universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Edge of Stability for Neural Networks: Extending Current Research to Stochastic Adaptive Optimizers

verfasst von | submitted by

Jakob Fabian Rohner BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2026

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 821
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Mathematik
Betreut von | Supervisor:	Assoz. Prof. Dr. Philipp Christian Petersen M.Sc.

Abstract

Die Trainingsdynamik moderner neuronaler Netze wird stark durch die Phänomene des progressive sharpening und der Edge of Stability (EoS) beeinflusst. Diese Arbeit bietet einen umfassenden Überblick der EoS für ein breites Spektrum von Optimierungsalgorithmen sowohl deterministischer als auch stochastischer Natur. Sowohl die genauen Bedingungen, unter denen diese Phänomene auftreten, als auch die dahinterstehenden Mechanismen werden untersucht. Während die Stabilitätsschwellen sowohl für stochastische Methoden wie SGD als auch für adaptive Methoden wie Adam im full batch Fall in der aktuellen Literatur gut dokumentiert sind, bleiben die Stabilisierungsmechanismen von stochastischen adaptiven Optimierern wie mini-batch Adam, ein sehr beliebter Optimierungsalgorithmus für reale neuronale Netze, weitgehend unerforscht.

Diese Arbeit stellt heuristisch neue Instabilitätskriterien für diese stochastischen adaptiven Methoden auf: die effective batch sharpness für scalar RMSProp und standard RMSProp sowie die momentum effective batch sharpness für Adam. Aufbauend auf diesen neu etablierten Metriken präsentieren wir einen rigorosen Beweis: Überschreitet Adams modified batch sharpness den theoretischen Schwellenwert der maximum stable sharpness, wächst die erwartete Root-Mean-Square-Norm des Momentumvektors exponentiell an und löst eine lokale Instabilität sowie einen Katapulteffekt aus.

Die in dieser Arbeit durchgeführten numerischen Experimente zeigen, dass für scalar RMSProp und standard RMSProp die gefundenen sharpness Werte wachsen, bis sie die prognostizierte Stabilitätsschwelle erreichen. Die effective batch sharpness ist somit ein valides Instabilitäötskriterion für scalar RMSProp und standard RMSProp. Adam verhält sich in den Experimenten grundlegend anders. Adams interne Mechanismen unterdrücken die Bewegung der Parameter entlang der Eigenvektoren mit der höchsten sharpness. Dadurch bleibt die momentum effective batch sharpness für Adam weit unterhalb der von uns abgeleiteten theoretischen Instabilitätsschwelle.

Abstract

The training dynamics of modern neural networks are heavily influenced by the phenomena of progressive sharpening and the Edge of Stability (EoS). This thesis provides a comprehensive overview of the EoS across a wide spectrum of optimization algorithms of both deterministic and stochastic nature. Both the precise conditions under which these phenomena occur as well as the mechanisms behind them are explored. While the stability thresholds for both stochastic methods like SGD and adaptive methods like Adam in the full batch case are well documented in current literature, the stabilization mechanisms of stochastic adaptive optimizers like mini-batch Adam, which is a very popular optimizer for real world neural networks, remain largely unexplored.

This work heuristically derives new instability criteria for these stochastic adaptive methods: the effective batch sharpness for scalar RMSProp and standard RMSProp, and the momentum effective batch sharpness for Adam. Building upon these newly established metrics, we present a rigorous theoretical proof demonstrating that if Adam's modified batch sharpness exceeds the theoretical maximum stable sharpness threshold, the expected root-mean-square norm of the momentum vector will grow exponentially, triggering local instability and a catapult effect.

The numerical experiments conducted in this thesis demonstrate that for scalar RMSProp and standard RMSProp, the found sharpness values grow until they reach the projected stability threshold, making them a valid instability criterion. Adam behaves fundamentally differently in the experiments. It's internal mechanics suppress its movement along the sharpest eigenvectors, allowing it to consistently plateau well below the theoretical instability threshold we derived.

Acknowledgments

I would like to thank my supervisor, Philipp Petersen, for his guidance in shaping my research topic and for his continuous support during my thesis. I also want to thank Arseniy Andreyev for answering many fo my questions during our email correspondence, as well as Pierfrancesco Beneventano and Arseniy for providing their code on the Edge of Stochastic Stability for SGD. I am also very grateful for the support of my family, which made my studies possible. Finally, I wish to thank all of my friends in Vienna, who greatly supported me in my studies and made my time during my Master's program wonderful.

Contents

1 Introduction **6**

2 Introduction **6**

3 Notation **8**

4 Neural Networks **11**
4.1 The Multilayer Perceptron 11
4.2 Other Neural Networks 12

5 Optimization of Neural Networks **13**
5.1 Gradient Descent 13
5.2 Stochastic Gradient Descent 14
5.3 Momentum Based Optimizers (Heavy Ball, Nesterov) 16
5.4 Adaptive Gradient Methods 17

6 Generalization and Sharpness **19**
6.1 Sharpness of a Neural Network 20
6.2 Relation Between Sharpness and Generalization 22

7 Progressive Sharpening **23**
7.1 Descent Lemma and Sharpness of Quadratic Function 23
7.2 Gradient Descent on Quadratic Functions 23
7.3 Progressive Sharpening 24

8 The Edge of Stability **25**
8.1 Maximum Stable Sharpness 25
8.2 Sharpness at the Edge of Stability 25
8.3 Training Loss at the Edge of Stability 26
8.4 Edge of Stability for Different Neural Network Architectures 26
8.5 Edge of Stability for Different Loss Functions 26

9 Self-Stabilization **27**
9.1 Self-Stabilization Property 28
9.2 Constrained Trajectory 29
9.3 The Four Stages 30
9.4 Formalizing the Stabilizing Mechanism 34

10 Modeling Gradient Descent Behavior with Differential Equations **38**
10.1 Gradient Flow 38
10.2 Central Flow 39
10.3 Multiple Unstable Eigenvalues 42
10.4 Time-Averaged Trajectories of Arbitrary Functions 43

11 Edge of Stability for Adaptive Gradient Methods **45**
11.1 Edge of Stability for Scalar RMSProp 46
11.2 Edge of Stability for RMSProp 47

11.3 Edge of Stability for Momentum Based Optimizers 50
11.4 Edge of Stability for Adam 50

12 Interaction-Aware Sharpness **51**

13 SGD: Progressive Sharpening and Edge of Stability **52**
13.1 Noise-Driven and Curvature-Driven Oscillations 53
13.2 Type-1 Oscillations 54
13.3 Instability Criterions 56
13.4 Batch Sharpness 58
13.5 Stability 61

14 Batch Sharpness for Adaptive Optimizers **68**
14.1 Batch Sharpness for Scalar RMSProp 68
14.2 Batch Sharpness for RMSProp 69
14.3 Batch Sharpness for Adam 70
14.4 Proof of Divergence of Adam if Batch Sharpness Exceeds the MSS 72

15 Numerical Experiments **76**
15.1 Scalar RMSProp 77
15.2 RMSProp 77
15.3 Adam 80

16 Conclusion **82**

Bibliography **83**

1 Introduction

2 Introduction

Over the last decade, neural networks have become the most influential and widely used machine learning technique. At the core of training these models lies the fundamental task of mathematical optimization. The algorithm needs to approximate an unknown underlying function by finding a set of parameters θ that minimizes a specific loss function over a given dataset. The theoretical mechanisms behind the trajectory of many common optimization algorithms remain a highly active area of research, especially in the non-convex loss landscapes of modern neural network architectures.

A central question in modern deep learning is how neural networks manage to generalize well to unseen data. The generalization capabilities of a model have been heuristically linked to the "flatness" of the local minimum found during training [11]. A measure of this flatness of the loss landscape at a given point in parameter space is the "sharpness", where low sharpness corresponds to a flat minimum. Sharpness is formally defined by the curvature of the loss surface, governed by the largest eigenvalue of the loss function's Hessian matrix [17].

This thesis explores the optimization phenomenon known as progressive sharpening [12, 6]. When training with gradient descent, the sharpness tends to increase during the early phases of training until it hits the maximum stable sharpness (MSS), a critical threshold. The MSS is determined by the optimizer's step size η. Classical optimization theory on quadratic functions tells us that surpassing this threshold should cause the algorithm to diverge instead of converging to a local minimum. However, in neural network optimization, the algorithm instead enters the Edge of Stability (EoS) regime [6]. At the EoS, progressive sharpening abruptly halts, and the sharpness hovers just above the MSS threshold while the loss continues to decrease over longer training periods.

This thesis investigates the underlying mechanisms that lead to this behavior. It focuses on the explanations of self-stabilization [7] and the central flow framework [4]. Self-stabilization relies on the third-order Taylor expansion of the loss function, showing that at the EoS, the algorithm actively steers itself away from directions of steepest sharpness increase. The central flow framework extends this by modeling the time-averaged trajectory of the parameters, which filters out the chaotic, high-frequency oscillations that occur at the EoS. This mathematically demonstrates that the underlying trajectory actively seeks flatter regions of the loss landscape, while formally proving that the loss consistently decreases over longer training periods.

While these phenomena are well-documented for standard gradient descent, modern deep learning almost exclusively uses mini-batch SGD and adaptive gradient methods (such as RMSProp and Adam). Recent works have begun to generalize the EoS analysis to full-batch adaptive methods [5] and to mini-batch SGD [2]. A primary focus of this thesis is to further investigate progressive sharpening and the EoS for the setting of stochastic adaptive optimizers. It derives

new batch sharpness criteria for mini-batch RMSProp and mini-batch Adam to evaluate their stability thresholds.

This thesis is organized as follows: Section 2 establishes the mathematical notation, while Sections 3 and 4 review foundational neural network architectures and standard optimization algorithms. Sections 5 through 7 formally define sharpness, explain its link to generalization and detail the empirical phenomena of progressive sharpening and the Edge of Stability for full-batch gradient descent. Sections 8 and 9 explore possible explanations for these optimization dynamics by introducing the theoretical frameworks of self-stabilization and central flows. Sections 10 and 11 extend this EoS analysis to adaptive methods and introduce a different sharpness measure called the Interaction-Aware Sharpness. Section 12 transitions to the mini-batch setting, detailing the Edge of Stochastic Stability (EoSS) and establishing batch sharpness as the primary instability criterion for SGD. Finally, Sections 13 and 14 present the primary novel contributions of this thesis. Section 13 heuristically derives new batch sharpness criteria and instability proofs for stochastic adaptive optimizers (scalar RMSProp, standard RMSProp, and Adam). Section 14 validates these theoretical derivations through rigorous numerical experiments.

In the preparation of this thesis, AI applications (specifically ChatGPT, Gemini Flash, and Gemini Pro) were utilized as tools for coding assistance, text editing, proofreading, and LaTeX formatting. All AI-generated output was thoroughly reviewed, verified, and revised by the author.

3 Notation

Symbol	Description
(X, YY)	input space X and output space Y
$M = \{(x_{\text{in},k}, y_{\text{out},k})\}_{k=1}^{m}$	m data points consisting of input $x_{\text{in},k}$ and output $y_{\text{out},k}$
σ	activation function
W	weight matrix
b	bias vector
Φ	multilayer perceptron (MLP)
$\text{vec}(\cdot)$	row by row vectorization
$\theta \in \Theta$	parameters of neural networks in a parameter space
p	number of parameters / length of θ
h_k	hidden state
$\mathcal{L}(x_{\text{in}}, y_{\text{out}})$	loss function for a single data point
$L(\theta)$	loss function for multiple data points, acting on the parameter space
$L_k(\theta)$	loss function for a single data point with index k, acting on the parameter space
η	learning rate / step size of the optimizer
$\mathcal{B}_i$	batch with index i
$L_{\mathcal{B}_i}(\theta)$	loss function for all data point of a batch, acting on the parameter space
m_t	momentum vector (EMA of gradients)
ν_t	EMA of squared gradients (update vector in adaptive methods)
$\hat{m}_t, \hat{v}_t$	bias-corrected momentum and squared gradient estimates (used in Adam)
β_1, β_2	hyperparameters used in advanced gradient algorithms
ϵ	small constant for numerical stability in adaptive optimizers
P, P_t	preconditioner matrix
$\mathcal{R}(\Phi)$	risk
$\lambda_1, \lambda_2, \ldots, \lambda_p$	eigenvalues of $H(\theta)$ sorted largest to smallest
$\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_p$	the corresponding eigenvectors
Q	matrix with eigenvectors as columns
Λ	diagonal matrix of eigenvalues
$S(\theta)$	Sharpness
MSS	maximum stable sharpness
$u(\theta), u_t = u(\theta_t)$	normalized version of $\mathbf{q}_1$, the eigenvector corresponding to the largest eigenvalue
$\mathcal{M}$	stable set for analysis in Section (9)
$\hat{\theta}_t$	constrained trajectory that stays in stable set

Continued on next page...

Table 1 – *Continued from previous page*

Symbol	Description
$\mathfrak{C}(\theta)$	progressive sharpening coefficient
$\mathfrak{B}(\theta)$	squared norm of gradient of sharpness
$\hat{x}_t, \hat{y}_t$	displacement along direction $u(\hat{\theta}_t)$ and $\nabla S(\hat{\theta}_t)$
x_t, y_t	displacement along direction $u(\theta^\star)$ and $\nabla S(\theta^\star)$
$w^\perp$	projection on the orthogonal complement of $u(\hat{\theta}_t)$
$\mathfrak{B}_{s\to t}$	quantifies sharpness change from time s to $t+1$ after an initial displacement
v_t	true displacement between real trajectory and constrained trajectory
$\overset{*}{v}_t, \overset{*}{x}_t, \overset{*}{y}_t$	predicted dynamics of v_t, x_t and y_t
$\mathfrak{C}_t$	progressive sharpening coefficient for $\hat{\theta}_t$
$\mathfrak{B}(\theta)$	squared norm of gradient of sharpness for $\hat{\theta}_t$
$\widetilde{\mathbb{E}}[\cdot]$	time average
$\bar{\theta}$	time average of θ at time t
γ_t	size of displacement in the unstable direction
$\gamma^2(t)$	continuous analogue to $\widetilde{\mathbb{E}}[\gamma_t^2]$
ξ_t	displacement vector (element of span of unstable eigenvectors)
$\Sigma(t)$	continuous analogue to $\widetilde{\mathbb{E}}\left[\xi_t \xi_t^\top\right]$
$S_{\text{scRMSP}}(\theta_t, \nu_t)$	effective sharpness for scalar RMSProp
$\overline{\nu}(t), \overline{\nu}_t$	stationary form that $\nu(t)$ converges to
$\overline{P}(\theta)$	stationary preconditioner
$S_{\text{RMSP}}(\theta_t, \nu_t)$	effective sharpness for standard RMSProp
$\overset{*}{\nu}$	fixed ν_t value used for analyzing Frozen Adam
$\|H(\theta)\|_{S_n}$	Interaction-Aware Sharpness (IAS) for full-batch gradient descent
$tr(\cdot)$	trace of a matrix
$\Sigma_{\nabla L}(\theta)$	covariance matrix of the gradients
Σ_θ	covariance matrix of θ (parameter noise)
$GNI(\theta)$	gradient-noise interaction
$C(\theta)$	scalar function representing an instability criterion
h	hyperparameters (learning rate, batch size)
τ	threshold for the instability criterion
$SS_{\mathcal{B}}(\theta)$	step sharpness for a single batch
$BS(\theta)$	Batch Sharpness
$\|H\|_{S_{\|\mathcal{B}\|}}$	Interaction-Aware Sharpness (IAS) for stochastic gradient descent
$\mathcal{F}_t$	filtration (σ-algebra of optimization history up to time t)
$\text{BS}_{\text{scRMSP}}$	effective batch sharpness for scalar RMSProp

Continued on next page...

Table 1 – *Continued from previous page*

Symbol	Description
BS_{RMSP}	effective batch sharpness for standard RMSProp
BS_{Adam}	effective batch sharpness for Adam
$\tilde{BS}_{SGD}$	modified batch sharpness for SGD
$\tilde{BS}_{Adam}$	modified batch sharpness for Adam

4 Neural Networks

Over the last decade, neural networks turned out to be the most influential machine learning technique there is. Like in many machine learning tasks, our goal is to approximate an unknown underlying function $\mathcal{F} : X \to Y$ from given data points $M = \{(x_{\text{in},k}, y_{\text{out},k})\}_{k=1}^{m}$. Neural networks are designed to resemble the structure and functioning of biological brains and were first introduced by McCulloch and Pitts [23]. There are many different architectures of neural networks, we will only introduce a few in this paper.

4.1 The Multilayer Perceptron

The multilayer perceptron (MLP) is one of the most fundamental and widely used neural network architectures. Its most basic building blocks, the so-called neurons, are given by a function

$$x_{\text{in}} \mapsto \sigma(w^\top x_{\text{in}} + b) \tag{1}$$

where w is the weight vector, b the bias term, and σ is a nonlinear activation function. The MLP itself is now created by organizing these neurons in layers: an input layer, one or multiple hidden layers, and an output layer.

Definition 1. *Let $\mathscr{L} \in \mathbb{N}$, $d_0, \dots, d_{\mathscr{L}+1} \in \mathbb{N}$, and let $\sigma : \mathbb{R} \to \mathbb{R}$. A function $\Phi : \mathbb{R}^{d_0} \to \mathbb{R}^{d_{\mathscr{L}+1}}$ is called a multilayer perceptron if there exist matrices $W^{(\ell)} \in \mathbb{R}^{d_{\ell+1} \times d_\ell}$ and vectors $b^{(\ell)} \in \mathbb{R}^{d_{\ell+1}}$, $\ell = 0, \dots, \mathscr{L}$, such that with*

$$\begin{aligned} x_{in}^{(0)} &:= x_{in} \\ x_{in}^{(\ell)} &:= \sigma\left(W^{(\ell-1)} x_{in}^{(\ell-1)} + b^{(\ell-1)}\right) && \text{for } \ell \in \{1, \dots, \mathscr{L}\} \\ x_{in}^{(\mathscr{L}+1)} &:= W^{(\mathscr{L})} x_{in}^{(\mathscr{L})} + b^{(\mathscr{L})} \end{aligned}$$

it holds

$$\Phi(x_{in}) = x_{in}^{(\mathscr{L}+1)} \quad \text{for all } x_{in} \in \mathbb{R}^{d_0}.$$

We call Φ a multilayer perceptron. $\mathscr{L}$ is called the depth and $d_{\max} = \max_{\ell=1,\dots,\mathscr{L}} d_\ell$ is called the width. Note that σ is meant to be applied component-wise on the respective vectors.

In the following, we will refer to entire set of matrices $W^{(\ell)}$ and bias vectors $b^{(\ell)}$ as the network's parameters θ. To be more precise, we create a single vector with the entries of each weight

matrix and bias vector like in the following:

$$\theta = \begin{pmatrix} \text{vec}(W^{(0)}) \\ b^{(0)} \\ \text{vec}(W^{(1)}) \\ b^{(1)} \\ \vdots \\ \text{vec}(W^{(\mathscr{L})}) \\ b^{(\mathscr{L})} \end{pmatrix}$$

where the vectorization of W is done row by row. This vector θ, called the parameter vector or just parameters, and its behavior during the training process of the neural network will be the primary focus of this thesis. We call the space of all parameters Θ and we address the i-th coordinate of this vector as $\theta^{(i)}$.

