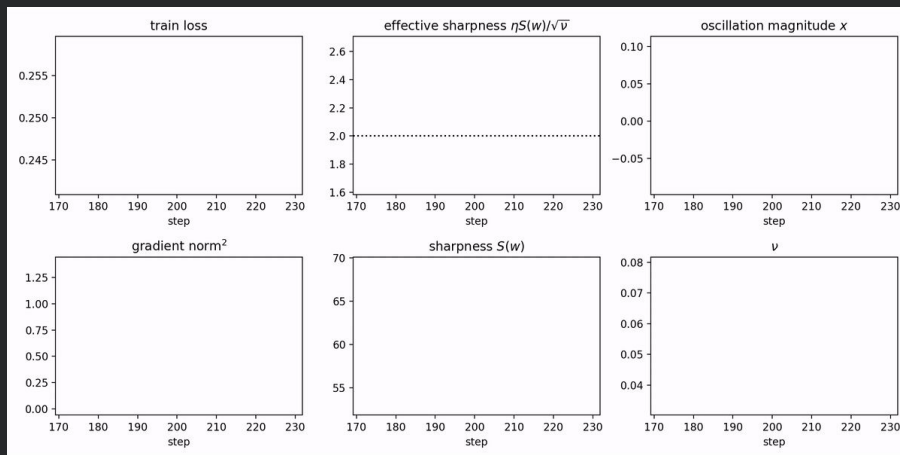
Indeed, the effective sharpness does display the Edge of Stability empirically



Scalar RMSProp

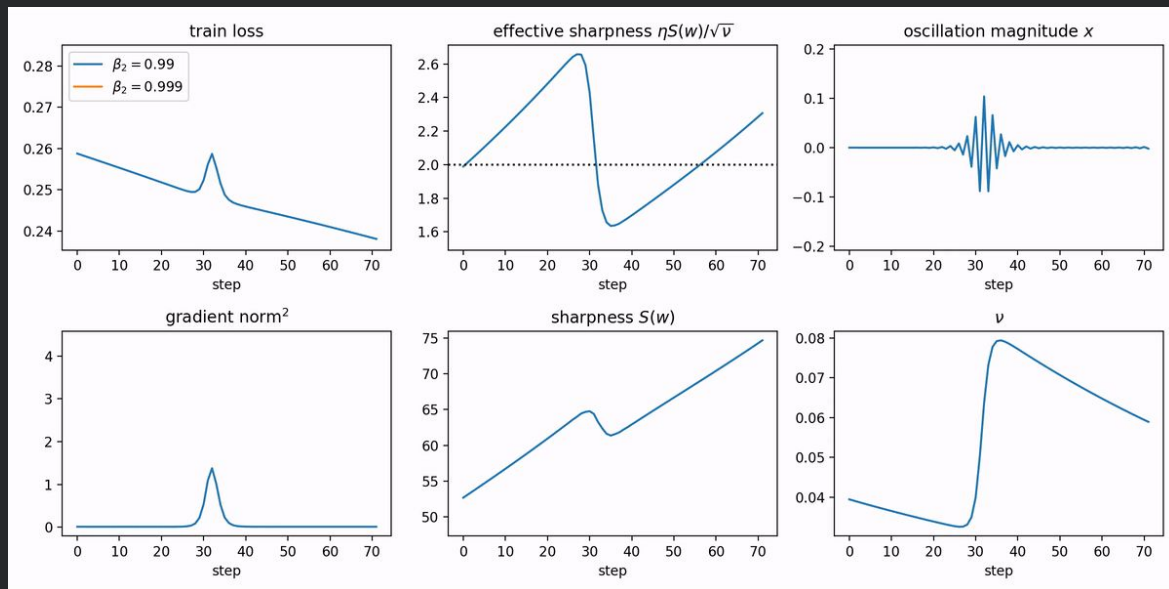
Self-stabilization of effective sharpness in RMSProp happens for two reasons

- 1) Similar to gradient descent, oscillations automatically reduce sharpness
- 2) In addition, oscillations increase the gradient norm, causing v_t to grow, reducing the effective sharpness $\eta / \sqrt{v_t}$ via the denominator (not the numerator)



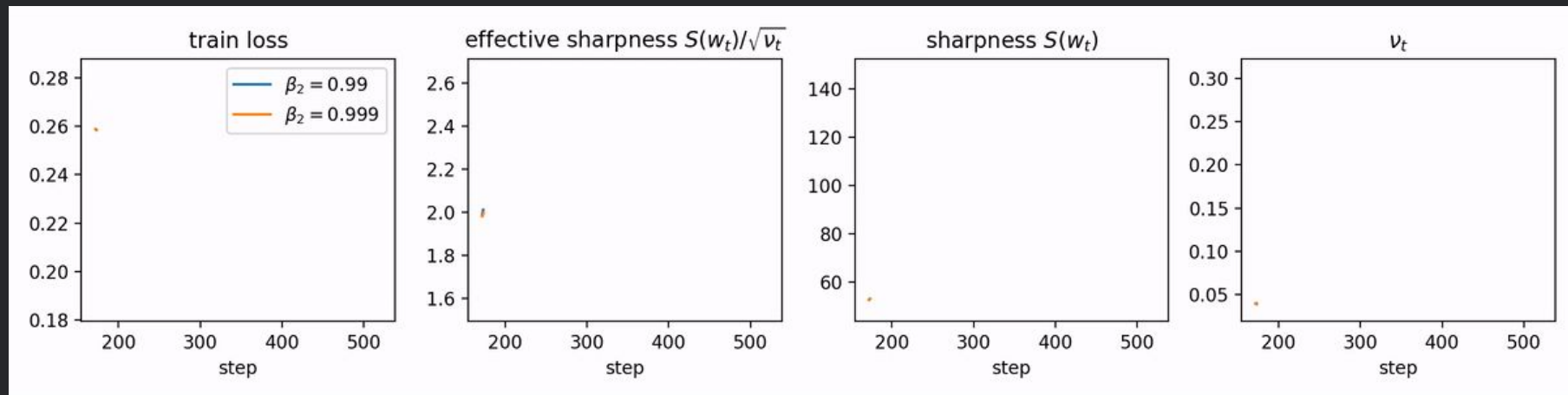
Scalar RMSProp

High β_2 implies that v_t will update slower, self-stabilization will happen slower, and the effective sharpness will come more from sharpness reduction, not v_t growth