4.2 Other Neural Networks

Almost all neural network types like convolutional neural networks (CNNs), recurrent neural networks (RNNs) and transformer networks will have an appearance in this thesis. However, giving a detailed definition for each of them would go beyond the scope of this thesis. Instead, a short explanation will be given in this section:

Convolutional Neural Networks[20]:
CNNs are mostly used for image classification or other image tasks. Instead of seeing each input, most commonly a pixel, as independent, we actually incorporate the surrounding pixels as well. Mathematically, this is done by taking the convolution of a kernel (which has learnable parameters) and the input image in the first layer or the processed image in deeper layers. Additionally, some information is thrown out in each layer in a process called pooling and after the final convolutional layer, we use one or multiple fully connected layers like in the MLP networks. Similarly to MLPs, we can save all the entries of the kernels, all the biases and all the weights from the fully connected layers in a parameter vector θ.
Recurrent Neural Networks[10]:
RNNs are designed for sequential data. For each input, the network not only processes the current input but also uses the information of all past inputs. We do this by using so called hidden states. The hidden state at time k, which is called h_k, depends on the input at this time step c_k and the hidden state from the previous time step (h_{k-1}).To be more precise, for a sequential input $i^{(0)} = (c_1, c_2, ..., c_K)$, we get the new hidden state h_k by the following equation

$$h_k = \sigma(W_{hh}h_{k-1} + W_{ih}c_k + b_h).$$

To get a sequential output, $y_{\text{out},k}$ can be calculated from its corresponding hidden state:

$$y_{\text{out},k} = g(W_{ho}h_k + b_y).$$

Again, we can put all parameters of the weight matrices and biases in a single vector θ.

Transformer Networks[32]:
Transformer networks are highly complex models that have become the foundational architecture for natural language processing, but are now used for a variety of machine learning tasks. Instead of processing data sequentially like RNNs, they rely on an advanced self-attention mechanism to weigh the importance of different parts of the input simultaneously. Their detailed mathematical formulation is beyond the scope of this section. We just need to know that all of their attention weights, normalizations, and feed-forward parameters can again be encapsulated into a single parameter vector θ.

5 Optimization of Neural Networks

To optimize a neural network, we must find a set of parameters $\theta \in \Theta$ that minimizes a specific objective function. To find this objective function, we first define the per-sample loss function $\mathcal{L} : Y \times Y \to \mathbb{R}$, which measures the distance between a single prediction and its true target. The actual objective function we minimize is the empirical risk, which takes the parameters θ as an input, calculates the corresponding output for each input using those parameters, and averages the per-sample loss over all data points. Because it is common in literature about neural network optimization, we refer to the empirical risk as loss function $L(\theta)$, loss, or training loss in the following sections.

Definition 2. *Let $\mathcal{L} : Y \times Y \to \mathbb{R}$ be a per-sample loss function acting on the label space. Let Φ_θ denote a multilayer perceptron from Definition 1 parameterized by the vector θ. Then we define the empirical risk $\hat{\mathcal{R}}(\theta)$ or loss function $L(\theta)$ as:*

$$\hat{\mathcal{R}}(\theta) = L(\theta) = \frac{1}{m} \sum_{k=1}^{m} \mathcal{L}(\Phi_\theta(x_{in,k}), y_{out,k})$$

5.1 Gradient Descent

One of the simplest optimization algorithms is called gradient descent. It works by iteratively taking a small step in the direction of steepest descent, which is equal to $-\nabla L(\theta_t)$, the negative gradient of the function at that point. In our case, we try to minimize the loss function $L(\theta_t)$. An important parameter is the step size η, which governs how big the step in the direction of steepest descent will be. The step size has a huge influence on the behavior of the algorithms and is especially vital for the topic of this thesis.

Each gradient descent update step looks like this, where θ_t describes the values of our parameters at step t.

$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t). \tag{2}$$

We will start the algorithm with some initialization θ_0, generally depending on the neural network architecture used.

Algorithm 1 Gradient Descent [34, p. 7]

Input: L, η
Initialize θ_0 randomly or with some initial values
$t \leftarrow 0$
while convergence criterion not fullfilled **do**
    Compute the gradient: $\nabla L(\theta_t)$
    $\theta_{t+1} \leftarrow \theta_t - \eta \nabla L(\theta_t)$
    $t \leftarrow t + 1$
end while

This means to minimize this loss function $L(\theta_t)$, we will need to compute its gradient. This can be done very efficiently using so called backpropagation [30]. In simple terms, we use that a neural network is a composition of its layers, so we can use the chain rule to efficiently calculate the derivatives. However, backpropagation computes the gradient of the per-sample loss $\mathcal{L}$ for only a single data point at a time. Due to the linearity of the derivative, the true full gradient is given by:

$$\nabla L(\theta_t) = \frac{1}{m} \sum_{k=1}^{m} \nabla_\theta \mathcal{L}(\Phi_\theta(x_{\text{in},k}), y_{\text{out},k}).$$

This means to compute the gradient $\nabla L(\theta_t)$ for a single update step, the backpropagation algorithm must be executed m times, once for every sample in the dataset.

When using gradient descent to optimize a neural network, a good mental image to have is that of a loss surface consisting of deep valleys and steep mountains. A 2-dimensional example of this is provided in Figure 1. At any point θ_t, the negative gradient $-\nabla L(\theta_t)$ points into the direction with the steepest descent, in which we take a small step. While this low-dimensional representation is useful for intuition, it is important to keep in mind real neural networks have a high-dimensional loss surface with dimension equal to the number of parameters.

5.2 Stochastic Gradient Descent

In practice, modern neural networks are trained on massive datasets containing billions of samples m, while also using a very large number of parameters θ. For example, the famous large language model GPT-3 was trained on around 300 billion tokens (equivalent to samples) and uses 175 billion parameters [3]. Since the total computation scales approximately linearly with both of these factors, standard gradient descent becomes entirely computationally unfeasible. For this reason, it is virtually never used for training large-scale neural networks.

A simple, way more computationally efficient alternative is stochastic gradient descent (SGD), introduced by [29] in 1951. SGD is the basis for modern optimization algorithms for neural

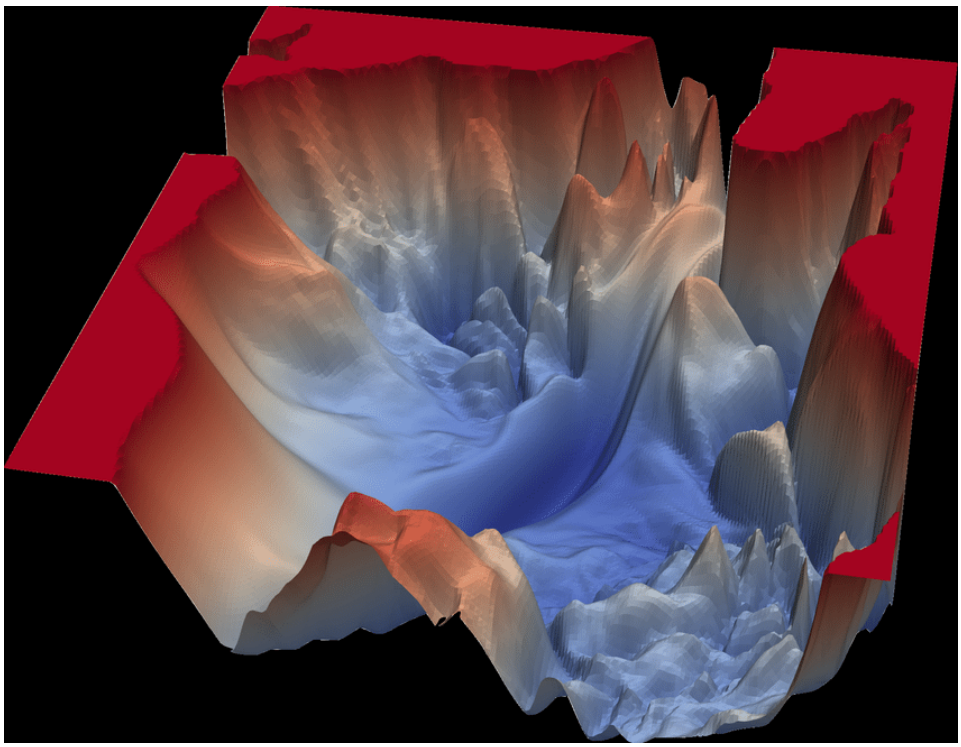


Figure 1: Visualization of a 2D slice of the loss surface of a residual neural network. The vertical axis is the value of the loss function. The two horizontal axes represent two random direction vectors that span a flat 2D plane through the high-dimensional parameter space Θ. Figure taken from [22].

networks. Instead of taking a step in the direction of the whole gradient $\nabla L(\theta_t)$, we randomly select a single data point $(x_{\text{in},k}, y_{\text{out},k})$ and calculate

$$\nabla L_k(\theta_t) = \nabla \mathcal{L}(\Phi(x_{\text{in},k}), y_{\text{out},k})$$

and take a step of step size η in this direction. But doing a gradient step for every single data point is also not that efficient, because we can not utilize the parallel processing of GPUs well. Additionally, the calculated gradients will differ widely in their directions and lead to highly unstable behavior during training. Because of this, the gradient descent method most commonly used in neural network training is mini-batch SGD, often referred to simply as SGD in machine learning literature. Mini-batch SGD works by dividing all the data points in batches. Each batch $\mathcal{B}_i$ consists of the indices of $|\mathcal{B}_i|$ samples, the so called batch size. We will often write it as $|\mathcal{B}|$ without the index, because typically all batches have the same size. Typical batch sizes reach from 16 to 256. For each batch, we calculate the average gradient of its samples and take a step in that direction. After doing this for every batch, we shuffle the dataset and create new batches, guaranteeing each sample is used in training before another sample is repeated. The training process for one set of batches is called an epoch. Let $L_{\mathcal{B}}$ be the loss of a full batch,

calculated like this:

$$L_{\mathcal{B}_i} = \frac{1}{|\mathcal{B}_i|} \sum_{k \in \mathcal{B}_i} L_k(\theta_t). \tag{3}$$

Algorithm 2 Mini-batch SGD

Input: L, samples M, η
Initialize parameters θ
while convergence criterion not fulfilled **do**
    Shuffle the training data M (start of new epoch)
    for each batch $\mathcal{B}$ in M **do**
        Compute the gradient of the batch: $\nabla L_{\mathcal{B}}(\theta_t) \leftarrow \frac{1}{|\mathcal{B}|} \sum_{k \in \mathcal{B}} \nabla L_k(\theta_t)$
        $\theta_{t+1} \leftarrow \theta_t - \eta \nabla L_{\mathcal{B}}(\theta_t)$
        $t \leftarrow t + 1$
    end for
end while

5.3 Momentum Based Optimizers (Heavy Ball, Nesterov)

A common variant of gradient descent are momentum based optimizers, the most common methods are the Heavy Ball method (also called Polyak algorithm) [27] and Nesterov algorithm [**nesterov1983AMF**]. In these optimizers, the parameter update includes a momentum vector m_{t+1}, resulting in the following update rule:

$$\theta_{t+1} = \theta_t - \eta m_{t+1}.$$

The momentum vector includes information about the previous steps the optimization algorithm took. This helps to avoid zig-zagging behavior and take bigger steps towards the local minimum. For each momentum based optimizer, there is a standard (st) version, which scales the previous momentum vector by β_1 and a exponential moving average (EMA) version, which also scales the gradient by $(1 - \beta_1)$ to ensure the momentum remains on the same numerical scale as the gradients. For Heavy Ball momentum, the momentum vectors are updated the following way:

$$\text{HB}^{\text{st}}(\eta, \beta_1): \quad m_{t+1} = \beta_1 m_t + \nabla L(\theta_t) \tag{4}$$

$$\text{HB}^{\text{EMA}}(\eta, \beta_1): \quad m_{t+1} = \beta_1 m_t + (1 - \beta_1) \nabla L(\theta_t). \tag{5}$$

The Nesterov algorithm incorporates a "lookahead" property, evaluating the gradient at a predicted future position based on the current momentum

$$m_{t+1} = \beta_1 m_t + \nabla L(\theta_t - \eta \beta_1 m_t).$$

By shifting the variables, we can get a mathematically equivalent formulation that allows the gradient to be calculated at the current position θ_t, which gives us the following formulas for

standard and EMA Nesterov updates:

$$\begin{aligned}
\text{Nes}^{\text{st}}(\eta, \beta_1): \quad & \hat{m}_{t+1} = \beta_1 m_t + \nabla L(\theta_t) \\
& m_{t+1} = \beta_1 \hat{m}_{t+1} + \nabla L(\theta_t) \\
\text{Nes}^{\text{EMA}}(\eta, \beta_1): \quad & \hat{m}_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla L(\theta_t) \\
& m_{t+1} = \beta_1 \hat{m}_{t+1} + (1-\beta_1)\nabla L(\theta_t).
\end{aligned}$$

β_1 is typically chosen as 0.9 for optimization of neural networks. [5]. Like with classical gradient descent, we can also use a mini-batch stochastic version of these algorithms, where the gradient is only calculated for a batch at each step.

5.4 Adaptive Gradient Methods

Adaptive gradient methods are methods that dynamically adjust the learning rate. [9] They do this by adapting to the gradients used in previous steps. Most of them can adjust the learning rate of each parameter individually, leading to quicker convergence, especially for sparse data.

5.4.1 Scalar RMSProp and RMSProp

One of the simplest adaptive gradient methods is scalar RMSProp (Scalar Root Mean Square Propagation) [31]. It calculates ν_t, which is a so-called exponential moving average EMA of the squared gradient. Instead of using the squared gradient directly, ν_t updates slowly by heavily weighting the past history of gradients. It uses the following formula:

$$\nu_{t+1} = \beta_2 \nu_t + (1-\beta_2)\|\nabla L(\theta_t)\|^2$$

A common value for β_2 is 0.99. The corresponding update rule of scalar RMSProp is

$$\theta_{t+1} = \theta_t - \eta \frac{1}{\sqrt{\nu_t} + \epsilon} \nabla L(\theta_t)$$

If the squared norm of the gradient is big, ν_t will increase over time and therefore the size of the steps the optimizer takes reduces. If the squared norm of the gradient is small, the optimizer will start to take larger steps. The $+\epsilon$ is present so we do not divide by 0. The more common standard RMSProp [31] weighs the size of the squared gradient for each parameter individually:

$$\nu_{t+1} = \beta_2 \nu_t + (1-\beta_2)\nabla L(\theta_t)^{\odot 2}$$

where $L(\theta_t)^{\odot 2}$ is the element wise squared gradient. The update formula is the same, but $\sqrt{\nu_t + \epsilon}$ represents a vector where for each entry of ν_t we add ϵ and take the entry wise square

root. The update formula becomes

$$\theta_{t+1} = \theta_t - \eta \frac{1}{\sqrt{\nu_t} + \epsilon} \odot \nabla L(\theta_t) \tag{6}$$

where $\odot$ is the entry wise product. The core principle of RMSProp is to dynamically adjust the step size for each parameter. If a certain entry of the gradient has low magnitude, the corresponding entry in ν_t will decline over time, which in turn increases the size of the step for that entry of the gradient. If an entry has high magnitude, the step size for that entry will decrease. This property allows RMSProp to effectively navigate ill-conditioned loss landscapes by decreasing the size of oscillations in steep directions and increasing steps in flat, gently sloping directions.

In the context of sharpness and the edge of stability, it makes sense to look at these algorithms as preconditioned gradient methods. A preconditioned momentum based gradient descent method uses the update formula

$$\theta_{t+1} = \theta_t - \eta P^{-1} m_{t+1}$$

where the matrix P is called a preconditioner. When using this formulation, RMSProp is an optimization algorithm with changing preconditioner, and the updates of RMSProp become [5]

$$\theta_{t+1} = \theta_t - \eta P_t^{-1} \nabla L(\theta_t) \tag{7}$$

where

$$P_t = \text{diag}(\sqrt{\nu_t}) + \epsilon I. \tag{8}$$

5.4.2 Adam Optimizer

A very popular optimizer used in real life deep learning tasks is Adaptive Moment Estimation (Adam) [19]. It is a combination of EMA Heavy Ball momentum and the step size adjustments from RMSProp. By merging these techniques, Adam simultaneously avoids zig-zagging and adapts to varying landscape curvatures.

The moments m_t and ν_t have the following update formulas:

$$\begin{aligned} m_{t+1} &= \beta_1 m_t + (1 - \beta_1) \nabla L(\theta_t) \\ \nu_{t+1} &= \beta_2 \nu_t + (1 - \beta_2) \nabla L(\theta_t)^{\odot 2} \end{aligned}$$

where commonly $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Because m_0 and ν_0 are initialized as vectors of zeros, these vectors are very close to zero for the first few optimization steps. Adam corrects

this bias by computing:

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1-\beta_1^{t+1}}$$
$$\hat{\nu}_{t+1} = \frac{\nu_{t+1}}{1-\beta_2^{t+1}}$$

The exponential terms in the denominator significantly increases the moments at the beginning of training, but do not change them much during later steps where $t+1$ is higher. The update formula then applies the scaled momentum step:

$$\theta_{t+1} = \theta_t - \eta \frac{1}{\sqrt{\hat{\nu}_{t+1}} + \epsilon} \odot \hat{m}_{t+1} \tag{9}$$

In the framework of preconditioned gradient descent, we can write Adam as the following

$$\theta_{t+1} = \theta_t - \eta P_t^{-1} m_{t+1} \tag{10}$$
$$P_t = (1-\beta_1^{t+1}) \left(\text{diag} \left(\sqrt{\frac{\nu_{t+1}}{1-\beta_2^{t+1}}} \right) + \epsilon I \right) \tag{11}$$

Again, we can define a stochastic variant that calculated the gradient from mini-batches. This stochastic Adam is one of the most widely used optimization algorithms in neural network training. The Adam optimizer efficiently reduces step size for noisy gradients and helps for smoother training in this stochastic setting [19].

6 Generalization and Sharpness

Generalization is a very important concept in machine learning. Broadly speaking, it describes the ability of a model to accurately predict outcomes for new, unseen samples. If for a given model the minimum θ not only gives us low loss for the data points it was trained on, but also for new data points from the same distribution, the model generalizes well.

Mathematically, if we assume our data points to come from a distribution $\mathcal{D}$ on $X \times Y$, we can define the risk of our network Φ as

$$\mathcal{R}(\Phi) = \mathbb{E}_{(x_{\text{in}}, y_{\text{out}}) \sim \mathcal{D}} \left[\mathcal{L}(\Phi(x_{\text{in}}), y_{\text{out}}) \right] \tag{12}$$

A neural network generalizes well if the absolute difference between the risk and the empirical risk is small. Because we never know the distribution $\mathcal{D}$ in practice, we estimate the true risk by calculating the empirical risk on a set of new, unseen data called the testset.

From an optimization perspective, there is a common notion that "flat" minima have better generalization properties. This idea was first introduced by [11], where a flat minimum is de-

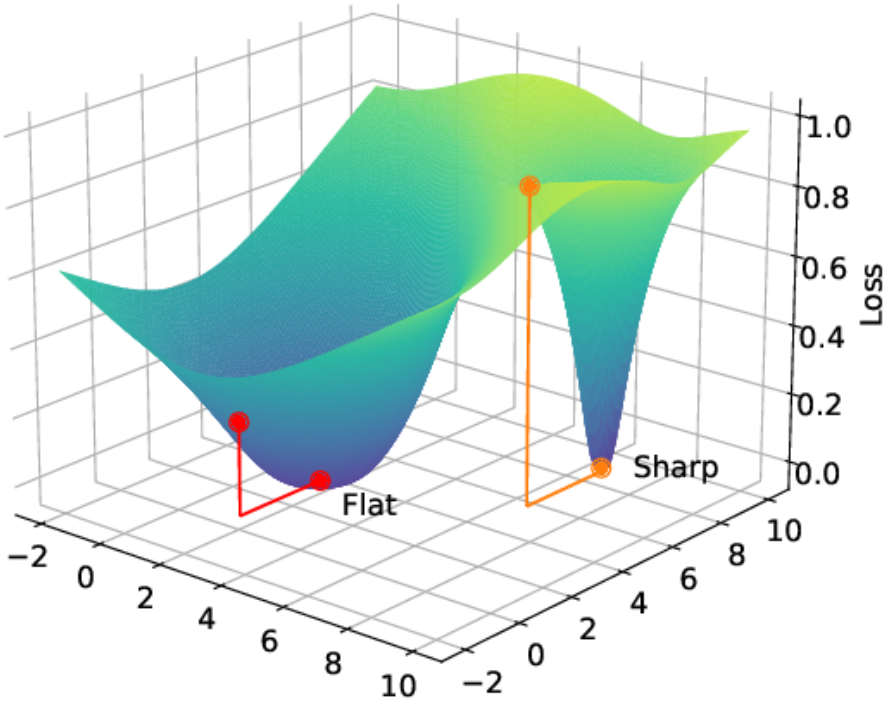


Figure 2: Illustration of a flat and a sharp minimum, visualized for a 2D slice of the loss surface. Illustration taken from [24].

fined by parameters close to the minima having a similarly small error. If the opposite is true and the region around the minimum does not have similarly low error, we call it a sharp minimum. An illustration of this is provided in Figure (2). [11] argues that flat minima do not have to be specified with as high of a precision as sharp minima, so their minimum description length (MDL) is shorter. Following the MDL principle introduced by [28], if the number of bits needed to describe the model is small, the complexity of our network is lower and therefore we have better generalization properties.

6.1 Sharpness of a Neural Network

The notion of flatness can also be related to the curvature of the loss function: If the region around the minimum has low curvature, the loss surface is relatively flat and the values of the loss function will not change a lot around the minima. The curvature of a scalar-valued function is characterized by its Hessian. The Hessian of specifically the loss function $L(\theta)$ will be very important all throughout this thesis. We will annotate it as $H(\cdot) = \nabla^2 L(\cdot)$.

By Clairaut's theorem, if the second partial derivatives of a function are continuous, the Hessian

is a symmetric matrix. For neural networks, this is true if both the activation function and the loss function are twice continuously differentiable. We can even use Clairaut's theorem for networks with an activation function like ReLU, where strictly speaking the second partial derivatives are discontinuous, since the points in the parameter space where the second derivatives are discontinuous are a set of measure 0. Since the Hessian $H(\theta)$ is a real and symmetric matrix, it can be diagonalized by an orthogonal matrix of its eigenvectors. If θ consists of p parameters, then $H(\theta) \in \mathbb{R}^{p \times p}$. Let $\lambda_1, \lambda_2, \ldots, \lambda_p$ be the eigenvalues of $H(\theta)$ ordered from largest to smallest and let $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_p$ be the corresponding eigenvectors. We define the orthogonal matrix Q and the diagonal matrix $\overline{\Lambda}$:

$$Q = \begin{bmatrix} | & | & & | \\ \mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_p \\ | & | & & | \end{bmatrix} \quad \overline{\Lambda} = \begin{pmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \lambda_p \end{pmatrix}$$

We can express $H(\theta)$ through these matrices

$$H(\theta) = Q\overline{\Lambda}Q^\top$$

Now we take a look at the second order Taylor expansion of the loss function around a point $\theta^\star$. Let $\Delta\theta := \theta - \theta^\star$ and $\Delta L := L(\theta) - L(\theta^\star)$ then the Taylor expansion gives us

$$\Delta L \approx \nabla L(\theta^\star)^\top \Delta\theta + \frac{1}{2}\Delta\theta^\top H(\theta^\star)\Delta\theta$$

Now we define a new vector $\Delta\theta' := Q^\top\Delta\theta$, define the transformed gradient $\nabla L(\theta^\star)' = Q^\top \nabla L(\theta^\star)$ and use $H(\theta) = Q\overline{\Lambda}Q^\top$ to get

$$\begin{aligned} \Delta L &\approx \nabla L(\theta^\star)^\top Q\Delta\theta' + \frac{1}{2}(\Delta\theta')^\top Q^\top Q\overline{\Lambda}Q^\top Q(\Delta\theta') \\ &= (\nabla L(\theta^\star)')^\top \Delta\theta' + \frac{1}{2}(\Delta\theta')^\top \overline{\Lambda}(\Delta\theta') \end{aligned}$$

Geometrically, this is a rotation and possible reflection of the coordinate system, aligning our new basis vectors with the eigenvectors. To see why this is useful, we write the last term as a sum:

$$\Delta L \approx \sum_{k=1}^{p} \left(\nabla L(\theta^\star)'_k \Delta\theta'_k + \frac{1}{2}\lambda_k (\Delta\theta'_k)^2 \right)$$

When calculating the slope of ΔL along the new axes aligned with the eigenvectors, we see that these principal axes are decoupled. This means that within the second order Taylor approximation, moving along one of these directions causes no change in the slope of any other direction. The corresponding eigenvalues λ_k now tell us if the curvature along the principal axes is positive or negative and also how strong this curvature is. For this reason, Keskar et al. [18] related the notion of sharpness of a function to the size of the positive eigenvalues of its Hessian. This means the largest eigenvalue corresponds to the maximum curvature in any direction, which

is why Keskar et al. [18] decided to use exactly this largest eigenvalue as a measure of curvature and define it as the sharpness $S(\theta)$ of the loss function.

$$S(\theta) = \lambda_{\max}(H(\theta)) \tag{13}$$

To build some intuition about the sharpness, it is good to think of the loss surface analogy again. Low sharpness corresponds to a wide, flat basin, while high sharpness corresponds to a steep, narrow valley. While the valley floor might be relatively flat, the valley walls rise abruptly. Going forward, we will analyze how $S(\theta)$ behaves during training.

6.2 Relation Between Sharpness and Generalization

The common notion of flatness of a minima being related to its generalization properties has some problems. Firstly, Dinh et al. [8] showed that for MLPs with ReLU activation function, for any minimum θ_1 with non-zero Hessian, we can create another minimum θ_2 with exactly the same prediction function everywhere, but arbitrarily high sharpness. This proves high sharpness does not necessarily guarantee high generalization properties. The paper was able to prove similar results for different definitions of flatness. It is important to note that in the paper, θ was deliberately changed in a specific way to increase the sharpness. So it is still possible that in practical deep learning settings, flatter minima have better generalization properties.

Kaur, Cohen, and Lipton [16] investigated the correlation between sharpness and generalization in numerical experiments, where they used SGD with varying batch sizes. They found many instances where lower sharpness solutions do not generalize better, for example when using large batch sizes or when scaling learning rate and batch size simultaneously. They also found methods like batch normalization improve generalization without affecting the sharpness at all. This means there can not be a simple 1 to 1 relation between sharpness and generalization. On the other hand, their research also found many instances where the results align with the conventional belief that flatter minima lead to better generalization. This was especially true for lower batch sizes and also for low dropout rates, where the correlation between low sharpness and high test accuracy was given.

Wen, Li, and Ma [33] showed that even for simple 2-layer ReLU MLPs, SAM can find solutions θ with low sharpness that do not generalize well when comparing them to full batch gradient descent. Interestingly, they found that for some architectures, SAM can still find a minimum with good generalization, even if there exist really flat minima with poor generalization.

To sum up, there definitely is not a causal relationship between low sharpness and good generalization. However, results seem to indicate, low sharpness solutions, or at least algorithms and optimization methods that lead to low sharpness solutions, still have good generalization properties and practice.

7 Progressive Sharpening

To understand how neural networks learn and at what minimum they will arrive, we want analyze the changes of sharpness $S(\theta)$ during the training. This was first investigated for SGD by Jastrzebski et al. [12], and later gained wider recognition in Cohen et al. [6] that did rigorous analysis and numerical experiments for GD. To establish a theoretical baseline, we first analyze the evolution of sharpness from a classical optimization perspective, utilizing the descent lemma.

7.1 Descent Lemma and Sharpness of Quadratic Function

In gradient descent, the step size η has a big influence on if and how fast the algorithm converges. There is a widely known result from classical optimization that uses the sharpness for choosing an appropriate η called the descent lemma [7].

Theorem 3. *Let $\mathcal{S}$ be an upper bound of $S(\theta)$ for all θ and let the gradient descent step be as in 1. Then:*

$$L(\theta_{t+1}) \leq L(\theta_t) - \frac{\eta(2-\eta\mathcal{S})}{2}\|\nabla L(\theta_t)\|^2 \tag{14}$$

From this theorem we see that at each step, the loss function L decreases at least by an amount proportional to $\frac{\eta(2-\eta\mathcal{S})}{2}$, which has a maximum at $\eta = 1/\mathcal{S}$. Because of this, $1/\mathcal{S}$ is a commonly used stepsize in practical applications. The theorem shows us that for any $\eta < 2/\mathcal{S}$, the function will decrease with each step. It does not tell use for general functions however, what happens if a step size bigger than $2/\mathcal{S}$ is chosen [7].

7.2 Gradient Descent on Quadratic Functions

For quadratic functions, the result from the descent lemma is even stronger than for general functions. Let $f : \mathbb{R}^n \to \mathbb{R}$, $A \in \mathbb{R}^{n\times n}$ be a positive definite matrix, $b \in \mathbb{R}^n$, $c \in \mathbb{R}$. Then the following function

$$f(\theta) = \frac{1}{2}\theta^\top A\theta + b^\top\theta + c \tag{15}$$

is a convex quadratic function. We can see easily that the minimum of this function is $\theta_{\min} = -b^\top A^{-1}$. Since the Hessian of f is always just A, the sharpness $S(\theta)$ of $f(\theta)$ is equal to the largest eigenvalue of A. For quadratic functions like f, we can prove an even stronger version of the descent lemma. The bound for the descent is sharp, this means for $\eta > 2/\mathcal{S}$, f will increase and the gradient descent algorithm will diverge.

Theorem 4 ([6])**.** *Consider running gradient descent on the quadratic function* (15) *starting from θ_0. Let $(\mathbf{q}, \lambda)$ be an eigenvector/eigenvalue pair of A. If $\lambda > 2/\eta$, then the sequence $\langle \mathbf{q}, \theta_t\rangle$ will diverge.*

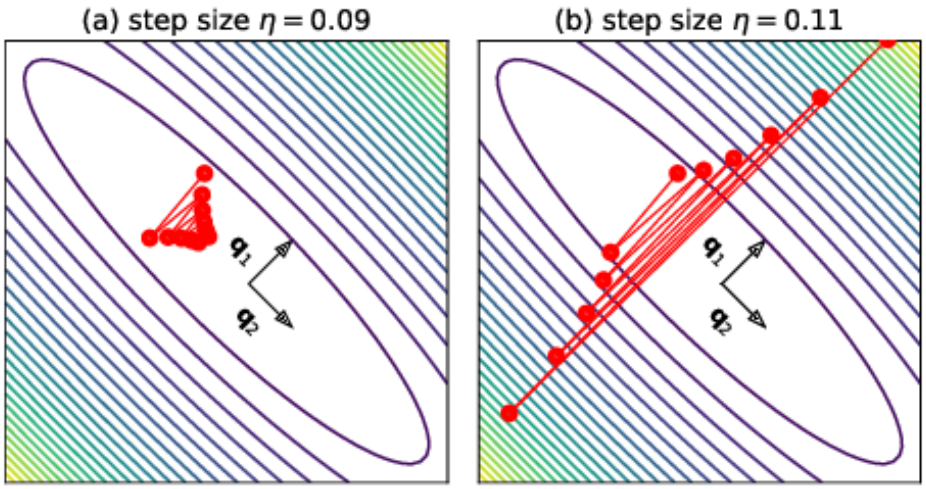


Figure 3: Divergence of gradient descent on a quadratic function with eigenvalues $\lambda_1 = 20$ and $\lambda_2 = 1$. Figure is taken from [7].

This means, as long as our initial value θ_0 has any component in the direction of the eigenvector $\mathbf{q}$, the algorithm will diverge. It can also be shown, that if $\langle \mathbf{q}, \theta_t \rangle$ diverges, then $\lambda > 2/\eta$. This means $\lambda > 2/\eta$ is both a necessary and a sufficient condition for divergence [6]. This relation between eigenvalue and step size can be observed well in Figure 3. Theoretically, this behavior should also occur for gradient descent on neural networks. Around the minima, we can assume the function to be approximately locally quadratic, so the above result should hold. Interestingly, the behavior of neural networks is totally different.

7.3 Progressive Sharpening

Progressive sharpening is a phenomenon that can be empirically observed when using full-batch gradient descent on neural networks (although we will later see variations of it apply for other optimization algorithms like mini-batch SGD or Adam). It describes the behavior of $S(\theta)$ as the iterations increase. If gradient descent is used on a neural network with a fixed step size η, $S(\theta)$ has a strong tendency to increase with the iterations. Especially in early iterations, it can happen that the sharpness decreases, but over longer time periods we always see an increase. For standard gradient descent, this increase of $S(\theta)$ happens until the sharpness reaches the maximum stable sharpness threshold, which is $2/\eta$ for gradient descent. During the progressive sharpening phase, the loss function decrease with each iteration, as it would be expected from the descent lemma.

The phenomenon of progressive sharpening was first conjectured by Jastrzebski et al. [12] in Assumption 4, where the phenomenon is investigated within the complex context of mini-batch SGD. The term "progressive sharpening" was first used in Cohen et al. [6], which studied the phenomenon in full-batch gradient descent and made some remarkable discoveries.

8 The Edge of Stability

The phenomenon of progressive sharpening comes to a sudden halt when the sharpness rises to the **maximum stable sharpness** (MSS). At this point, the algorithm enters the **Edge of Stability** (EoS) regime.

8.1 Maximum Stable Sharpness

The MSS depends solely on the optimization algorithm used. For the gradient descent based algorithms we introduced earlier, the maximum stable sharpness values are the following [5][6]:

$$\text{MSS}_{\text{GD}}(\eta) = \frac{2}{\eta} \tag{16}$$

$$\text{MSS}_{\text{HB}^{\text{st}}}(\eta, \beta_1) = \frac{2 + 2\beta_1}{\eta} \tag{17}$$

$$\text{MSS}_{\text{HB}^{\text{EMA}}}(\eta, \beta_1) = \frac{2 + 2\beta_1}{\eta(1 - \beta_1)} \tag{18}$$

$$\text{MSS}_{\text{Nes}^{\text{st}}}(\eta, \beta_1) = \frac{2 + 2\beta_1}{\eta(1 + 2\beta_1)} \tag{19}$$

$$\text{MSS}_{\text{Nes}^{\text{EMA}}}(\eta, \beta_1) = \frac{2 + 2\beta_1}{\eta(1 - \beta_1)(1 + 2\beta_1)} \tag{20}$$

Note that these MSS correspond exactly to the divergence bound when optimizing quadratic functions. This already gives us a hint that before we reach the edge of stability, the optimization behaves similar to optimization on a quadratic objective, but afterwards the behavior changes.

8.2 Sharpness at the Edge of Stability

When the optimization procedure reaches the edge of stability, the phenomenon of progressive sharpening stops. $S(\theta)$ does not monotonically increase anymore, instead it hovers just at or slightly above the MSS threshold. At this point, the algorithm is restrained from increasing the sharpness even further. Let $\mathbf{q_1}$ be the eigenvector corresponding to the largest eigenvalue of $H(\theta^\star)$, where $\theta^\star$ is the value of parameters at the point where the sharpness crosses the MSS. As in the quadratic case, $\langle \mathbf{q_1}, \theta_t \rangle$ gives us a single scalar value that tells us how the network's parameters are aligned with this direction of sharpest curvature. If the behavior was similar to the quadratic case, $\langle \mathbf{q_1}, \theta_t \rangle$ would start to diverge after the MSS is reached, because $S(\theta_t) > 2/\eta$ there, but in numerical experiments we see that this is not the case.

In the bottom plot in Figure 4(b) we see that at first, $\langle \mathbf{q_1}, \theta_t \rangle$ starts to oscillate really slowly, but these oscillations grow bigger over time. In the bottom plot in 4(c) however, we see that after some iterations these oscillations start to decrease again (Note that in the plot, the functions parameters are called x_i instead of θ_t). This trends of oscillations with rising and falling

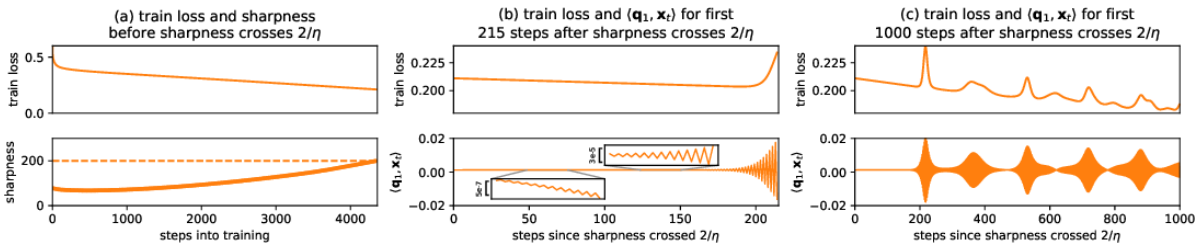


Figure 4: Closer look at sharpness and train loss before and after edge of stability. Figure taken from [7].

magnitudes continues. It can be seen that clearly the allocations never grow arbitrarily big and therefore the iterates are not diverging [6].

8.3 Training Loss at the Edge of Stability

We now examine how the behavior of sharpness and $\langle \mathbf{q_1}, \theta_t \rangle$ affects the actual value of the loss function. When the oscillations are still small, the training loss decreases as in the progressive sharpening phase, but when they grow bigger the loss actually increases. In the long term however, we still have decreasing loss as can be seen in the upper plot in 4(c). To sum up, when the sharpness reaches the MSS, the decrease in the loss function is no longer monotonic, but it still decreases over longer periods of time. This behavior is in stark contrast with the quadratic regime in Section 7.2, where the loss function would grow arbitrarily big after the sharpness passes the MSS [6].

8.4 Edge of Stability for Different Neural Network Architectures

Progressive sharpening and edge of stability show up for a wide range of neural network architectures. Cohen et al. [6] observed the phenomena empirically for both fully-connected networks and convolutional networks with various pooling methods (max and average) and activation functions (tanh, ReLU, and ELU).