Scalar RMSProp

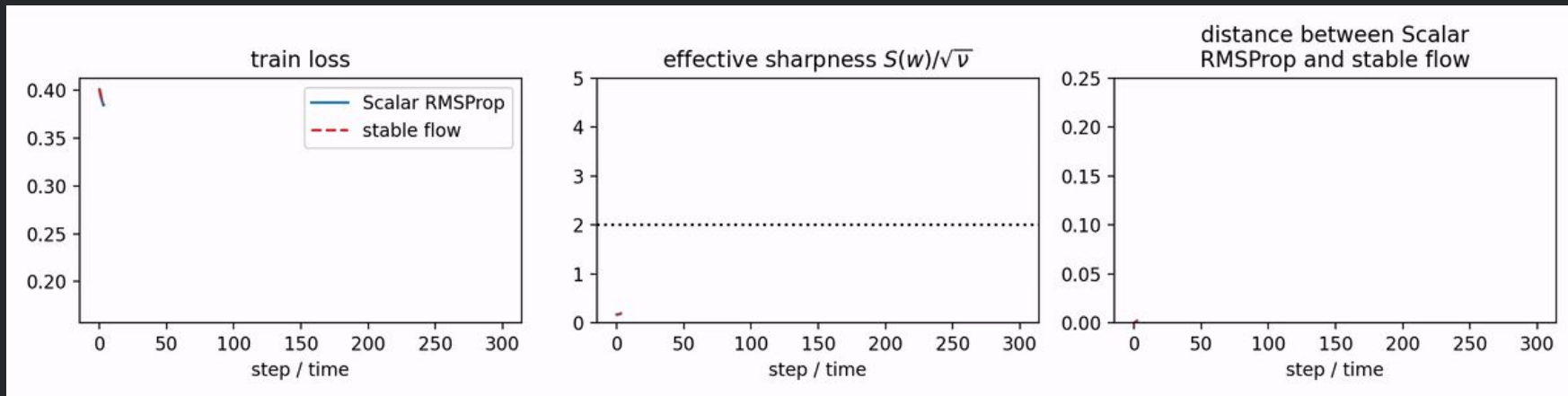
In the long run, high β_2 in the long term will result in lower sharpness and lower v_t for the same effective sharpness



Scalar RMSProp

Similar to gradient flow, we can write down the stable flow for Scalar RMSProp

$$\frac{dw}{dt} = -\frac{\eta}{\sqrt{\nu}} \nabla L(w), \quad \frac{d\nu}{dt} = \frac{1 - \beta_2}{\beta_2} (\|\nabla L(w)\|^2 - \nu).$$



Scalar RMSProp

To derive a central flow for Scalar RMSProp, we can repeat the GD derivation

Time-averaged gradient

$$\underbrace{\mathbb{E}[\nabla L(w_t)]}_{\text{time-averaged gradient}} \approx \underbrace{\nabla L(w(t))}_{\text{gradient at time-averaged iterate}} + \underbrace{\frac{1}{2} \mathbb{E}[x_t^2] \nabla S(w(t))}_{\text{implicit sharpness reduction}}.$$

Approximate gradient norm

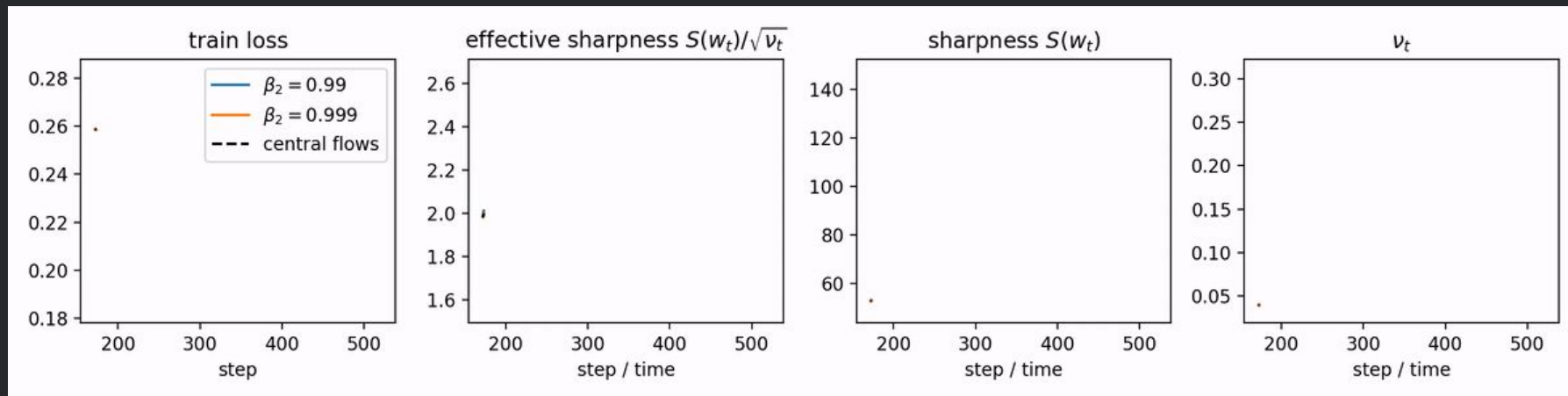
$$\underbrace{\mathbb{E}[\|\nabla L(w_t)\|^2]}_{\text{time-averaged gradient norm}^2} \approx \underbrace{\|\nabla L(w(t))\|^2}_{\text{gradient norm}^2 \text{ at time-averaged iterate}} + \underbrace{\mathbb{E}[x_t^2] S(w(t))^2}_{\text{contribution from oscillations}}.$$

Final prediction
for central flow

$$\frac{dw}{dt} = -\frac{\eta}{\sqrt{\nu}} \underbrace{\left[\nabla L(w) + \frac{1}{2} \sigma^2(t) \nabla S(w) \right]}_{\text{time-averaged gradient}}, \quad \frac{d\nu}{dt} = \frac{1 - \beta_2}{\beta_2} \left[\underbrace{\|\nabla L(w)\|^2 + \sigma^2(t) S(w)^2}_{\text{time-averaged gradient norm}^2} - \nu \right],$$

Scalar RMSProp

Similar to GD, there is a unique choice of variance schedule that works. It makes excellent empirical predictions!



Scalar RMSProp

Over the course of training for Scalar RMSProp, the effective sharpness stays stable at 2, the sharpness grows, and the effective step size decreases

Therefore, scalar RMSProp manages to keep the effective step size automatically balanced at $2 / S$ without explicitly computing the sharpness

This is one precise sense in which RMSProp *adapts* to the loss landscape

Scalar RMSProp

Is this the only benefit of an adaptive optimizer? No, there is an additional effect from sharpness reduction that happens every oscillation

At EoS, v_t stabilizes so we can analyze only how the parameter evolves

$$\frac{dw}{dt} = - \underbrace{\frac{2}{S(w)}}_{\text{adapt step size}} \left[\nabla L(w) + \underbrace{\frac{1}{2} \sigma^2(w; \eta, \beta_2)}_{\text{implicitly reduce sharpness}} \nabla S(w) \right]$$

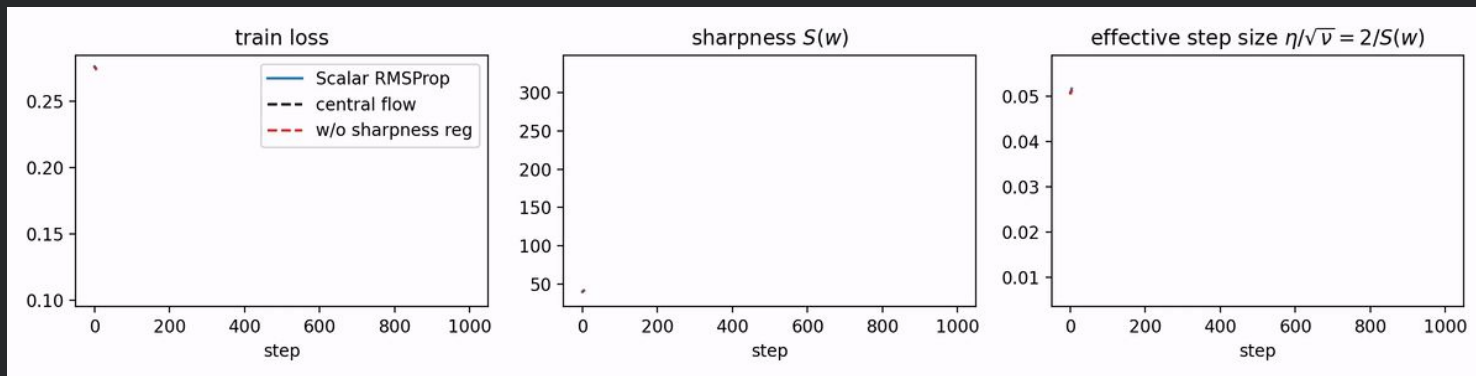
In addition to adapting the step size, we also reduce sharpness. With larger learning rates, we face more oscillation, and result in reduced sharpness

Scalar RMSProp

We can ablate the effects of the sharpness regularization by simulating a flow that's missing the sharpness regularization term. Note how Scalar RMSProp reduces the sharpness, increases effective step size, and accelerates convergence

$$\frac{dw}{dt} = - \underbrace{\frac{2}{S(w)}}_{\text{adapt step size}} [\nabla L(w)],$$

$$\frac{dw}{dt} = - \underbrace{\frac{2}{S(w)}}_{\text{adapt step size}} \left[\nabla L(w) + \underbrace{\frac{1}{2} \sigma^2(w; \eta, \beta_2) \nabla S(w)}_{\text{implicitly reduce sharpness}} \right]$$



Scalar RMSProp

One common confusion is why the learning rate in an adaptive optimizer is meaningful since it gets rescaled (for example, at EoS, the effective step size is fixed at $2 / S$)

We now see that larger learning rate results in higher sharpness regularization, increasing effective step size. Therefore, increasing learning rate does increase step size, but quite indirectly

RMSProp

Now that we understand Scalar RMSProp, we can extend our analysis to RMSProp which maintains a coordinate-wise squared gradient EMA. This is equivalent to Adam without the momentum

$$\nu_t = \beta_2 \nu_{t-1} + (1 - \beta_2) \nabla L(w_t)^{\odot 2}, \quad w_{t+1} = w_t - \frac{\eta}{\sqrt{\nu_t}} \odot \nabla L(w_t).$$

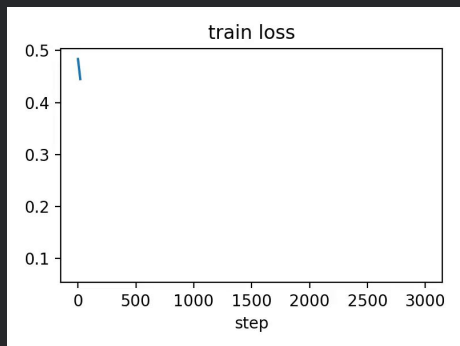
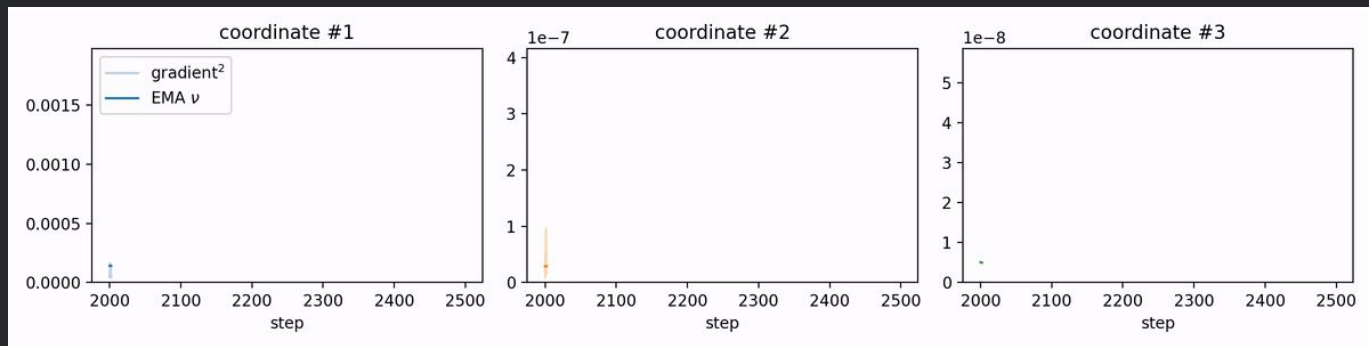
update EMA of squared gradient take preconditioned gradient step

This is equivalent to diagonal preconditioning of $\text{diag}(\sqrt{\nu_t} / \eta)$

Though people have an intuition that Adam/RMSProp helps similar to Newton's method, it's worth noting that they are purely first-order methods