Damian, Nichani, and Lee [7] observed progressive sharpening and edge of stability for residual neural networks (ResNet18) and for a 2-layer transformers with MSE loss.

8.5 Edge of Stability for Different Loss Functions

While most papers on the edge of stability focus on MSE loss, the phenomenon shows up for various loss functions.

Mean Squared Error (MSE) Loss: $L_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$. This is one of the most basic loss functions and also the standard example for progressive sharpening and EoS. The neural network in Figure 4 used MSE loss.

Cross-entropy Loss: $L_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$. For this loss function, we have slightly different behavior than in the stable regime. The sharpness is able to rise above the MSS by a greater amount than in the MSE case. If training is almost finished and the training loss is low, sharpness can decrease again, as seen in the right plot in Figure 5.

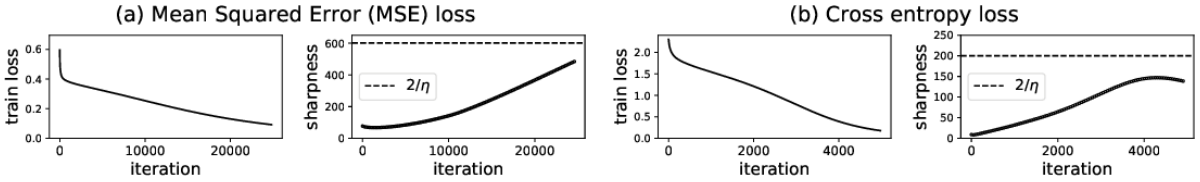


Figure 5: Progressive sharpening for MSE and CE loss. Figure taken from [6].

Simple Loss Functions: Ahn et al. [1] analyzed progressive sharpening for a very simple network and various loss functions. The paper uses a toy model setup with a single-neuron linear network model and margin-based loss to arrive at a loss function that uses just the product of the two learnable parameters: $L(\theta_1\theta_2)$. They were able to show progressive sharpening and edge of stability under the following assumptions:

(A1) L is convex, even, 1-Lipschitz, C^2 near the origin, and $L''(0) = 1$.

(A2) $\exists c_1 > 1, \exists c_2 > 0$ such that $\forall s \neq 0$:

$$\frac{L'(s)}{s} \leq 1 - c_2|s|^{c_1}\mathbb{1}\{|s| \leq c_2\}.$$

Assumption (A2) is always fulfilled if L is smooth and has a strictly negative fourth derivative at the origin. With these assumptions in place, it can be proven that GD converges both in the "gradient flow regime" with a small learning rate as well as in the "EoS regime" with a high learning rate. The exact theorems proving this can be found in [1].

Progressive sharpening and the EoS even show up when just using gradient descent on simple loss functions without any neural network. When trying to find the minimum of functions like $L(\theta_1, \theta_2) = (1-\theta_1^2\theta_2^2)^2$ [35] or $L(\theta_1) = (1-\theta_1^2)^2$, we can see both progressive sharpening and even edge of stability behavior when choosing an appropriate learning rate.

9 Self-Stabilization

When Cohen et al. [6] identified the phenomena of progressive sharpening and the edge of stability, it was still a big open question why the optimization trajectory behaves like that. Damian, Nichani, and Lee [7] found a heuristic intuitive explanation for the behavior, which is closely tied to the third-order Taylor expansion of the loss function and a property of gradient descent itself, rather than neural networks.

9.1 Self-Stabilization Property

We will examine a heuristic explanation for the edge of stability behavior given in [7]. To understand this, we need the following lemma:

Lemma 5 (Self-Stabilization Property)**.** *Let the top eigenvalue of $H(\theta)$ be unique. Let $u(\theta)$ be the corresponding normalized eigenvector. Then the sharpness $S(\theta)$ is differentiable at θ and $\nabla S(\theta) = \nabla^3 L(\theta)(u(\theta), u(\theta))$.*

While this identity is common knowledge in optimization theory, no explicit proof was given in [7]. For completeness, we prove it here:

Proof. Let $H(\theta)$ be the Hessian matrix of the loss function $L(\theta)$. Let $u(\theta)$ be the normalized eigenvector corresponding to the largest eigenvalue $S(\theta)$. By the definition of eigenvalues:

$$H(\theta)u(\theta) = S(\theta)u(\theta) \tag{21}$$

Since $H(\theta)$ is differentiable and $S(\theta)$ is unique, we can locally choose a differentiable branch for the normalized eigenvector $u(\theta)$. Any ± 1 sign ambiguity in this choice vanishes in our final derivative, as $u(\theta)$ appears quadratically. This means both $S(\theta)$ and $u(\theta)$ are differentiable functions with respect to θ.

By differentiating (21) with respect to a single parameter θ_i, we get:

$$\frac{\partial}{\partial \theta_i}(Hu) = \frac{\partial}{\partial \theta_i}(Su)$$

Applying the product rule and multiplying by $u^\top$ from the left yields:

$$u^\top \left(\frac{\partial H}{\partial \theta_i}\right) u + u^\top H \left(\frac{\partial u}{\partial \theta_i}\right) = u^\top \left(\frac{\partial S}{\partial \theta_i}\right) u + u^\top S \left(\frac{\partial u}{\partial \theta_i}\right)$$

Now we use $u^\top H = (Hu)^\top = (Su)^\top = Su^\top$, where the first equality is by the symmetry of H and the second equality is by the definition of the eigenvalue. Substituting this into the second term on the left-hand side gives:

$$u^\top \left(\frac{\partial H}{\partial \theta_i}\right) u + Su^\top \left(\frac{\partial u}{\partial \theta_i}\right) = \left(\frac{\partial S}{\partial \theta_i}\right) u^\top u + Su^\top \left(\frac{\partial u}{\partial \theta_i}\right)$$

By canceling $Su^\top \left(\frac{\partial u}{\partial \theta_i}\right)$ from both sides and using $u^\top u = 1$, we arrive at:

$$\frac{\partial S}{\partial \theta_i} = u^\top \left(\frac{\partial H}{\partial \theta_i}\right) u$$

The term $\frac{\partial H}{\partial \theta_i}$ is the derivative of the Hessian with respect to θ_i, which is a slice of the third-order derivative tensor $\nabla^3 L(\theta)$. Assembling the components for all i gives the gradient vector:

$$\nabla S(\theta) = \nabla^3 L(\theta)(u(\theta), u(\theta)) \tag{22}$$

□

For reasons which will be discussed in the next section, after we crossed the MSS and our iterates grow bigger in the direction of u, the term $\nabla^3 L(\theta^\star)(u(\theta^\star), u(\theta^\star))$ shows up in the Taylor expansion of $\nabla L(\theta)$, where $\theta^\star$ is a reference point we will define shortly. When expanding $\nabla L(\theta_t)$ in the update formula for gradient descent and using the self-stabilization property, we see this steers the algorithm in the opposite direction of greatest increase in sharpness.

$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t) = \theta_t - \eta(... + \nabla S(\theta^\star))$$

This decreases the sharpness again, which decreases the magnitude of $\nabla^3 L(\theta^\star)(u(\theta^\star), u(\theta^\star))$ in the Taylor expansion of $\nabla L(\theta)$. After we returned to stability, the algorithm is again in the phase of progressive sharpening.

9.2 Constrained Trajectory

Given that for gradient descent the sharpness never rises much above $2/\eta$, it makes sense to look at the optimization trajectory as implicitly solving the constrained minimization problem

$$\min_\theta L(\theta) \quad \text{such that} \quad S(\theta) \leq 2/\eta \tag{23}$$

We want to define a stable, constrained trajectory that avoids unstable regions. For this, we define the stable set:

$$\mathcal{M} := \{\theta | S(\theta) \leq 2/\eta, \nabla L(\theta) \cdot u(\theta) = 0\}$$

where the condition $\nabla L(\theta) \cdot u(\theta) = 0$ makes sure the gradient never points in the unstable direction, defined by the eigenvector $u(\theta)$ corresponding to the largest eigenvalue. This ensures the constrained trajectory stays stable. We can track both the real trajectory $\{\theta_t\}$ and the constrained trajectory $\{\hat{\theta}_t\}$ defined as

$$\hat{\theta}_{t+1} = \text{proj}_{\mathcal{M}}\left(\hat{\theta}_t - \eta \nabla L(\hat{\theta}_t)\right) \quad \text{where} \quad \text{proj}_{\mathcal{M}}(\theta) := \arg\min_{\theta' \in \mathcal{M}} \|\theta - \theta'\|$$

Because $\mathcal{M}$ is a closed set, the projection above is able to find a distance-minimizing point for each $\hat{\theta}_t$. However, because the loss landscape and $\mathcal{M}$ are both non-convex, this point is not necessarily unique. While the original paper by Damian, Nichani, and Lee [7] does not establish global uniqueness for this mapping, their theoretical proofs bypass this issue by applying the

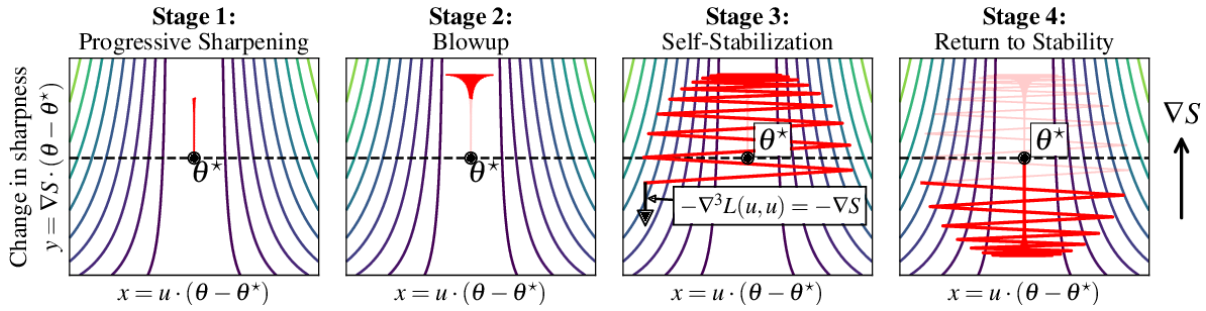


Figure 6: The 4 stages of the edge of stability visualized for a simple loss function. Figure from [7].

Karush-Kuhn-Tucker (KKT) conditions (Lemma 13). They demonstrate that the constrained update is governed by a unique first-order linear approximation of the projection and use this for their formalized proofs. For now, we will ignore the exact geometric projection for our heuristic argument to build intuition for the underlying dynamics of the Edge of Stability.

We define $\mathfrak{C}(\theta) := -\nabla L(\theta) \cdot \nabla S(\theta)$. We assume progressive sharpening exists and there is only one large eigenvalue. These assumptions are taken directly from [7].
Assumption 1 (Existence of progressive sharpening). $\mathfrak{C}(\hat{\theta}_t) > 0$.
Assumption 2 (Eigengap). For some absolute constant $c < 2$ we have $\lambda_2(H(\theta_t^\dagger)) < c/\eta$.

9.3 The Four Stages

A good start to investigate how gradient descent behaves at the edge of stability is to analyze the displacement between θ_t and $\hat{\theta}_t$. Using this framework, we can divide the behavior of the optimization algorithm into four stages. The algorithm starts in the first stage: progressive sharpening. When the edge of stability is reached the algorithm starts to cycle through the 4 stages [7]:

1. **Progressive Sharpening**
2. **Blow-up**
3. **Self-Stabilization**
4. **Return to Stability**

To analyze the displacement $\theta_t - \hat{\theta}_t$, we look specifically how it changes in 2 directions: The first is the direction of highest curvature, which is exactly the eigenvector $u(\hat{\theta}_t)$ corresponding to the largest eigenvalue. The second direction is the direction of steepest ascent of the sharpness $\nabla S(\hat{\theta}_t)$.

$$\hat{x_t} = u(\hat{\theta}_t) \cdot (\theta_t - \hat{\theta}_t)$$
$$\hat{y_t} = \nabla S(\hat{\theta}_t) \cdot (\theta_t - \hat{\theta}_t)$$

For a heuristic explanation of self-stabilization, these quantities are too complicated to use. Therefore, it makes sense to use a fixed reference point. We shift time such that θ_0 are the

parameters such that the sharpness first crossed the $2/\eta$ border. Then let the parameters of the constrained trajectory at that time be our reference point $\theta^\star = \hat{\theta}_0$. Notice that $S(\theta^\star) = 2/\eta$.

$$x_t = u(\theta^\star) \cdot (\theta_t - \theta^\star)$$
$$y_t = \nabla S(\theta^\star) \cdot (\theta_t - \theta^\star)$$

In a later section, we will show the simplified version yields really similar results to analyzing $\theta_t - \hat{\theta}_t$ directly [7].

The 4 stages can be seen in Figure 6. We see that in the progressive sharpening phase y_t is increasing, which means the displacements $\theta_t - \theta^\star$ grow in the direction of steepest increase in sharpness. In the second stage, our iterates start to oscillate along the direction of $u(\theta^\star)$, the eigenvector corresponding to our largest eigenvalue at point $\theta^\star$. As can be seen next section, this increases the impact of $-\nabla S(\theta^\star)$ in the Taylor expansion of $L(\theta_t)$, which drives gradient descent in a direction where sharpness decreases again. In the fourth and final stage, this decrease in sharpness also leads gradient descent to decrease the displacement in the direction of $u(\theta^\star)$ again. After the 4th stage, the algorithm goes back to stage 1 and the cycle starts again [7]. In the following, we will take a closer look at each of the stages in the heuristic example with the fixed reference point $\theta^\star$. For this simplified example, we also need the assumptions $\nabla S(\theta^\star) \perp u(\theta^\star)$ and $\nabla L(\theta^\star), \nabla S(\theta^\star) \in \ker(H(\theta^\star))$. These assumptions rarely hold in practice, they are only necessary for this heuristic model not needed in the more rigorous argument.

Stage 1: Progressive Sharpening

To start, we will look at how the displacement in the ∇S direction changes.

$$\begin{aligned} y_{t+1} - y_t &= \nabla S(\theta^\star) \cdot (\theta_{t+1} - \theta_t) \\ &= \nabla S(\theta^\star) \cdot (\theta_t - \eta \nabla L(\theta_t) - \theta_t) \\ &= -\eta \nabla S(\theta^\star) \cdot \nabla L(\theta_t) \end{aligned}$$

Since x_t and y_t are small in this stage, the gradients of the loss function are approximately equal for the real and the constrained trajectory: $\nabla L(\theta_t) \approx \nabla L(\theta^\star)$.

$$y_{t+1} - y_t = \nabla S(\theta^\star) \cdot -\eta \nabla L(\theta_t) \approx \nabla S(\theta^\star) \cdot -\eta \nabla L(\theta^\star) = \eta \mathfrak{C}(\theta^\star)$$

This means the movement in the direction of changing sharpness is equal to $\eta \mathfrak{C}(\theta^\star)$, which tells us the sharpness increases linearly with rate $\eta \mathfrak{C}(\theta^\star)$. [7]

Stage 2: Blowup

In the blowup stage, the displacement grows exponentially in the direction of the eigenvector u, which we will write as shorthand for $u(\theta^\star)$.This can be seen by analyzing the change of x_t:

$$\begin{aligned} x_{t+1} &= u \cdot (\theta_{t+1} - \theta^\star) \\ &= u \cdot (\theta_t - \eta \nabla L(\theta_t) - \theta^\star) \\ &= x_t - \eta u \nabla L(\theta_t) \end{aligned}$$

Next, we show that $u \cdot \nabla L(\theta_t) \approx S(\theta_t) x_t$.

$$\nabla L(\theta_t) \approx \nabla L(\theta^\star) + H(\theta^\star)(\theta_t - \theta^\star)$$

$$u \cdot \nabla L(\theta_t) \approx u \cdot \nabla L(\theta^\star) + u^\top H(\theta^\star)(\theta_t - \theta^\star)$$

For the quadratic term, the following rearrangement can be done, where we use that $H(\theta^\star)$ is symmetric and u a corresponding eigenvector:

$$u^\top H(\theta^\star) = u^\top H(\theta^\star)^\top = (H(\theta^\star)u)^\top = S(\theta^\star)u^\top$$

By the assumption $u \cdot \nabla L(\theta^\star) = 0$ and the definition of x_t we get

$$u \cdot \nabla L(\theta_t) \approx 0 + S(\theta^\star) u^\top (\theta_t - \theta^\star) = S(\theta^\star) x_t$$

In [7] $S(\theta_t)$ was used instead of $S(\theta^\star)$, according to the paper this requires using the third-order Taylor approximation. By Taylor expanding $S(\theta_t)$ again we can use $S(\theta_t) \approx S(\theta^\star) + \nabla S(\theta^\star)(\theta_t - \theta^\star) = 2/\eta + y_t$ to get:

$$\begin{aligned} x_{t+1} &= x_t - \eta u \nabla L(\theta_t) \\ &= x_t - \eta S(\theta_t) x_t \approx x_t - \eta(2/\eta + y_t) x_t \\ &= x_t - 2x_t - \eta y_t x_t = -(1 + \eta y_t) x_t \end{aligned} \tag{24}$$

We can see from this update rule that whenever $y_t \in [-2/\eta, 0]$, $|x_t|$ will decrease. Since $S(\theta_t) \approx 2/\eta + y_t$, y_t remains in $[-2/\eta, 0]$ during the progressive sharpening phase. Once $S(\theta_t)$ reaches $2/\eta$, y_t becomes positive and x_t will start to oscillate with increasing magnitude. The increase of $|x_t|$ is exponential, causing the iterates to diverge in the direction of top eigenvector u.

Stage 3: Self-Stabilization

Once the displacement $\theta_t - \theta^\star$ grows sufficiently large in the u direction, the loss is not locally quadratic anymore, so a higher order Taylor expansion is necessary to capture the behavior. We use the second-order Taylor expansion of the gradient of the loss function, which includes

the third-order derivative of the loss.

$$\nabla L(\theta_t) \approx \nabla L(\theta^\star) + \nabla^2 L(\theta^\star)(\theta_t - \theta^\star) + \frac{1}{2}\nabla^3 L(\theta^\star)[\theta_t - \theta^\star, \theta_t - \theta^\star] \tag{25}$$

$|x_t|$ grew exponentially during the blowup stage, meaning that in the direction of u there is a large displacement between θ_t and $\theta^\star$. Because the eigenvalues corresponding to other eigenvectors than u are strictly less than $2/\eta$, components of $\theta_t - \theta^\star$ orthogonal to u are still relatively small. The displacement is now approximately aligned with u, so $\theta_t - \theta^\star \approx (u \cdot (\theta_t - \theta^\star))u = x_t u$. Using the tri-linearity of the third-order tensor and the self-stabilization property, we get

$$\begin{aligned}
\frac{1}{2}\nabla^3 L(\theta^\star)[\theta_t - \theta^\star, \theta_t - \theta^\star] &\approx \frac{1}{2}\nabla^3 L(\theta^\star)\,[x_t u, x_t u] \\
&= \frac{x_t^2}{2}\nabla^3 L(\theta^\star)[u, u] \\
&= \frac{x_t^2}{2}\nabla S(\theta^\star)
\end{aligned}$$

By expanding the definition of y_{t+1} and plugging in the Taylor expansion of $\nabla L(\theta_t)$ we get

$$\begin{aligned}
y_{t+1} &= \nabla S(\theta^\star) \cdot (\theta_{t+1} - \theta^\star) \\
&= \nabla S(\theta^\star) \cdot (\theta_t - \eta \nabla L(\theta_t) - \theta^\star) \\
&= y_t - \eta \nabla S(\theta^\star) \cdot \nabla L(\theta_t) \\
&\approx y_t - \eta \nabla S(\theta^\star) \cdot \left(\nabla L(\theta^\star) + \nabla^2 L(\theta^\star)(\theta_t - \theta^\star) + \frac{1}{2}\nabla^3 L(\theta^\star)[\theta_t - \theta^\star, \theta_t - \theta^\star] \right) \\
&\approx y_t - \eta \nabla S(\theta^\star) \cdot \nabla L(\theta^\star) - \eta \nabla S(\theta^\star)^T \nabla^2 L(\theta^\star)(\theta_t - \theta^\star) - \eta \frac{x_t^2}{2} \|\nabla S(\theta^\star)\|^2
\end{aligned}$$

Since $\nabla S(\theta^\star) \in \ker(H(\theta^\star))$, the quadratic term vanishes. If we plug in $\nabla S(\theta^\star) \cdot \nabla L(\theta^\star) = -\mathfrak{C}(\theta^\star)$ and define $\mathfrak{B}(\theta^\star) := \|\nabla S(\theta^\star)\|^2$, we get the following equation:

$$y_{t+1} \approx y_t + \eta \left(\mathfrak{C}(\theta^\star) - \mathfrak{B}(\theta^\star)\frac{x_t^2}{2} \right) \tag{26}$$

When $|x_t| > \sqrt{2\mathfrak{C}(\theta^\star)/\mathfrak{B}(\theta^\star)}$, the change from y_t to y_{t+1} becomes negative, meaning y_t decreases at each iteration. Since $|x_t|$ grows at each step in the blowup stage, it will reach $\sqrt{2\mathfrak{C}(\theta^\star)/\mathfrak{B}(\theta^\star)}$ after some iterations. This leads to the iterates going in directions where the sharpness decreases.

We call $\mathfrak{C}(\theta^\star)$ the progressive sharpening force, which measures how much each gradient descent step increases the sharpness. $\mathfrak{B}(\theta^\star)$ represents the strength of the stabilization effect, which drives the sharpness lower again.

Stage 4: Return to Stability

Since in the self-stabilization stage the sharpness decreases, eventually the sharpness will pass the MSS again, but this time from above. This means y_t returns to the stable regime $y_t < 0$. Because the update formula (24) for x_t still holds, $y_t < 0$ leads to $|x_t|$ decreasing as well. The iterates return to the stable regime, where the progressive sharpening force described in Assumption 1 once again drives the sharpness toward the stability threshold.

9.4 Formalizing the Stabilizing Mechanism

To precisely depict the behavior of the algorithm at the edge of stability, we replace the fixed parameters $\theta^\star$ with the parameters $\hat{\theta}_t$ corresponding to the constrained trajectory at every time step. As established in Section 9.2, to ensure this trajectory is uniquely defined, the formal analysis relies on its unique first-order linear approximation. In this thesis, only the basic idea and the most important results will be outlined. More precise explanations and formalized proofs can be found in [7].

Let $P^{\perp}_{u(\hat{\theta}_t)}$ be the projection onto the orthogonal complement of the top eigenvector $u(\hat{\theta}_t)$. Using this projection, we can isolate the stable dynamics of the system from the unstable dynamics. For example, $P^{\perp}_{u(\hat{\theta}_t)} \nabla L(\hat{\theta}_t)$ is the direction of steepest ascent of the loss function within the stable subspace. Returning to the loss surface analogy, recall that high sharpness corresponds to a narrow, steep valley. In this setting, the top eigenvector $u(\hat{\theta}_t)$ represents the direction pointing directly up the steep valley walls, between which gradient descent oscillates. $P^{\perp}_{u(\hat{\theta}_k)}$ projects the iterates to the $(d-1)$-dimensional subspace where the fast, unstable oscillations are absent. Geometrically, this subspace corresponds to the valley floor, see Figure 7.

This allows us to ignore the impact of the short-term, rapid oscillations in the $u(\hat{\theta}_t)$ direction and focus entirely on measuring the slow, stable progress made along the bottom of the valley.

In subsection (9.3), we assumed $\nabla S(\theta^\star)$ and $u(\theta^\star)$ to be perpendicular. Now, this assumption is not needed anymore. We break down the stabilizing push in the $-\nabla S(\hat{\theta}_t)$ direction into two parts:

1. **Parallel component**: The part of $-\nabla S(\hat{\theta}_t)$ that aligns with $u(\hat{\theta}_t)$ and is just part of the rapid oscillations.
2. **Orthogonal component**: $S(\hat{\theta}_t)^{\perp} := P^{\perp}_{u(\hat{\theta}_t)} \nabla S(\hat{\theta}_t)$, which creates the stabilizing feedback loop.

In the following, let $w^{\perp} := P^{\perp}_{u(\hat{\theta}_t)} w$ be a shorthand notation for this projection.

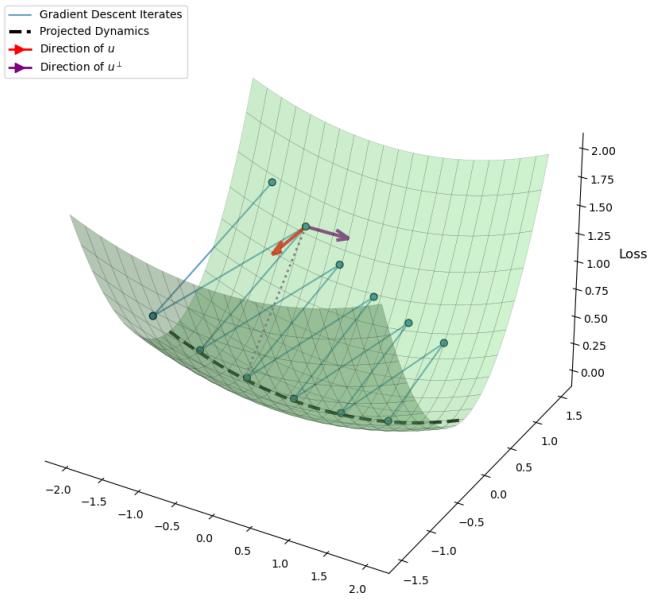


Figure 7: Geometry of $P^\perp$. The blue trajectory shows the gradient descent iterates oscillating. The oscillations occur primarily in the direction of u. The projection $P_u^\perp$ isolates the component of the dynamics along $u^\perp$. The projected dynamics (black dashed line) track the stable progress along the valley floor while filtering out high-frequency oscillations.

We now define the term $\mathfrak{B}_{s\to t}$, which quantifies the change in sharpness from time s to $t+1$ after an initial displacement of $\nabla S(\hat{\theta}_s)^\perp$ at time s.[1]

$$\mathfrak{B}_{s\to t} := \nabla S(\hat{\theta}_{t+1})^\perp \left[\prod_{k=t}^{s+1}(I-\eta H(\hat{\theta}_k))P^\perp_{u(\hat{\theta}_k)}\right]\nabla S(\hat{\theta}_s)^\perp$$

The term $-\nabla S^\perp(\hat{\theta}_s)$ is the correction of sharpness at time s. After we take a gradient step, this displacement vector evolves. Let d_s be the displacement between two points, θ_s and $\theta_s + d_s$. Then we use gradient descent to update both points:

$$\theta_{s+1} = \theta_s - \eta\nabla L(\theta_s)$$
$$\theta'_{s+1} = (\theta_s + d_s) - \eta\nabla L(\theta_s + d_s)$$

d_{s+1} is the difference between the updated points:

$$d_{s+1} = d_s - \eta\big[\nabla L(\theta_s + d_s) - \nabla L(\theta_s)\big]$$

[1] since s is smaller than t, the sum $\prod_{k=t}^{s+1}(I-\eta H(\hat{\theta}_k))P^\perp_{u(\hat{\theta}_k)}$ is to be interpreted as $(I-\eta H(\hat{\theta}_t))P^\perp_{u(\hat{\theta}_t)} * \ldots * (I-\eta H(\hat{\theta}_{s+1}))P^\perp_{u(\hat{\theta}_{s+1})}$.

By using the Taylor expansion of $\nabla L(\theta_s + d_s)$ around θ_s, we arrive at

$$d_{s+1} \approx d_s - \eta H(\theta_s) \cdot d_s = (I - \eta H(\theta_s))d_s$$

To analyze the long-term behavior of the constrained trajectory, we evaluate the Hessian at the projected iterates $\hat{\theta}_k$ and apply the projection $P^{\perp}_{u(\hat{\theta}_k)}$ at each step. The updated displacement d_t after $t - s$ steps is:

$$d_t \approx \left[\prod_{k=t}^{s+1}(I - \eta H(\hat{\theta}_k))P^{\perp}_{u(\hat{\theta}_k)}\right] d_s$$

Since the specific initial displacement at time s is the orthogonal sharpness gradient, we get:

$$d_t \approx \left[\prod_{k=t}^{s+1}(I - \eta H(\hat{\theta}_k))P^{\perp}_{u(\hat{\theta}_k)}\right] \nabla S(\hat{\theta}_s)^{\perp} \tag{27}$$

According to the first-order Taylor approximation, the change in a function f can be approximated by

$$f(\theta + d) - f(\theta) \approx \nabla f(\theta)^{\top} d$$

By taking the inner product of the projected sharpness gradient at time $t + 1$ with our evolved displacement vector d_t, we obtain the scalar value $\mathfrak{B}_{s\to t}$:

$$\mathfrak{B}_{s\to t} := (\nabla S(\hat{\theta}_{t+1})^{\perp})^{\top} \left[\prod_{k=t}^{s+1}(I - \eta H(\hat{\theta}_k))P^{\perp}_{u(\hat{\theta}_k)}\right] \nabla S(\hat{\theta}_s)^{\perp}$$

$\mathfrak{B}_{s\to t}$ quantifies the change in sharpness at time $t + 1$ after the initial displacement at time s. Next, we perform a similar analysis to Section 9.3, but instead of using the reference point $\theta^{\star}$, we are interested in the real displacement at each step $\theta_t - \hat{\theta}_t$. We define the quantities:

$$\begin{aligned} v_t &:= \theta_t - \hat{\theta}_t \\ x_t &:= u(\hat{\theta}_t) \cdot v_t \\ y_t &:= \nabla S(\hat{\theta}_t)^{\perp} \cdot v_t \end{aligned} \tag{28}$$

To model the behavior of these quantities, Damian, Nichani, and Lee [7] found a highly accurate prediction for these quantities. To define the predicted dynamics $\overset{*}{v}$, we need the following

definitions:

$$
\begin{aligned}
\overset{*}{v}_0 &= v_0 \\
\overset{*}{x}_t &= u(\hat{\theta}_t) \cdot \overset{*}{v}_t \\
\overset{*}{y}_t &= \nabla S(\hat{\theta}_t)^{\perp} \cdot \overset{*}{v}_t \\
\mathfrak{C}_t &= -\nabla L(\hat{\theta}_t) \cdot \nabla S(\hat{\theta}_t) \\
\mathfrak{B}_t &= \left\| \nabla S(\hat{\theta}_t)^{\perp} \right\|^2
\end{aligned}
$$

Then the update formula for $\overset{*}{v}$ is:

$$
\begin{aligned}
\overset{*}{v}_{t+1} = P^{\perp}_{u(\hat{\theta}_{t+1})} \left(I - \eta H(\hat{\theta}_t) \right) P^{\perp}_{u(\hat{\theta}_t)} \overset{*}{v}_t \\
+ \eta P^{\perp}_{u(\hat{\theta}_{t+1})} \nabla S(\hat{\theta}_t)^{\perp} \left[\frac{\left(\frac{2\mathfrak{C}_t}{\mathfrak{B}_t} \right)^2 - \overset{*}{x}_t^2}{2} \right] \\
- \left(1 + \eta \overset{*}{y}_t\right) \overset{*}{x}_t \cdot u(\hat{\theta}_{t+1})
\end{aligned}
\tag{29}
$$

While the formula for $\overset{*}{v}$ is quite involved, we get simple update formulas for $\overset{*}{x}_t$ and $\overset{*}{y}_t$:

$$
\begin{aligned}
\overset{*}{x}_{t+1} &= -(1 + \eta \overset{*}{y}_t) \overset{*}{x}_t \\
\overset{*}{y}_{t+1} &= \eta \sum_{s=0}^{t} \mathfrak{B}_{s \to t} \left[\frac{\left(\frac{2\mathfrak{C}_s}{\mathfrak{B}_s} \right)^2 - \overset{*}{x}_s^2}{2} \right]
\end{aligned}
$$

The update formula for $\overset{*}{x}_t$, the displacement in the direction of $u(\hat{\theta}_t)$, is structurally analogue to the simplified model in Section 9.3. For $\overset{*}{y}_t$, the update formula is similar as well, but incorporates the entire history of the process from $s = 0$ to t. For a certain s, if $|\overset{*}{x}_s| > \sqrt{\frac{2\mathfrak{C}_s}{\mathfrak{B}_s}}$, the corresponding term in the sum will be negative and drive $\overset{*}{y}_{t+1}$ in a negative direction, reducing the sharpness. The coefficient $\mathfrak{B}_{s \to t}$ determines how much influence this force has.

Damian, Nichani, and Lee [7] state that under certain assumptions, we can use the predicted dynamics $\overset{*}{v}_t, \overset{*}{x}_t$ and $\overset{*}{y}_t$ to approximate the trajectory θ_t as well as the evolution of the loss $L(\theta_t)$ and sharpness $S(\theta_t)$ at any given time t.

The assumptions needed are **Assumption 1** and **Assumption 2** from Section 9.2 as well as additional assumptions like upper bounds on $\nabla^3 L$ and its Lipschitz constant, $\mathfrak{C}$ not being arbitrarily small, the self-stabilization force being large enough, and $\nabla^2 L$ and $\nabla^3 L$ being nicely behaved as well as a generalization of the eigen gap assumption. The exact assumptions can be found in the appendix of this thesis.

Definitions:

$$\epsilon_t := \eta\sqrt{\mathfrak{C}_t}, \quad \epsilon := \sup_t \epsilon_t, \quad \delta_t := \sqrt{\frac{2\mathfrak{C}_t}{\mathfrak{B}_t}}, \quad \delta := \sup_t \delta_t$$

If assumptions 2, 3, 4 and 5 are fulfilled (**Assumption 1** is implied by **Assumption 4**, which is its generalized form), the following theorem holds.

Theorem 6. *Let $\mathfrak{T} := O(\epsilon^{-1})$ and assume that $\min_{t\leq\mathfrak{T}} |\overset{*}{x}_t| \geq c_1\delta$. Then for any $t \leq \mathfrak{T}$, we have*

$$\begin{aligned} L(\theta_t) &= L(\hat{\theta}_t) + \overset{*}{x}_t^2/\eta + O(\epsilon\delta^2/\eta) && \textit{(Loss)} \\ S(\theta_t) &= 2/\eta + \overset{*}{y}_t + (\nabla S(\hat{\theta}_t) \cdot u(\hat{\theta}_t))\overset{*}{x}_t + O(\epsilon^2/\eta) && \textit{(Sharpness)} \\ \theta_t &= \hat{\theta}_t + \overset{*}{v}_t + O(\epsilon\delta) && \textit{(Deviation from } \hat{\theta}_t\textit{)} \end{aligned}$$

The theorem provides a decomposition of the optimization process, it reveals that the real trajectory θ_t consists of the stable, constrained trajectory $\hat{\theta}_t$ and an oscillating process $\overset{*}{v}_t$ as well as a small error term. While θ_t appears chaotic, it remains close to a stable trajectory. The sharpness $S(\theta_t)$ is governed by a slowly evolving part $\overset{*}{y}_t$ as well as $(\nabla S(\hat{\theta}_t)\cdot u(\hat{\theta}_t))\overset{*}{x}_t$, which oscillates with a period of two steps. The behavior of the loss function itself can also be deduced from this theorem. When $u(\hat{\theta}_t)$ and $\theta_t - \hat{\theta}_t$ are aligned, $\overset{*}{x}_t^2$ is big and the real loss $L(\theta_t)$ is further away from the monotonically decreasing constrained loss $L(\hat{\theta}_t)$. When $\overset{*}{x}_t^2$ is small, $L(\theta_t)$ is close to $L(\hat{\theta}_t)$.

10 Modeling Gradient Descent Behavior with Differential Equations

The dynamics of gradient descent in neural networks can be modeled using differential equations. In classical optimization, an equation called gradient flow is used to model gradient descent. Cohen et al. [4] derived a new differential equation called central flow to model the time-averaged trajectory of θ_t. Their analysis leads to similar results as the previous section while providing a more intuitive framework as well as being easily extendable to multiple unstable eigenvalues.

10.1 Gradient Flow

When reordering the update formula for gradient descent (2) and incorporating the time t as a variable we get the following equation.

$$\frac{\theta_{t+1} - \theta_t}{(t+1) - t} = -\eta\nabla L(\theta_t)$$

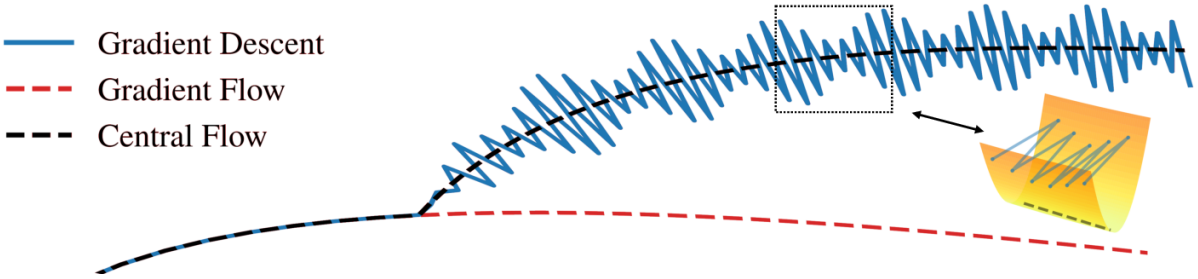


Figure 8: Central flow models the time-averaged trajectory of gradient descent. While gradient flow only stays close to this time-average during the progressive sharpening phase, central flow predicts gradient descent accurately even after the sharpness crossed the MSS. Figure taken from [4].

This translates to the continuous differential equation

$$\frac{d\theta}{dt} = -\eta \nabla L(\theta) \tag{30}$$

In Cohen et al. [6], it was observed that this equation approximates gradient descent in neural networks in the stable regime, where the sharpness is below the MSS of $2/\eta$. Because gradient flow assumes a infinitesimally small step size, it is no longer able to approximate the behavior of gradient descent at the edge of stability.

10.2 Central Flow

To encapsulate the behavior of gradient descent even after sharpness surpassed the MSS, we require a more sophisticated equation known as **central flow**, first derived by Cohen et al. [4]. Because of its highly oscillatory movements, describing the exact path of the iterates θ_t is unfeasible. However, when averaging θ_t over time we can predict the general trend of the optimization with high accuracy.