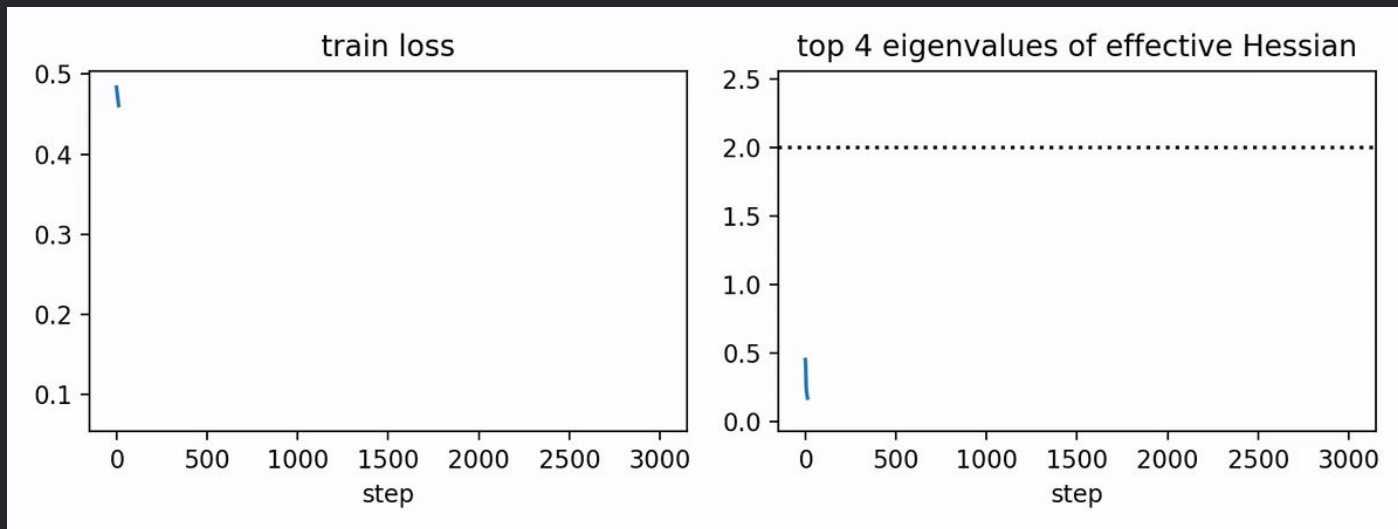
RMSProp

RMSProp has a smooth loss curve. However, its coordinates oscillate quite a bit



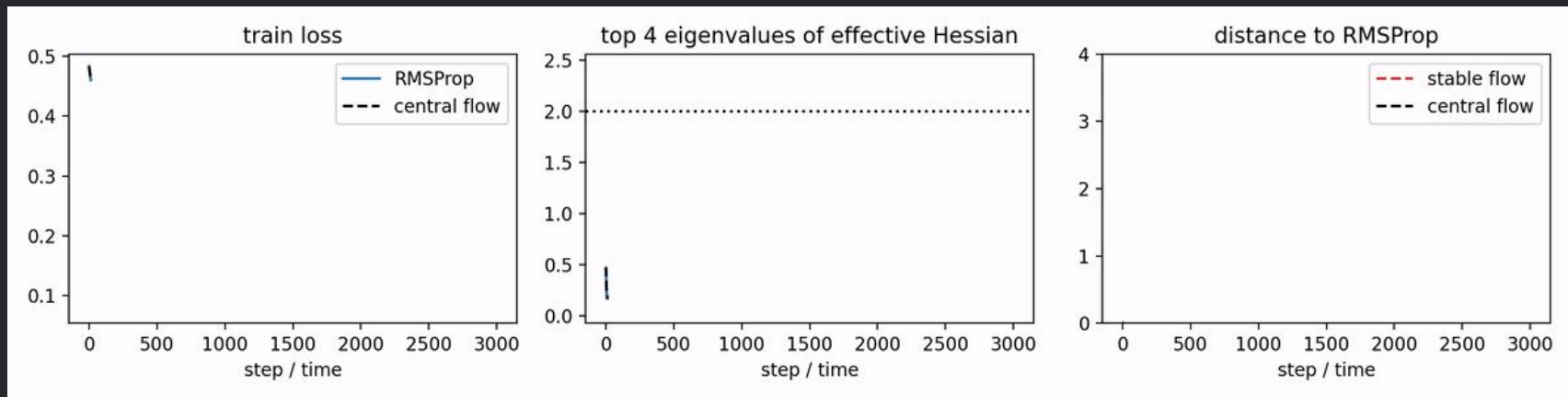
RMSProp

Since we are reasoning about preconditioned gradient descent, it's most helpful to think about the effective Hessian $P^{-1}H = \text{diag}(\eta / \sqrt{v_t})H$ and whether its eigenvalues exceed 2. We see the familiar EoS



RMSProp

For RMSProp, you can also derive a similar-looking central flow for all the eigenvalues. It unfortunately does not have a simple closed form, we'll skip the math. As expected, it makes the right predictions and such



RMSProp

So, what is RMSProp's preconditioner actually computing? We can reason about the stationary preconditioner, equivalent to

$$\overline{P}(w) := \frac{\sqrt{\overline{v}(w)}}{\eta}$$

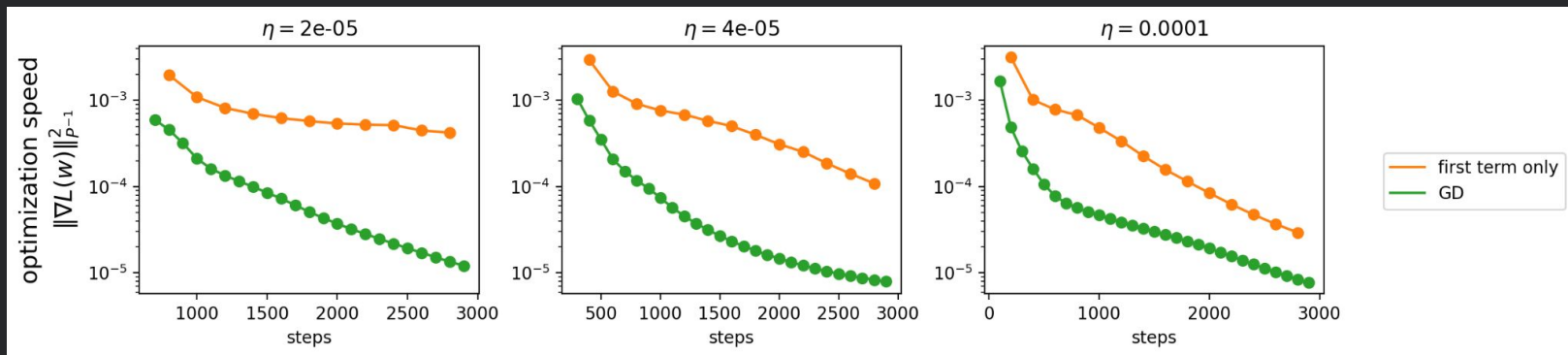
This is equivalent to solving the following convex program