10.2.1 Local Time Averaging

In the following, we denote the local time averaged value as $\widetilde{\mathbb{E}}[\cdot]$. A short notation for time-averaging will be $\bar{\theta}_t = \widetilde{\mathbb{E}}[\theta_t]$. While Cohen et al. [4] do not specify an exact definition of local time average, three defining properties are listed that have to be satisfied.

1. **Linearity**: $\widetilde{\mathbb{E}}[f+g] = \widetilde{\mathbb{E}}[f] + \widetilde{\mathbb{E}}[g], \quad \forall c \in \mathbb{R} : \widetilde{\mathbb{E}}[cf] = c\widetilde{\mathbb{E}}[f]$
2. **Time average of constant**: $\forall c \in \mathbb{R} : \widetilde{\mathbb{E}}[c] = c$
3. **Time average at EoS**: at the EoS, the time average smooths out the oscillations such that $S(\widetilde{\mathbb{E}}[\theta_t]) = 2/\eta$.

In numerical experiments, we use Gaussian smoothing with a standard deviation of 50 steps [4]:

$$\widetilde{\mathbb{E}}\left[f_t\right] := \frac{\sum_s f_s c_{t-s}}{\sum_s c_{t-s}} \quad \text{where} \quad c_j := \exp\left(\frac{-j^2}{2 \times 50^2}\right).$$

f_t represents a certain quantity at time step t, the time average of f_t is weighted by a symmetric Gaussian function that depends on the squared distance in time.

10.2.2 Central Flow of Gradient Descent for a Single Unstable Eigenvalue

To gain an intuitive understanding, we begin with the same case we handled in Section 9. Only the largest eigenvalue has crossed the MSS, so there is only one unstable direction $u_t := u(\theta_t)$ corresponding to the eigenvector of said eigenvalue. We assume the sharpness already reached $2/\eta$, so we can write θ_t as a combination of its time-averaged value and the displacement:

$$\theta_t = \bar{\theta}_t + \gamma_t u_t$$

A central assumption is that we only consider the oscillations around $\bar{\theta}$ in the u_t direction, not in the $\nabla S(\bar{\theta}_t)$ direction or the stable directions. While the impact of $\nabla S(\bar{\theta}_t)$ is important for the long-term optimization process, the oscillations in the u direction are the dominant factor on shorter time scales. We call γ_t the signed displacement magnitude (or displacement for short) along the u_t direction. It is important to note that because the displacement is symmetrically distributed around $\bar{\theta}_t$, it holds that $\widetilde{\mathbb{E}}\left[\gamma_t\right] = 0$.

These assumptions allow us to do a Taylor expansion of $L(\theta)$ around the time-averaged $\bar{\theta}_t$ similar to the one we did for self-stabilization in subsection 9.3:

$$\begin{aligned}
\nabla L(\theta_t) &= \nabla L(\bar{\theta}_t) + H(\bar{\theta}_t)(\theta_t - \bar{\theta}_t) + \frac{1}{2}\nabla^3 L(\bar{\theta}_t)[\theta_t - \bar{\theta}_t, \theta_t - \bar{\theta}_t] + \mathcal{O}(\|\theta_t - \bar{\theta}_t\|^3) \\
&= \nabla L(\bar{\theta}_t) + H(\bar{\theta}_t)(\gamma_t u_t) + \frac{1}{2}\nabla^3 L(\bar{\theta}_t)[\gamma_t u_t, \gamma_t u_t] + \mathcal{O}(\|\gamma_t u_t\|^3) \\
&= \nabla L(\bar{\theta}_t) + \gamma_t H(\bar{\theta}_t) u_t + \frac{\gamma_t^2}{2}\nabla^3 L(\bar{\theta}_t)[u_t, u_t] + \mathcal{O}(\gamma_t^3) \\
&= \nabla L(\bar{\theta}_t) + \gamma_t S(\bar{\theta}_t) u_t + \frac{\gamma_t^2}{2}\nabla S(\bar{\theta}_t) + \mathcal{O}(\gamma_t^3)
\end{aligned}$$

where in the last step, we used the fact that $S(\bar{\theta}_t)$ is the eigenvalue corresponding to u_t for $H(\bar{\theta}_t)$ along with the self-stabilization property 5. To see how the time-averaged values change with the iterations, we perform the following calculation.

$$\bar{\theta}_{t+1} = \widetilde{\mathbb{E}}\left[\bar{\theta}_{t+1}\right] = \widetilde{\mathbb{E}}\left[\bar{\theta}_t - \eta \nabla L(\bar{\theta}_t)\right] = \bar{\theta}_t - \eta \widetilde{\mathbb{E}}\left[\nabla L(\theta_t)\right] \tag{31}$$

By plugging in the Taylor expansion from above in $\nabla L(\theta_t)$ we get:

$$\begin{aligned}\widetilde{\mathbb{E}}\left[\nabla L(\theta_t)\right] &\approx \widetilde{\mathbb{E}}\left[\nabla L(\bar{\theta}_t)\right] + \widetilde{\mathbb{E}}\left[\gamma_t S(\bar{\theta}_t) u_t\right] + \widetilde{\mathbb{E}}\left[\frac{\gamma_t^2}{2}\nabla S(\bar{\theta}_t)\right] \\ &= \nabla L(\bar{\theta}_t) + \frac{1}{2}\widetilde{\mathbb{E}}\left[\gamma_t^2\right]\nabla S(\bar{\theta})\end{aligned}$$

In the second step, we assume that the magnitude of oscillation changes way faster than the eigenvector u itself, so $\widetilde{\mathbb{E}}\left[\gamma_t u_t\right] \approx \widetilde{\mathbb{E}}\left[\gamma_t\right] u_t$. Since $\widetilde{\mathbb{E}}\left[\gamma_t\right] = 0$, we can remove the term from the equation. This means we do not need the $\nabla S(\theta^\star) \in \ker(H(\theta^\star))$ assumption from Section 9.

Combining the previous formulas yields

$$\bar{\theta}_{t+1} = \bar{\theta}_t - \eta\left[\nabla L(\bar{\theta}_t) + \frac{1}{2}\widetilde{\mathbb{E}}\left[\gamma_t^2\right]\nabla S(\bar{\theta})\right] \tag{32}$$

From this formula, we see the time-averaged iterates $\bar{\theta}_{t+1}$ do not merely take a step in the direction of negative time-averaged gradient of the loss $-\nabla L(\bar{\theta}_t)$. The step also includes a part in the direction of $-\nabla S(\bar{\theta})$, which reduces the value of the sharpness. The step size of this second direction is $\frac{1}{2}\eta\widetilde{\mathbb{E}}\left[\gamma_t^2\right]$. We discovered the exact same mechanism for sharpness reduction as in the Section 9 in an even more elegant way.

Now we want to determine the size of the sharpness reduction term $\widetilde{\mathbb{E}}\left[\gamma_t^2\right]$. It represents the average squared magnitude of oscillations, called variance of the oscillations in [4]. If the optimizer is oscillating wildly, $\widetilde{\mathbb{E}}\left[\gamma_t^2\right]$ will be large, if it does not oscillate at all, $\widetilde{\mathbb{E}}\left[\gamma_t^2\right]$ will be 0. If we define $\gamma^2(t)$ as the continuous analogue to $\widetilde{\mathbb{E}}\left[\gamma_t^2\right]$, we can derive a differential equation similar to (30). It describes the sharpness-penalized gradient flow $\theta(t)$:

$$\frac{d\theta}{dt} = -\eta\left[\nabla L(\theta) + \frac{\gamma^2(t)}{2}\nabla S(\theta)\right] \tag{33}$$

To find the value of $\gamma(t)$, we first take the derivative of the sharpness over time, using the chain rule and equation (33)

$$\frac{dS(\theta)}{dt} = \left\langle \nabla S(\theta), \frac{d\theta}{dt}\right\rangle = \eta\left\langle \nabla S(\theta), -\nabla L(\theta)\right\rangle - \frac{1}{2}\eta\gamma^2(t)\|\nabla S(\theta)\|^2$$

We assumed we are already at the edge of stability, and we know from empirical experiments that at this point, the sharpness will stop increasing further. Therefore, we enforce the sharpness to stay at $2/\eta$. Since the sharpness is already at $2/\eta$, this is equivalent to $\frac{dS(\theta)}{dt} = 0$. Rearranging for $\gamma^2(t)$ yields

$$\gamma^2(t) = \frac{2\left\langle \nabla S(\theta), -L(\theta)\right\rangle}{\|\nabla S(\theta)\|^2} \tag{34}$$

Substituting $\gamma^2(t)$ into equation (33) gives us the central flow for a single unstable eigenvalue after the EoS was reached. If we were still in the progressive sharpening phase, we set $\gamma^2(t) = 0$ and equation (33) reduces to standard gradient flow.

The central flow equation can also be seen as a projected gradient flow. We can rearrange (33) after substituting the definition of $\gamma^2(t)$ to get

$$\frac{d\theta}{dt} = -\eta \left[I - \frac{\nabla S(\theta) \nabla S(\theta)^\top}{\|\nabla S(\theta)\|^2} \right] \nabla L(\theta)$$

This equation is exactly gradient flow (30), with the difference that $\nabla L(\theta)$ is projected onto the orthogonal complement of $\nabla S(\theta)$ to ensure the sharpness does not rise any further [4].

10.3 Multiple Unstable Eigenvalues

Using the central flow framework, we can generalize to the case where multiple eigenvalues reach the MSS [4]. In this regime, the EoS oscillations happen in the span of the eigenvectors corresponding to the eigenvalues that reached the MSS. We define the multi-dimensional displacement ξ_t as an element in the span of these eigenvectors, and

$$\theta_t = \bar{\theta}_t + \xi_t$$

Like in the previous section, we Taylor expand the gradient

$$\nabla L(\theta_t) \approx \nabla L(\bar{\theta}_t) + H(\bar{\theta}_t)[\xi_t] + \frac{1}{2}\nabla^3 L(\bar{\theta}_t)[\xi_t, \xi_t] + \mathcal{O}(\|\xi_t\|^3)$$

Note that $\nabla^3 L(\bar{\theta}_t)[\xi_t, \xi_t] = \nabla_{\bar{\theta}_t} \left\langle H(\bar{\theta}_t), \xi_t \xi_t^\top \right\rangle = \nabla_{\bar{\theta}_t} \xi_t^\top H(\bar{\theta}_t) \xi_t$, so the third order term can be imagined as the gradient of the curvature in the direction of ξ_t. We again calculate the time-average of $\nabla L(\theta_t)$. Using $\widetilde{\mathbb{E}}[\xi_t] = 0$, we get:

$$\begin{aligned} \widetilde{\mathbb{E}}[\nabla L(\theta_t)] &= \nabla L(\bar{\theta}_t) + H(\bar{\theta}_t)\widetilde{\mathbb{E}}[\xi_t] + \frac{1}{2}\nabla^3 L(\bar{\theta}_t)\widetilde{\mathbb{E}}[\xi_t, \xi_t] \\ &= \nabla L(\bar{\theta}_t) + \frac{1}{2}\nabla^3 L(\bar{\theta}_t)[\widetilde{\mathbb{E}}[\xi_t, \xi_t]] \end{aligned} \tag{35}$$

The term $\widetilde{\mathbb{E}}[\xi_t, \xi_t]$ is the tensor product of the oscillation vector with itself which is equal to the time-average of $\xi_t \xi_t^\top$, so $\widetilde{\mathbb{E}}[\xi_t, \xi_t] = \widetilde{\mathbb{E}}\left[\xi_t \xi_t^\top\right]$. The matrix captures the average squared magnitude of oscillation along each principal direction. Using the definition of the covariance $\text{Cov}(X) = E[(X - \mu)(X - \mu)^\top]$ and the fact $\mu = \widetilde{\mathbb{E}}[\xi_t] = 0$, we get:

$$\text{Cov}(\xi_t) = \widetilde{\mathbb{E}}\left[\xi_t \xi_t^\top\right]$$

Transitioning to continuous time, we define the matrix $\Sigma(t)$ to model $\widetilde{\mathbb{E}}\left[\xi_t \xi_t^\top\right]$. By substituting the continuous version of equation (35) into equation (31), we get a differential equation that

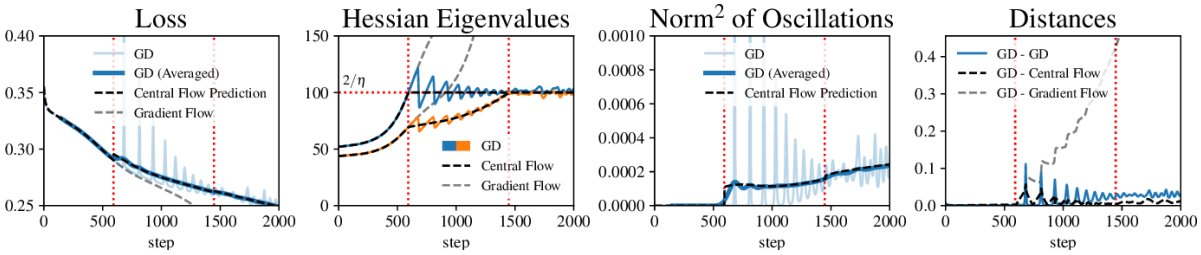


Figure 9: Comparison of gradient flow and central flow for a ViT trained on CIFAR-10 with gradient descent and $\eta = 0.05$. Figure reprinted from [4]

describes the path of $\bar{\theta}$:

$$\frac{d\theta}{dt} = -\eta \left[\nabla L(\theta) + \frac{1}{2} \nabla^3 L(\theta)[\Sigma(t)] \right] \tag{36}$$

Cohen et al. [4] impose three conditions that uniquely determine a single possible value for $\Sigma(t)$:

1. **Stability limit**: No eigenvalue of $H(\theta)$ is allowed to grow bigger than $2/\eta$.
2. **Support in span**: $\Sigma(t)$ is supported in the span of the eigenvectors of $H(\theta)$ corresponding to the eigenvalues equal to $2/\eta$.
3. **Positive Semidefiniteness**: $\Sigma(t)$ is positive semidefinite since it is a covariance matrix.

With these assumptions, Cohen et al. [4] were able to prove that $\Sigma(t)$ is equal to the unique solution of a semidefinite complementarity problem (SDCP). Again, if all eigenvalues are below $2/\eta$, then $\Sigma(t) = 0$ and central flow is equal to gradient flow. If one or multiple eigenvalues are at the $2/\eta$ boundary, the SDCP will output a $\Sigma(t)$ that generates the exact implicit sharpness penalty required to prevent those eigenvalues from increasing further. The rank of $\Sigma(t)$ is less or equal to the number of unstable eigenvalues. If we look at the eigenvalues of $\Sigma(t)$ itself, we see that each of them predicts the average squared magnitude of oscillations along their corresponding eigenvector.

The mathematical reasoning for central flow is largely informal, but the method performs exceptionally well in empirical studies and numerical experiments. In Figure (9), we can see that central flow captures the time-averaged values of the loss and of the Hessian eigenvalues well even after the sharpness reaches the MSS, while gradient flow only approximates these values well before the algorithm reaches the EoS.

10.4 Time-Averaged Trajectories of Arbitrary Functions

The central flow framework allows us to apply functions to the gradient descent trajectory and compute their time-averaged values. We can decompose the real iterates of gradient descent

θ_t into their continuous time-average $\theta(t)$ and the displacement ξ_t

$$\theta_t = \theta(t) + \xi_t, \quad \text{where } \widetilde{\mathbb{E}}[\xi_t] = 0 \text{ and } \widetilde{\mathbb{E}}[\xi_t, \xi_t] = \Sigma(t)$$

For any twice differentiable function f with input θ, we can perform the following Taylor expansion

$$f(\theta_t) = f(\theta(t) + \xi_t) \approx f(\theta(t)) + \nabla f(\theta(t))^\top \xi_t + \frac{1}{2}\xi_t^\top \nabla^2 f(\theta(t))\xi_t$$

If we apply time averaging, we get

$$\widetilde{\mathbb{E}}[f(\theta_t)] \approx f(\theta(t)) + \nabla f(\theta(t))^\top \underbrace{\widetilde{\mathbb{E}}[\xi_t]}_{=0} + \frac{1}{2}\widetilde{\mathbb{E}}\left[\xi_t^\top \nabla^2 f(\theta(t))\xi_t\right] \tag{37}$$

Since $\xi_t^\top \nabla^2 f(\theta(t))\xi_t$ is scalar, we can apply the trace and use the cyclic property to get $\text{tr}(\nabla^2 f(\theta(t))\xi_t\xi_t^\top)$, so

$$\widetilde{\mathbb{E}}\left[\xi_t^\top \nabla^2 f(\theta(t))\xi_t\right] = \text{tr}\left(\nabla^2 f(\theta(t))\widetilde{\mathbb{E}}\left[\xi_t\xi_t^\top\right]\right) = \text{tr}\left(\nabla^2 f(\theta(t))\Sigma(t)\right)$$

Applying this to equation (37), we end up with

$$\widetilde{\mathbb{E}}[f(\theta_t)] \approx f(\theta(t)) + \frac{1}{2}\text{tr}\left(\nabla^2 f(\theta(t))\Sigma(t)\right) \tag{38}$$

Setting $f(\theta) = L(\theta)$ yields

$$\widetilde{\mathbb{E}}[L(\theta_t)] \approx L(\theta(t)) + \frac{1}{2}\text{tr}\left(H(\theta(t))\Sigma(t)\right)$$

By condition (2) we required for $\Sigma(t)$, each column vector of $\Sigma(t)$ lies in the span of eigenvectors of $H(\theta)$ with eigenvalue $2/\eta$, meaning the column vector itself is an eigenvector with eigenvalue $2/\eta$. This implies $H(\theta(t))\Sigma(t) = \frac{2}{\eta}\Sigma(t)$.

$$\widetilde{\mathbb{E}}[L(\theta_t)] \approx \underbrace{L(\theta(t))}_{\text{central flow loss}} + \underbrace{\frac{1}{\eta}\text{tr}(\Sigma(t))}_{\text{oscillations}} \tag{39}$$

This formula tells that the loss of pure central flow $L(\theta(t))$ is much lower than the time-averaged loss along the gradient descent trajectory $\widetilde{\mathbb{E}}[L(\theta_t)]$, which is confirmed by numerical experiments. In the mental image of the loss landscape, the reason for this is quite intuitive: While gradient descent can be imagined as bouncing along the walls of a valley, central flow moves along the bottom of that valley. The additional loss from gradient descent not moving along the bottom of the valley is captured in the $\frac{1}{\eta}tr(\Sigma(t))$ term. Much of the gradient is "wasted" on oscillations, as can be seen by the size of gradient norm in Figure (10).