$$\overline{P}(w) = \underset{P \text{ diagonal, } P \succeq 0}{\operatorname{argmin}} \quad \operatorname{trace}(P) + \underbrace{\frac{1}{\eta^2} \|\nabla L(w)\|_{P^{-1}}^2}_{\text{optimization speed}} \quad \text{such that} \quad \underbrace{H(w) \preceq 2P}_{\text{local stability}}.$$

RMSProp

$$\bar{P}(w) = \underset{P \text{ diagonal, } P \succeq 0}{\operatorname{argmin}} \quad \operatorname{trace}(P) + \underbrace{\frac{1}{\eta^2} \|\nabla L(w)\|_{P^{-1}}^2}_{\text{optimization speed}} \quad \text{such that} \quad \underbrace{H(w) \preceq 2P}_{\text{local stability}}$$

The constraint enforces the stable effective step size. $\operatorname{trace}(P)$ corresponds to minimizing the sum of the inverse effective learning rates (i.e. increasing step size subject to stability). Speeds up training

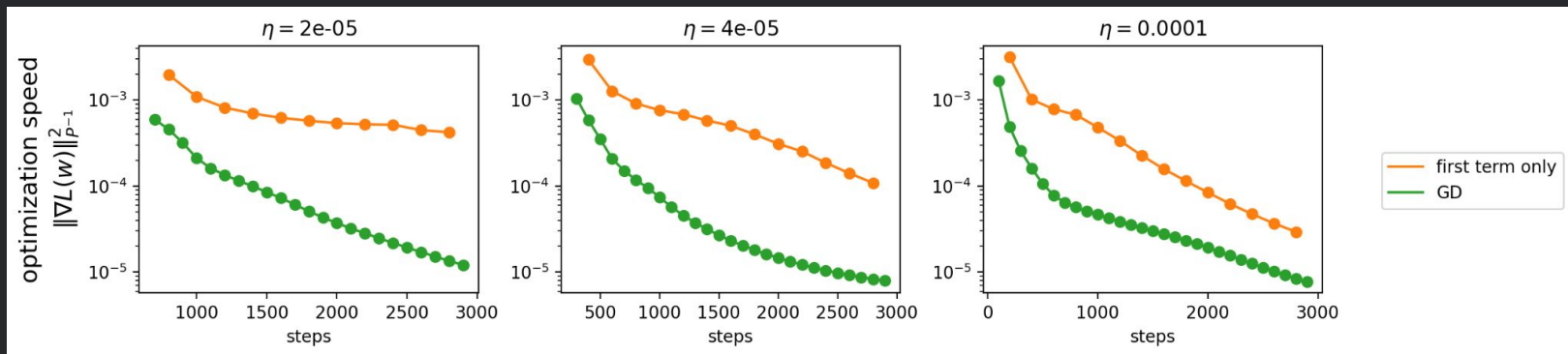


RMSProp

$$\bar{P}(w) = \underset{P \text{ diagonal, } P \succeq 0}{\operatorname{argmin}} \quad \operatorname{trace}(P) + \underbrace{\frac{1}{\eta^2} \|\nabla L(w)\|_{P^{-1}}^2}_{\text{optimization speed}} \quad \text{such that} \quad \underbrace{H(w) \preceq 2P}_{\text{local stability}}$$

Interestingly, only optimizing the first term without the diagonal constraint is equivalent to preconditioning via the Hessian

$$\hat{P}(w) = \frac{1}{2} H(w).$$

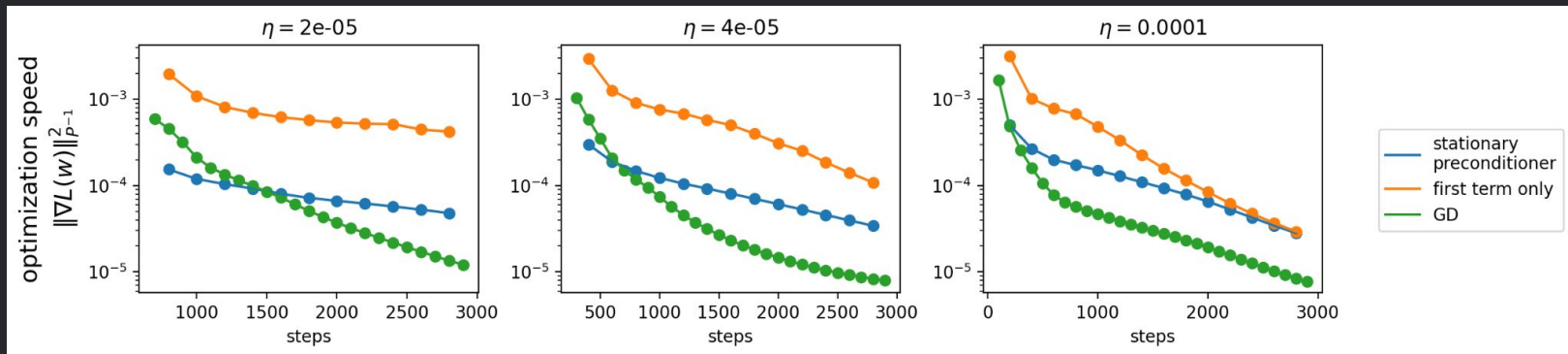


RMSProp

$$\bar{P}(w) = \underset{P \text{ diagonal, } P \succeq 0}{\operatorname{argmin}} \quad \operatorname{trace}(P) + \underbrace{\frac{1}{\eta^2} \|\nabla L(w)\|_{P^{-1}}^2}_{\text{optimization speed}} \quad \text{such that} \quad \underbrace{H(w) \preceq 2P}_{\text{local stability}}$$

On the other hand, the second term exists to directly slow down training. When introducing this second term into the central flow, training proceeds more slowly

This might be a suboptimal aspect of RMSProp, even though it is what it does



Summary

When normalizing the gradient (i.e. via RMSProp), we change the effective step size and the effective sharpness

Training via (Scalar) RMSProp both adapts to the curvature of the loss landscape while also reducing the sharpness. (Scalar) RMSProp accelerates training because of the adaptive component and its effective sharpness reduction

The stationary preconditioner of RMSProp reveals the similarities and differences between second-order methods

Failures of Edge of Stability / Central Flows

Generally less reliable phenomenology for

- Cross-entropy loss
- Stochastic setting
- Modern optimizers such as AdamW
- Extremely high learning rates

Furthermore, it is mostly unknown why progressive sharpening happens

Your assignment

In your assignment, you will be analyzing where sharpness and effective sharpness stabilize as a function of different hyperparameters (learning rate, batch size, momentum, etc.).

It will be most helpful to first process EoS/Central Flows and what predictions they make. Then, you can run your experiments and see where the models made correct and incorrect predictions.

Why do we care about sharpness?

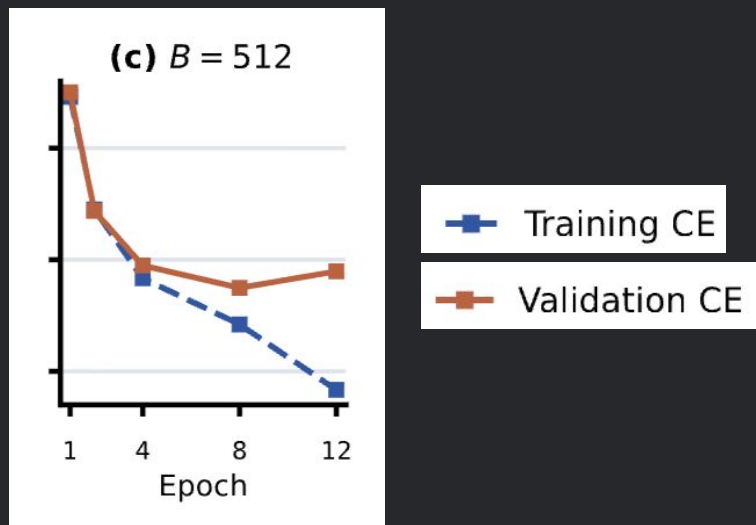
We've now reasoned about how decreasing sharpness can accelerate training from the perspective of increasing step size

Is there any other reason we should care about sharpness?

Generalization

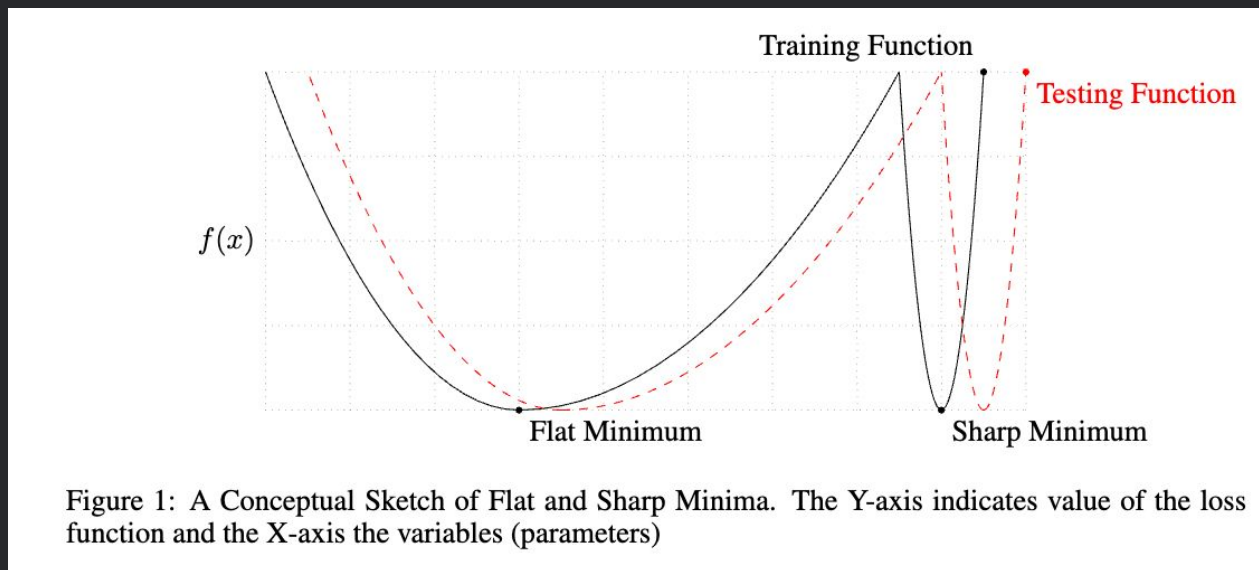
So far in this class, we've considered model performance in the context of unlimited data for a fixed number of steps. However, we may be constrained by the number of samples rather than the number of steps

In this setting, train loss and validation loss are not correlated. Even though we can minimize train loss quite aggressively by memorizing samples, what we care about is loss on the validation set.



Generalization

There is a folklore intuition that flatter minima will display better generalization
[[Hochreiter and Schmidhuber, 1997](#), [Dziugaite and Roy, 2017](#)]



Fantastic Generalization

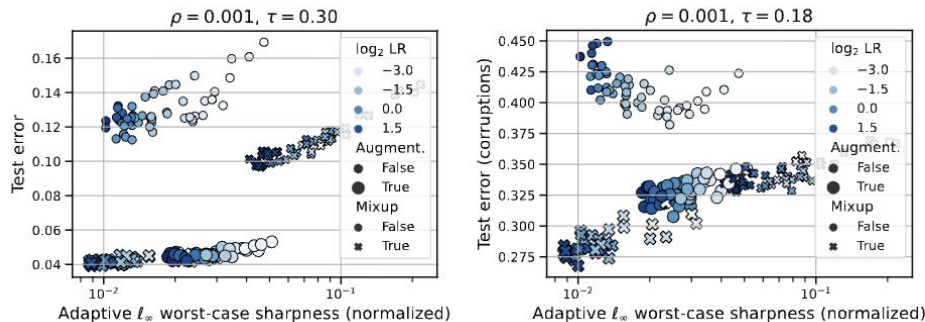
[Liang et al, 2019](#) studies how well various generalization measures correlate with model performance over a sweep of 2187 models. Overall, they find that most measures correlate poorly, with sharpness being a promising contender