Using Equation (39), Cohen et al. [4] provide a formal proof that the time-averaged loss strictly decreases over time:

$$\frac{dL(\theta(t))}{dt} \leq 0$$

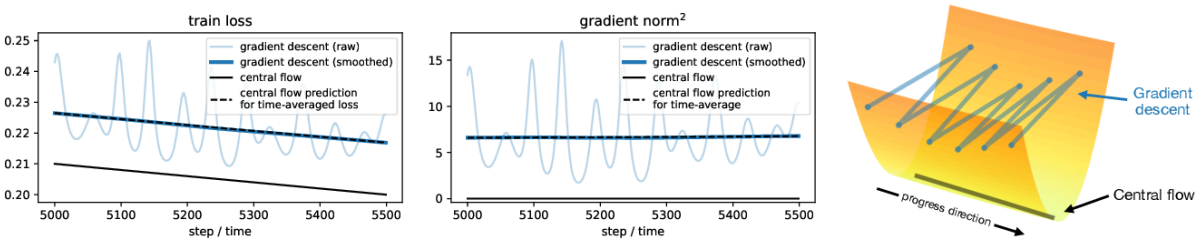


Figure 10: Central flow loss and gradient norm squared (solid black) are much lower than the time-averaged loss and gradient norm squared of gradient descent (dark blue, computed with Gaussian smoothing). The gap can be successfully predicted with equation (39). The figure on the right visualizes the intuitive explanation for this. Figure is taken from [4].

This confirms that the underlying trajectory of the optimizer consistently makes progress toward a minimum, despite the chaotic, high-frequency oscillations of the exact iterates. However, these oscillations still implicitly force gradient descent toward regions of reduced curvature, which slows down the training process. Proposition 2 in Cohen et al. [4] provides a strict lower bound on the rate of this loss decrease:

$$\frac{d}{dt}L(\theta(t)) \geq -\eta\|\nabla L(\theta(t))\|^2$$

The central flow framework has many other advantages. Through $\frac{d\theta}{dt}$, we can compute the actual direction the parameters are moving in over longer periods of time without getting distracted by the oscillations. By the chain rule, the rate of change of any function $f(\theta)$ can be computed:

$$\frac{df}{dt} = \nabla f(\theta)^\top \frac{d\theta}{dt}$$

To sum up, the central flow framework is an intuitive and useful extension of the self-stabilization described in Section 9. The time-averaging approach helps to ignore the chaotic oscillations and focus on the overall trajectory [4].

11 Edge of Stability for Adaptive Gradient Methods

In previous sections, the phenomena of progressive sharpening and the edge of stability were only investigated for standard gradient descent. In the following, we want to extend this analysis to the adaptive optimizers introduced in Section 5.4.

Cohen et al. [5] observed self-stabilization for various adaptive gradient methods: Adam (no bias correction), AdamW, Adafactor, AMSGrad, Padam, NAdam, RMSProp, and Adagrad. Cohen et al. [4] extended their central flow framework to scalar and standard RMSProp, making a

remarkable contribution to understanding how these optimizers operate at the edge of stability.

11.1 Edge of Stability for Scalar RMSProp

To gain an intuition how the sharpness of RMSProp and Adam behaves, we will first analyze scalar RMSProp, which offers a great intuition. Recall the scalar RMSProp update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\nu_t} + \epsilon} \nabla L(\theta_t) \tag{40}$$

The effective step size the algorithm takes is $\eta/(\sqrt{\nu_t} + \epsilon)$. By our previous analysis, we would expect the optimizer to become unstable when $S(\theta) \geq 2/\left(\frac{\eta}{\sqrt{\nu_t}+\epsilon}\right) = (\sqrt{\nu_t} + \epsilon)\frac{2}{\eta}$. When the sharpness crosses this new boundary, the optimizer starts to oscillate along the top eigenvector direction and enters the EoS regime. Cohen et al. [4] introduces effective sharpness S_{scRMSP} defined as the following[2]:

$$S_{\text{scRMSP}}(\theta_t, \nu_t) := S(\theta_t)/(\sqrt{\nu_t} + \epsilon) \tag{41}$$

Each time the effective sharpness crosses the $2/\eta$ boundary, the algorithm enters the EoS. Notably, the decay rate β_2 does not change this border. Because ν_t is allowed to increase further, the sharpness $S(\theta)$ can still increase, but $S_{\text{scRMSP}}(\theta_t, \nu_t)$ is constrained. Two mechanisms are responsible for this. Not only does $S(\theta)$ decrease because of the self-stabilization effect, the oscillations also result in a really large squared gradient norm $\|\nabla L(\theta_t)\|^2$, which increases ν_t and therefore decreases the effective sharpness. The effective sharpness $S(\theta)/(\sqrt{\nu_t} + \epsilon)$ staying at $2/\eta$ at the EoS in turn means that the effective step size $\eta/(\sqrt{\nu_t} + \epsilon)$ stays at $2/S(\theta)$. It is the largest stable step size available for gradient descent, which is utilized by scalar RMSProp automatically without even needing to calculate the sharpness. Figure 11 shows relevant metrics for an MLP trained with scalar RMSProp.

Although scalar RMSProp does not impose a strict upper bound on the true sharpness, the sharpness is still implicitly regularized. Training with larger learning rates results in larger oscillations, which in turn strengthens the sharpness regularization through the self-stabilization mechanism. This drives the optimizer into flatter regions of the loss landscape, resulting in smaller squared gradients, which in turn decrease ν_t and therefore increase the effective step size. This means a larger learning rate η also ensures a larger effective step size by keeping the sharpness low. This mechanism is coined "acceleration via regularization" by Cohen et al. [4].

[2] In [4], the effective sharpness also includes the step size in the numerator so the MSS is always equal to 2.

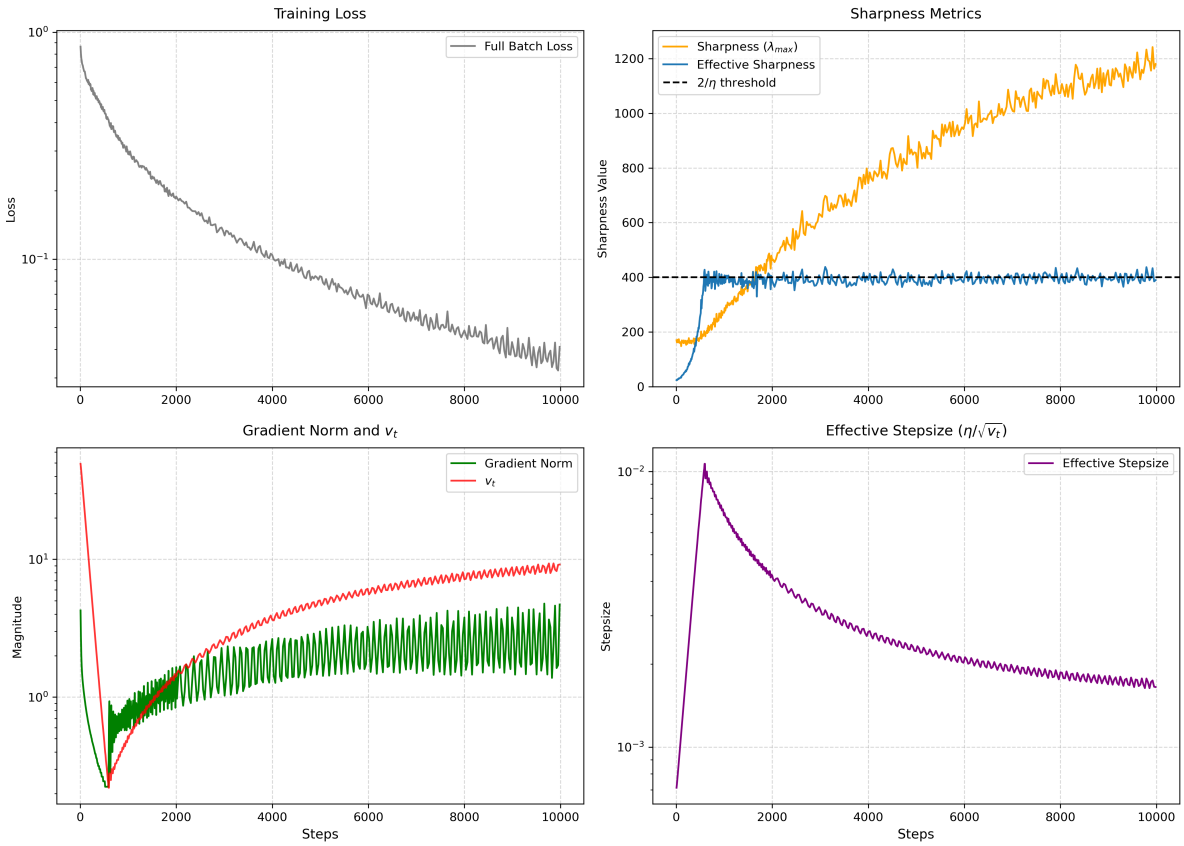


Figure 11: Full-batch scalar RMSProp training of an MLP with $\eta = 0.005$ on the first 4 classes of the CIFAR-10 dataset. Top right: Sharpness increases indefinitely, while effective sharpness is capped at $2/\eta$. Bottom left: After a steep decline from its initial value, both gradient norm and ν_t increase.

11.2 Edge of Stability for RMSProp

For the more widely used standard RMSProp optimizer, the update formula is:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\nu_t} + \epsilon} \odot \nabla L(\theta_t)$$

$$\theta_{t+1} = \theta_t - \eta P_t^{-1} \nabla L(\theta_t)$$

where $P_t = \text{diag}(\sqrt{\nu_t} + \epsilon)$. Cohen et al. [5] show in Appendix A.2 that preconditioned gradient descent on an objective function $f(x)$ is mathematically isomorphic to standard gradient descent on the reparameterized objective function $\hat{f}(x) = f(P^{-1/2}x)$. The Hessian of $\hat{f}(x)$ is $P^{-1/2}\nabla^2 f(x) P^{-1/2}$, which has exactly the same eigenvalues as $P^{-1}\nabla^2 f(x) = \text{diag}(1/(\sqrt{\nu_t} + \epsilon))\nabla^2 f(\theta_t)$. This means the effective sharpness for RMSProp is:

$$S_{\text{RMSP}}(\theta_t, \nu_t) = \lambda_1(P_t^{-1} H(\theta_t)) \tag{42}$$

As described for standard gradient descent in Section 10.3, the optimizer oscillates along the top eigenvector direction or the span of top eigenvector directions depending on whether one or multiple eigenvalues have crossed the $2/\eta$ boundary. Because RMSProp scales the geometry of the loss landscape independently along each coordinate axis, these oscillations occur strictly within the span of the eigenvectors of the preconditioned Hessian $P_t^{-1}H(\theta_t)$ instead of just the Hessian $H(\theta)$.

Cohen et al. [4] derived central flow equations for both scalar and standard RMSProp. These differential equations describe the time averages of the trajectories of their respective optimizers. For scalar RMSProp, the central flow is defined as:

$$\frac{d\theta}{dt} = -\frac{\eta}{\sqrt{\nu}+\epsilon}\left[\nabla L(\theta) + \frac{1}{2}\sigma^2_{\text{scRMSP}}(t)\nabla S(\theta)\right]$$
$$\frac{d\nu}{dt} = \frac{1-\beta_2}{\beta_2}\left[\|\nabla L(\theta)\|^2 + \sigma^2_{\text{scRMSP}}(t)S(\theta)^2 - \nu\right]$$

where $\sigma^2_{\text{scRMSP}}(t)$ is again derived under the assumption that at the EoS, $\frac{dS_{\text{scRMSP}}(\theta)}{dt} = 0$. For standard RMSProp, the central flow is:

$$\frac{d\theta}{dt} = -\frac{\eta}{\sqrt{\nu}+\epsilon} \odot \left[\nabla L(\theta) + \frac{1}{2}\nabla^3 L(\theta)\Sigma_{RMSP}(t)\right]$$
$$\frac{d\nu}{dt} = \frac{1-\beta_2}{\beta_2}\left[\nabla L(\theta)^{\odot 2} + \frac{4\nu}{\eta^2} \odot \text{diag}[\Sigma_{RMSP}(t)] - \nu\right]$$

where $\Sigma_{RMSP}(t)$ is derived similarly to $\Sigma(t)$ in Section 10.3. The exact derivation of these central flow equations is provided in Cohen et al. [4].

Using these central flow equations, we can accurately predict the time averages of various quantities during training. If we assume all the oscillations take place exclusively within the span of the eigenvectors whose eigenvalues have reached the MSS, the following approximation holds:

$$\underbrace{\mathbb{E}[\nabla L(\theta_t)^{\odot 2}]}_{\substack{\text{time-average of}\\ \text{squared gradient}}} \approx \underbrace{\nabla L(\theta(t))^{\odot 2}}_{\substack{\text{squared gradient at}\\ \text{time-averaged iterate}}} + \underbrace{\frac{4\nu}{\eta^2} \odot \text{diag}[\mathbb{E}[\xi_t \xi_t^\top]]}_{\substack{\text{contribution from}\\ \text{oscillations}}} \tag{43}$$

Examining the squared gradient for a specific coordinate, as pictured in Figure 12, we see the squared gradient evaluated on the central flow itself is very small. The majority of the squared gradient actually comes from the oscillations themselves. In the mental image of the optimizer bouncing between the walls of a valley, almost all of the size of the gradient is used for bouncing between the walls rather than going in the direction of the valley. The central flow, which smoothly runs along the bottom of the valley, has low gradient magnitude.

While the central flow $\nu(t)$ can not be expressed as a function of $\theta(t)$, if we freeze the parameters θ, there is a unique $\overline{\nu}(t)$ that $\nu(t)$ converges to. Empirically, Cohen et al. [4] found that $\nu(t) \approx \overline{\nu}(t)$ in the later stages of training at the Eos. The paper showed that the stationary

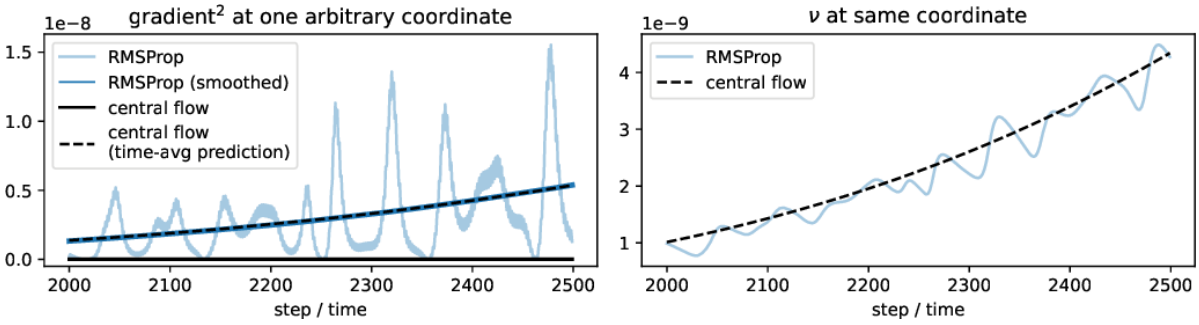


Figure 12: Central flow prediction for gradient squared and ν at a chosen coordinate. **Right**: The squared gradient of the central flow itself (solid black line at the bottom) is really low, as almost all of the magnitude of the squared gradients stems from the oscillations. Figure from [4].

preconditioner $\overline{P}(\theta) = \text{diag}(\sqrt{\overline{\nu_t}})$ is given by the following optimization problem:

$$\overline{P}(\theta) = \underset{P \text{ diagonal},\, P \succeq 0}{\text{argmin}} \; \text{trace}(P) + \underbrace{\|\nabla L(\theta)\|^2_{P^{-1}}}_{\text{optimization speed}} \text{ such that } \underbrace{H(\theta) \preceq \frac{2}{\eta} P}_{\text{local stability}}. \tag{44}$$

where $\preceq$ describes the Löwner partial order ($A \preceq B$ implies that $B - A$ is positive semi-definite). Since P is inversely proportional to the effective step size, minimizing it increases the effective step size and leads to faster convergence. By Taylor expansion for $L(\theta_{t+1})$ around θ_t, we can see $\|\nabla L(\theta)\|^2_{P^{-1}}$ is the instantaneous rate of loss reduction, known as the optimization speed.

$$\begin{aligned} L(\theta_{t+1}) - L(\theta) &\approx \nabla L(\theta_t)^\top (\theta_{t+1} - \theta_t) \\ &= -\eta \nabla L(\theta_t)^\top P_t^{-1} \nabla L(\theta_t) \\ &= -\eta \|\nabla L(\theta_t)\|^2_{P_t^{-1}} \end{aligned}$$

Minimizing $\|\nabla L(\theta)\|^2_{P^{-1}}$ actually slows optimization down. Cohen et al. [4] showed that the stationary preconditioner without this term actually performs better. If we assume $\nu(t)$ to converge infinitely fast to its stationary form $\overline{\nu}(t)$, we get the following simplified formula for RMSProp central flow:

$$\frac{d\theta}{dt} = -\eta \overline{P}(\theta)^{-1} \left[\nabla L(\theta) + \frac{1}{2} \nabla_\theta \langle \Sigma, H(\theta) \rangle \right]$$

$\overline{P}(\theta)$ being in the differential equation shows us that RMSProp is an implicit second-order optimizer. Despite only having access to first-order gradients, the optimizer forces its preconditioner to adapt to the curvature, which we see in the $H(\theta) \preceq \frac{2}{\eta} P$ constraint.

11.3 Edge of Stability for Momentum Based Optimizers

Before analyzing the edge of stability behavior of Adam in the subsequent section, we must first examine the edge of stability for momentum-based methods in general. Cohen et al. [6] investigated EoS for common momentum based optimizers and found their maximum stable sharpness is at a different value based on the same theoretical analysis on quadratics used for the divergence bound for gradient descent on quadratic functions. The exact MSS values were already introduced in 8.1.

Phunyaphibarn et al. [26] provide a highly intuitive explanation for how momentum based methods behave at the edge of stability. They explicitly analyze the edge of stability for Polyak's Heavy Ball momentum, which is equivalent to the standard Heavy Ball momentum defined in Section 4. When an optimizer bounces between the walls of a narrow valley, the gradient alternates signs at every step. Because the momentum vector "remembers" past gradients, these alternating gradients effectively cancel each other out. This damping effect keeps the update steps relatively small, allowing the optimizer to survive in much steeper, sharper regions of the loss landscape without diverging. This is precisely why momentum exhibits a higher maximum stable sharpness (MSS) than standard gradient descent.

However, if the sharpness continues to climb, the valley eventually becomes so incredibly steep that the current gradient becomes overwhelmingly large. At this threshold, the new gradient completely overpowers the canceling effect of the momentum. Similar to the gradient descent case when sharpness exceeds $2/\eta$, the optimizer begins to bounce violently. The high-magnitude gradients generated during this phase are accumulated in the momentum buffer, trapping the algorithm in this chaotic phase for much longer than standard gradient descent. As a result, the optimizer experiences a violent catapult effect, launching the parameters across the loss landscape and into flatter minima.

While the analysis in Phunyaphibarn et al. [26] strictly focuses on standard Heavy Ball momentum, their core insights directly translate to the exponential moving average (EMA) Heavy Ball used in Adam. The primary difference is that for EMA Heavy Ball the $(1-\beta_1)$ scaling further dampens the immediate impact of the gradient on the momentum, pushing the stability threshold even higher to $\text{MSS}_{\text{HB}^{\text{EMA}}} = \frac{2(1+\beta_1)}{\eta(1-\beta_1)}$ before the massive catapults are triggered.