$1/\alpha'$ sharpness mag	0.516
sum-of-fro/margin	0.381
pacbayses-orig	0.360
path-norm/margin	0.305
grad-noise-final	0.292
fro/spec	0.243
cross-entropy	0.147
# params	-0.15

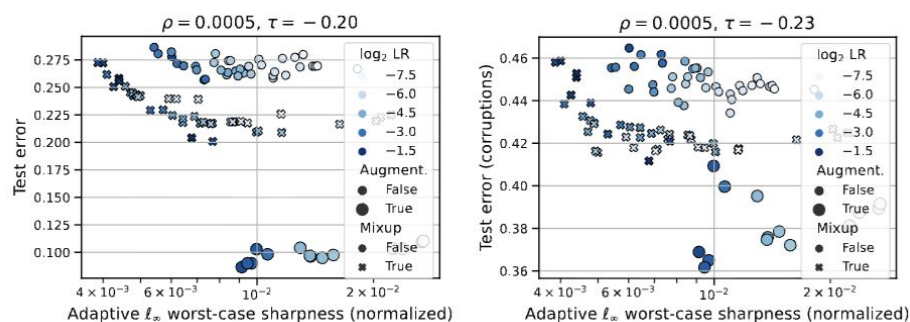
Fantastic Generalization?

However, later studies find that this correlation is actually quite weak and often negative as well [[Andriushchenko et al, 2023](#)], Furthermore, you can construct examples where the global minima are extremely sharp [[Dinh et al, 2017](#)]

ResNets-18 with logit normalization

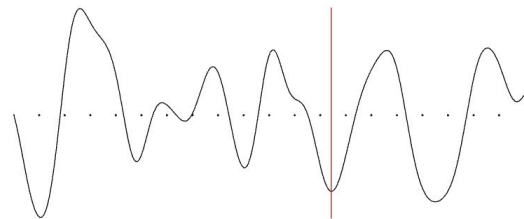


ViTs with logit normalization

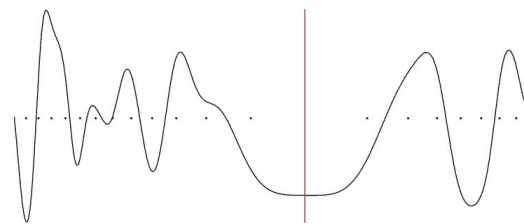


Fantastic Generalization?

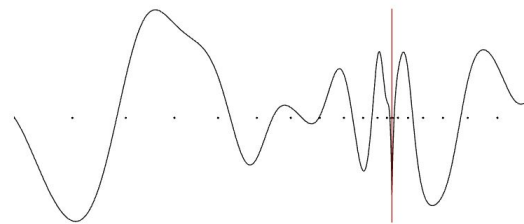
You can construct examples where the global minima are extremely sharp [[Dinh et al, 2017](#)]. One general failure of sharpness is that is not invariant to your parameterization.



(a) Loss function with default parametrization



(b) Loss function with reparametrization



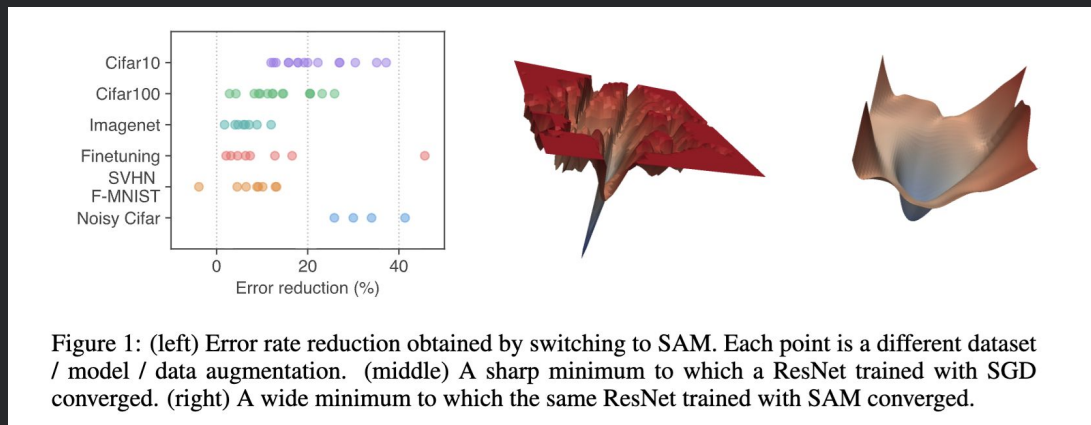
(c) Loss function with another reparametrization

Sharpness Aware Minimization

Suppose believed sharpness was good for training speed and generalization. Can we optimize for it?

We know of a few indirect knobs: high learning rate, low batch size

[Foret et al, 2020](#) proposed Sharpness-Aware Minimization (SAM) to directly optimize sharpness



Sharpness Aware Minimization

Core intuition: instead of seeking weights that have low loss, seek out minima where the entire region surrounding the weights also have low loss

Operationalized as minimize the loss under a bounded adversarial update around the weights

$$\min_{\mathbf{w}} L_S^{SAM}(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2 \quad \text{where} \quad L_S^{SAM}(\mathbf{w}) \triangleq \max_{\|\boldsymbol{\epsilon}\|_p \leq \rho} L_S(\mathbf{w} + \boldsymbol{\epsilon}),$$

In practice, the adversary is estimated via a gradient

Sharpness Aware Minimization

Original paper reports improvements in generalization over standard SGD (at the cost of some extra compute)

Model	Augmentation	CIFAR-10		CIFAR-100	
		SAM	SGD	SAM	SGD
WRN-28-10 (200 epochs)	Basic	2.7 ± 0.1	3.5 ± 0.1	16.5 ± 0.2	18.8 ± 0.2
WRN-28-10 (200 epochs)	Cutout	2.3 ± 0.1	2.6 ± 0.1	14.9 ± 0.2	16.9 ± 0.1
WRN-28-10 (200 epochs)	AA	2.1 $\pm <0.1$	2.3 ± 0.1	13.6 ± 0.2	15.8 ± 0.2

Empirically, it is known to work best under label noise (analyzed in [Baek et al, 2024](#))

Method	Noise rate (%)			
	20	40	60	80
Sanchez et al. (2019)	94.0	92.8	90.3	74.1
Zhang & Sabuncu (2018)	89.7	87.6	82.7	67.9
Lee et al. (2019)	87.1	81.8	75.4	-
Chen et al. (2019)	89.7	-	-	52.3
Huang et al. (2019)	92.6	90.3	43.4	-
MentorNet (2017)	92.0	91.2	74.2	60.0
Mixup (2017)	94.0	91.5	86.8	76.9
MentorMix (2019)	95.6	94.2	91.3	81.0
SGD	84.8	68.8	48.2	26.2
Mixup	93.0	90.0	83.8	70.2
Bootstrap + Mixup	93.3	92.0	87.6	72.0
SAM	95.1	93.4	90.5	77.9
Bootstrap + SAM	95.4	94.2	91.8	79.9

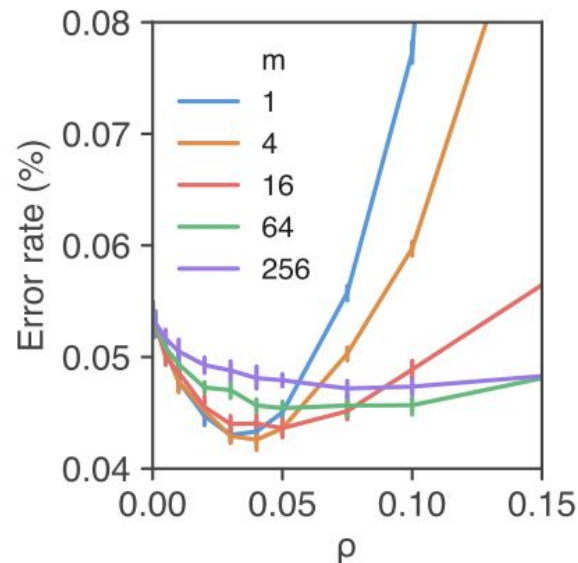
Table 4: Test accuracy on the clean test set for models trained on CIFAR-10 with noisy labels. Lower block is our implementation, upper block gives scores from the literature, per [Jiang et al. \(2019\)](#).

Sharpness Aware Minimization

Original SAM paper found that SAM works better with a smaller batch size (denoted by m on the right)

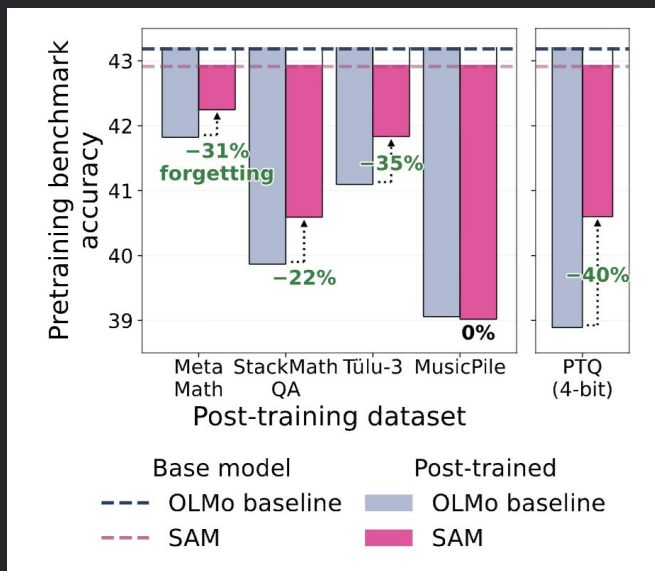
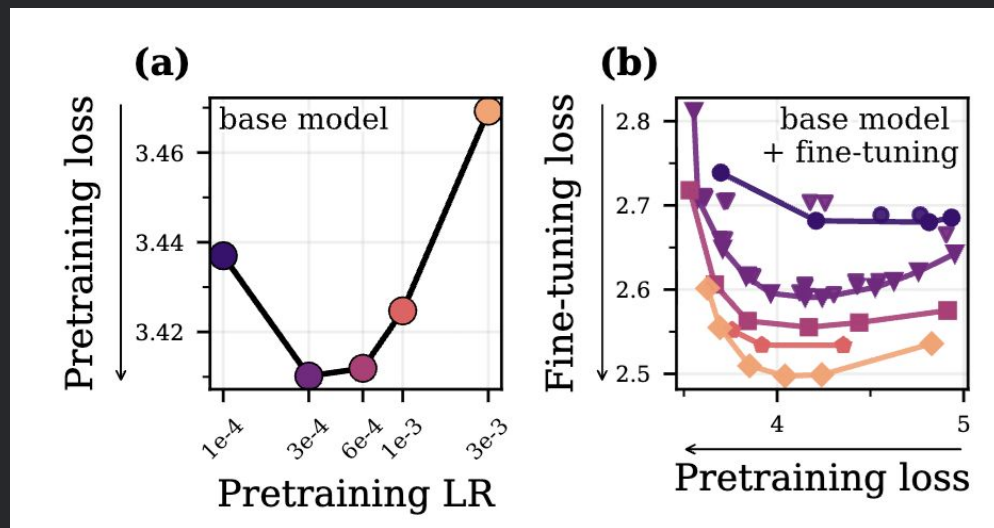
Later it is theoretically proved that the sharpness SAM optimize for depends on batch size. At batch size 1, SAM actually minimizes a different version of sharpness, called the average case sharpness.

$$\min_w L_S^{\text{avg}}(w) + \lambda \|w\|_2^2, \quad L_S^{\text{avg}}(w) \triangleq \mathbb{E}_{\epsilon \sim \text{Unif}(B_\rho(\rho))} [L_S(w + \epsilon)]$$



Application: Sharpness Aware Pretraining

One intuition is that flatter minima provide better initializations for fine-tuning. Therefore, seeking flatter minima (either via SAM or higher learning rates) can lead to pre-trained initializations that are more amenable to fine-tuning



Sharpness Testimonials

“I feel like I should’ve been nerd sniped by this for at most 6 months of first year of grad school 😂 not 2 years”

“I feel like I should have been nerd sniped for my whole PhD. Sharpness is the only interesting problem in ML”

“I realize sharpness is not the solution to generalization before I enter my PhD and stop working on it, I am still glad that I make this choice”

“Sharpness is super interesting. It’s just that at some point, I realized it’s not useful.”

Summary

Sharpness, in addition to accelerating convergence, may or may not improve generalization when limited by samples

We can try to directly optimize for sharpness, though it's unclear what we are actually optimizing

Sharpness can serve as a helpful mental model for algorithm design in components such as fine-tuning