11.4 Edge of Stability for Adam

To analyze the edge of stability for Adam, we need to combine our knowledge of EoS for Heavy Ball momentum and RMSProp. We already know that the MSS for EMA Heavy Ball is $\text{MSS}_{\text{HB}^{\text{EMA}}} = \frac{2+2\beta_1}{\eta(1-\beta_1)}$. We also can write Adam as preconditioned gradient descent with the preconditioner (11).

To better understand the distinct roles that the preconditioner and momentum play in Adam's stability, Cohen et al. [5] introduce a simplified version termed **Frozen Adam**. In this setup, the authors first train Adam normally and note the value of ν_t at some point during training,

denoted as $\overset{*}{\nu}$. They then restart the training from the beginning with the fixed $\overset{*}{\nu}$ in place of ν_t.

This is mathematically equivalent to preconditioned Heavy Ball momentum with $P = \text{diag}(\overset{*}{\nu}^{1/2}) + \epsilon I$. Empirical experiments confirm the edge of stability is reached when the preconditioned sharpness reaches $\text{MSS}_{\text{HB}^{\text{EMA}}}$. For Frozen Adam, the unconditioned sharpness also stops growing when the edge of stability is reached.

As we already saw for RMSProp, a gradually moving ν_t can lead to the sharpness still increasing even when the effective sharpness comes to a halt. The preconditioned sharpness has a different preconditioner at each step, but it still continuously rises until it reaches $\text{MSS}_{\text{HB}^{\text{EMA}}}$, where it remains bounded. [5] termed this the adaptive edge of stability (AEoS). Contrary to the behavior of Frozen Adam, the unconditioned sharpness is still able to grow when the AEos is reached. The Adam optimizer can change the values of ν_t in Preconditioner P_t to increase the true sharpness $S(H(\theta_t))$ while keeping the preconditioned sharpness $S(P_t^{-1}H(\theta_t))$ bounded. This way, theoretically the optimizer is able to train in regions with arbitrarily high curvature [5].

12 Interaction-Aware Sharpness

Lee and Jang [21] proposes the hypothesis that not the sharpness itself governs the EoS, but rather a quantity closely tied to sharpness, the interaction-aware sharpness (IAS). For full-batch gradient descent, the IAS is denoted as $\|H(\theta)\|_{S_n}$, where $S_n(\theta) = \nabla L(\theta)^\top \nabla L(\theta)$. It is defined as

$$\|H(\theta)\|_{S_n} := \frac{tr(H(\theta)S_n(\theta))}{tr(S_n(\theta))} \tag{45}$$

With some reformatting we get

$$\|H(\theta)\|_{S_n} = \left(\frac{-\nabla L(\theta)}{\|\nabla L(\theta)\|}\right)^\top H(\theta) \left(\frac{-\nabla L(\theta)}{\|\nabla L(\theta)\|}\right) = \frac{\nabla L(\theta)^\top H(\theta) \nabla L(\theta)}{\|\nabla L(\theta)\|^2} \tag{46}$$

Contrary to the sharpness, which is a measure for the highest curvature in any direction, $\|H(\theta)\|_{S_n}$ tells us how much the curvature changes along the direction of the GD step $-\nabla L(\theta)$. Note that $\|H(\theta)\|_{S_n} \leq S(\theta)$, and $\|H(\theta)\|_{S_n}$ is equal to $S(\theta)$ if and only if $\nabla L(\theta)$ is aligned with the top eigenvector $u(\theta)$.

Lee and Jang [21] prove that under certain assumptions, the edge of stability in full-batch gradient descent is triggered not when the standard sharpness $S(\theta)$ hits $2/\eta$, but rather when the interaction-aware sharpness $\|H(\theta)\|_{S_n}$ reaches this threshold. This is not a big contradiction to previous analysis using the normal sharpness to mark the EoS: When $S(\theta)$ reaches the $2/\eta$ mark, the gradient $\nabla L(\theta)$ quickly aligns with the top eigenvector $u(\theta)$, therefore also crossing the $2/\eta$ mark. This explains why in numerical experiments, instead of hovering right at $2/\eta$, the sharpness $S(\theta)$ stays slightly above $2/\eta$ at the EoS. All of this is backed up by numerical experiments in [21].

A heuristic derivation of the IAS can be done by Taylor expanding $L(\theta_{t+1})$

$$L(\theta_{t+1}) \approx L(\theta_t) + \nabla L(\theta_t)^\top (\theta_{t+1} - \theta_t) + \frac{1}{2}(\theta_{t+1} - \theta_t)^\top H(\theta_t)(\theta_{t+1} - \theta_t)$$

By plugging in the update formula of standard gradient descent (Algorithm 1), and simplifying we get

$$L(\theta_{t+1}) - L(\theta_t) \approx -\eta \nabla L(\theta_t)^\top \nabla L(\theta_t) + \frac{\eta^2}{2} \nabla L(\theta_t)^\top H(\theta_t) \nabla L(\theta_t)$$

For the optimization step to be locally stable, we need $L(\theta_{t+1}) - L(\theta_t) \leq 0$. Applying this to the equation yields

$$-\eta \|\nabla L(\theta_t)\|^2 + \frac{\eta^2}{2} \nabla L(\theta_t)^\top H(\theta_t) \nabla L(\theta_t) \leq 0$$

Assuming we are not at a stationary point yet, we can divide both sides by $\eta \|\nabla L(\theta_t)\|^2$ and rearrange the terms. We see that $\|H(\theta)\|_{S_n} \leq 2/\eta$ has to hold for the optimization step to be stable:

$$\frac{\nabla L(\theta_t)^\top H(\theta_t) \nabla L(\theta_t)}{\|\nabla L(\theta_t)\|^2} \leq \frac{2}{\eta}$$

Lee and Jang [21] also state a version of IAS that is suitable for SGD, which we will study in the next section.

13 SGD: Progressive Sharpening and Edge of Stability

While previous sections focused on full-batch dynamics, real-world neural networks are trained almost exclusively using mini-batch versions of optimization algorithms. This section will investigate progressive sharpening and edge of stability behavior for stochastic gradient descent (SGD) (2). Many publications ([17], [14], [15], [13]) investigated the sharpness in the case of SGD even before Cohen et al. [6] introduced the phenomena of progressive sharpening and EoS. They found that the sharpness of SGD goes through a progressive sharpening phase similar to that of GD, but stabilizes at different values not only depending on the learning rate, but also on batch size. The papers failed to find a clear boundary (like $2/\eta$ in the GD case) where the sharpness stabilizes. In 2024, Andreyev and Beneventano [2] reinvestigated the EoS phenomenon for mini-batch SGD with the new results from the GD case and made some remarkable discoveries. They found a generalized version of the IAS, which they termed batch sharpness $BS(\theta)$. When using SGD, at some point during training the batch sharpness stabilizes at $2/\eta$, while the sharpness $S(\theta)$ is generally below that. The phenomenon was termed edge of stochastic stability (EoSS).

Andreyev and Beneventano [2] use a slightly simplified version of mini-batch SGD, where each mini-batch $\mathcal{B}$ is constructed by sampling with replacement (meaning individual data points can be selected multiple times before the entire dataset is seen) from the entire dataset according

to a fixed distribution $\mathcal{P}$. While this differs from the standard practice of sampling without replacement (training in epochs), the authors demonstrate in Appendix E that nearly identical results hold for both case. In the subsequent analysis, let $L_{\mathcal{B}}$ be defined as in Equation (3) and let $H_{\mathcal{B}} = \nabla^2 L_{\mathcal{B}}(\theta)$ denote the stochastic Hessian.

13.1 Noise-Driven and Curvature-Driven Oscillations

The random sampling of mini-batches causes the gradient and Hessian to be stochastic, fluctuating from step to step. This means SGD exhibits inherent noise. In the previous chapters about GD, oscillations of θ and oscillations of the loss were necessarily linked to the curvature of the loss surface and instability, but for SGD this is not the case. The oscillations may arise solely from the random sampling of batches. Because of this, it is important to differentiate between the Type-1 **noise-driven oscillations** inherent to SGD and the Type-2 **curvature-driven oscillations** at the edge of stochastic stability.

To treat this with mathematical rigor, we introduce the notion of quadratic instability and catapults.

Definition 7 (Quadratic Instability and Catapults)**.** *Let $c_k \in \mathbb{R}^d$ be the parameter vector that minimizes the loss $L_k(\theta)$ for the k-th data point. Let $H_k = \nabla^2 L_k(c_k)$ be the Hessian matrix of the loss for a single data point evaluated at c_k. Consider the quadratic approximation of the loss landscape for each individual data point:*

$$L_k(\theta) \approx \frac{1}{2}(\theta - c_k)^\top H_k(\theta - c_k). \tag{47}$$

We say that a set of hyperparameters is unstable if the trajectory exits all the compact subsets of the region in which the quadratic approximation holds up to $\mathcal{O}(\eta)$. We say the algorithm experiences a catapult when this event occurs.

In simple terms, a catapult happens when the optimizer launches to a region that does not fulfill the quadratic approximation. Definition (7) allows us to define the difference between Type-1 and Type-2 oscillations mathematically.
Noise-driven oscillations are stable under the definition above. Although the trajectory oscillates only because of stochastic gradient noise, the step size is small relative to the curvature. Therefore the parameters θ stay strictly within the region where the quadratic approximation holds. No catapults occur.
Curvature-driven oscillations saturate the stability defined in Definition 7. This means the trajectory operates exactly at the boundary of the region where the quadratic approximation holds. The optimizer does not exit the region, instead it reaches the limit of stability. This means Type-2 oscillations are very sensitive to catapults. Even slight changes in the hyperparameters (like increasing the learning rate or decreasing the batch size) force the trajectory out of the quadratic region, triggering the catapult.

13.2 Type-1 Oscillations

We can mathematically model Type-1 oscillations locally around a minimum $\theta^\star$ (where $\nabla L(\theta^\star) = 0$). When using SGD in this scenario, θ does not converge to a single point, but rather settles in a stationary distribution, oscillating around $\theta^\star$. To understand this behavior, first we want to introduce the covariance matrix of the gradients $\Sigma_{\nabla L}(\theta)$. It uses the noise $\nabla L_{\mathcal{B}}(\theta) - \nabla L(\theta)$, the difference between the gradient for a random batch $\mathcal{B}$ and the true gradient at that point $\nabla L(\theta)$.

$$\Sigma_{\nabla L}(\theta) = \mathbb{E}_{\mathcal{B}\sim\mathcal{P}}[(\nabla L_{\mathcal{B}}(\theta) - \nabla L(\theta))(\nabla L_{\mathcal{B}}(\theta) - \nabla L(\theta))^\top]$$

The matrix $\Sigma_{\nabla L}(\theta)$ describes the geometry and magnitude of noise, meaning it indicates how high the noise is as well as which directions in the parameter space are more or less affected by noise. Because at the minimum $\nabla L(\theta^\star) = 0$, we get $\Sigma_{\nabla L}(\theta^\star) = \mathbb{E}_{\mathcal{B}\sim\mathcal{P}}[\nabla L_{\mathcal{B}}(\theta^\star)\nabla L_{\mathcal{B}}(\theta^\star)^\top]$. This matrix is called the covariance of the noise at the optimum.

A mathematical way to check if a system is oscillating at all is checking the gradient-noise interaction (GNI).

$$GNI(\theta) := \frac{\mathbb{E}_{\mathcal{B}\sim\mathcal{P}}\left[\nabla L_{\mathcal{B}}(\theta)^\top H(\theta)\nabla L_{\mathcal{B}}(\theta)\right]}{\|\nabla L(\theta)\|^2} \tag{48}$$

If $GNI \approx \frac{2}{\eta}$, then θ is oscillating. However, its is not possible to tell from GNI alone if the oscillations are noise-driven or curvature-driven.

In Appendix D of [2], we see intuitive explanations as well as a proof for this. Let $\theta^\star$ be an optimum our algorithm is oscillating around because of the random noise of SGD. Let $H := H(\theta^\star)$ be the Hessian at the optimum and let $\Delta := (\theta - \theta^\star)$. By Taylor expanding $\nabla L(\theta)$ and plugging the result into the denominator of GNI we get:

$$\nabla L(\theta) \approx \nabla L(\theta^\star) + H(\theta - \theta^\star) = H\Delta$$

$$\|\nabla L(\theta)\|^2 = (H\Delta)^\top(H\Delta) = tr((H\Delta)^\top(H\Delta))$$

Then we average over the iterations (not the batches) by taking the expectation with respect to the stationary distribution of θ. After using the cycle property and linearity of the trace as well as the symmetry of H, we obtain

$$\mathbb{E}[\|\nabla L(\theta)\|^2] = tr(HH\mathbb{E}[\Delta\Delta^\top]) = tr(H\Sigma_\theta H) \tag{49}$$

$\Sigma_\theta := \mathbb{E}[\Delta\Delta^\top]$ is exactly the covariance matrix of θ, describing the shape of the cloud of points the algorithm visits around the optimum.

In Appendix E, [2] uses discrete Lyapunov analysis to relate the parameter noise Σ_θ and the gradient noise at the minimum $\Sigma_{\nabla L}(\theta^\star)$. When the oscillations stabilize around the minimum,

$\Sigma_\theta \approx \frac{\eta}{2} H^{-1} \Sigma_{\nabla L}(\theta^\star)$ holds. Substituting this into our equation gives

$$\mathbb{E}[\|\nabla L(\theta)\|^2] \approx tr(H(\frac{\eta}{2} H^{-1} \Sigma_{\nabla L}(\theta^\star))H) = \frac{\eta}{2} tr(H \Sigma_{\nabla L}(\theta^\star))$$

For the numerator of GNI, we use that the trace of a scalar is still the scalar combined with the properties of tr used earlier to induce

$$\nabla L_{\mathcal{B}}(\theta)^\top H(\theta) \nabla L_{\mathcal{B}}(\theta) = tr(H(\theta) \nabla L_{\mathcal{B}}(\theta) \nabla L_{\mathcal{B}}(\theta)^\top)$$

$$\mathbb{E}_{\mathcal{B}\sim\mathcal{P}}\left[\nabla L_{\mathcal{B}}(\theta)^\top H(\theta) \nabla L_{\mathcal{B}}(\theta)\right] = tr(H(\theta) \mathbb{E}_{\mathcal{B}\sim\mathcal{P}}[\nabla L_{\mathcal{B}}(\theta) \nabla L_{\mathcal{B}}(\theta)^\top])$$

We can decompose the second moment of the gradients into the squared mean gradient plus the covariance:

$$\mathbb{E}_{\mathcal{B}\sim\mathcal{P}}[\nabla L_{\mathcal{B}}(\theta) \nabla L_{\mathcal{B}}(\theta)^\top] = \nabla L(\theta) \nabla L(\theta)^\top + \Sigma_{\nabla L}(\theta)$$

The squared norm of the full-batch gradient $\|\nabla L(\theta)\|^2$ scales with $\mathcal{O}(\eta)$, but the noise covariance $\Sigma_{\nabla L}$ is only of order $\mathcal{O}(1)$. This means for small step sizes, gradient noise is much larger than the squared norm. We also use an assumption that the noise covariance $\Sigma_{\nabla L}(\theta)$ is approximately constant for all θ within the small neighborhood around $\theta^\star$. Therefore, we can approximate the second moment at θ using the covariance at the optimum:

$$\mathbb{E}_{\mathcal{B}\sim\mathcal{P}}[\nabla L_{\mathcal{B}}(\theta) \nabla L_{\mathcal{B}}(\theta)^\top] \approx \mathbb{E}_{\mathcal{B}\sim\mathcal{P}}[\nabla L_{\mathcal{B}}(\theta^\star) \nabla L_{\mathcal{B}}(\theta^\star)^\top] = \Sigma_{\nabla L}(\theta^\star)$$

Plugging this in the previous equation gives us

$$\mathbb{E}_{\mathcal{B}\sim\mathcal{P}}\left[\nabla L_{\mathcal{B}}(\theta)^\top H(\theta) \nabla L_{\mathcal{B}}(\theta)\right] = tr(H \Sigma_{\nabla L}(\theta^\star))$$

Plugging both results into the GNI equation shows

$$GNI \approx \frac{tr(H \Sigma_{\nabla L}(\theta^\star))}{\frac{\eta}{2} tr(H \Sigma_{\nabla L}(\theta^\star))} = \frac{2}{\eta} \tag{50}$$

In the proof in Appendix D.2 from Andreyev and Beneventano [2], we assumed to be close to the minima to show oscillations mean $GNI \approx \frac{2}{\eta}$. However, numerical experiments by both Lee and Jang [21] and Andreyev and Beneventano [2] demonstrate that during neural network training, type-1 oscillations are present from the start. After only a few steps of SGD algorithm, GNI already jumps to $2/\eta$. A possible explanation for this is that for a small step size, the oscillations equilibrate faster than the landscape evolves, leading to a stationary distribution. The system behaves locally similar to if it were at a minimum. The oscillations happen in only few iterations, while $H(\theta)$ and $\Sigma_{\nabla L}$ change quite slowly and can be seen as nearly constant. This means the Lyapunov analysis from the Appendix E still holds up during most of training.

The discrete Lyapunov analysis demonstrates that GNI is equal to $2/\eta$ for any stable oscillating system, including the curvature based Type-2 oscillations. Type-2 oscillations do not emerge naturally in the case of quadratic objectives, they are more unique to complex settings like neural network optimization.

13.3 Instability Criterions

To treat type-2 oscillations and the resulting behavior of the optimizer with the appropriate mathematical rigor, we introduce the notion of an instability criterion. Let $U \subset \Theta$ be the region of the parameter space where we approximate the loss locally, similar to the region where the quadratic approximation holds in Definition 7. An instability criterion is a scalar function $C(\theta)$ and a threshold τ where $C(\theta) > \tau$ is sufficient to force divergence from the region U.

This means we have a stable regime ($C(\theta) \leq \tau$) where the optimization step remains within the region where the local approximation holds and an unstable regime ($C(\theta) > \tau$) where the optimizer overshoots, leading to a catapult: a distinct spike in the loss value occurs and θ jumps out of the current local region. If we have approximate equality ($C(\theta) \approx \tau$), we call the instability criterion "saturated".

Definition 8 (Instability Criterion, taken directly from [2])**.** *Consider a training algorithm (a discrete-time dynamical system) $(\theta_t)_{t\geq 0}$ on a parameter space $\Theta \subseteq \mathbb{R}^d$ with fixed hyperparameters h (e.g. learning rate, batch size). Let $U \subseteq \Theta$ be an open set (typically, a region where a given local approximation of the loss is trusted). Let $C : U \to \mathbb{R}$ and $\tau \in \mathbb{R}$. We say that C is a valid instability criterion with threshold τ for the algorithm on U if the following holds:*

$$C(\theta_0) > \tau \quad \Longrightarrow \quad (\theta_t)_{t\geq 0} \text{ leaves every compact subset of } U.$$

That is, for any compact $K \subset U$ containing θ_0, there exists a finite time T such that $\theta_T \notin K$. We say that the instability criterion C is saturated at θ if $C(\theta)$ is (approximately) equal to τ; in practice, up to an $O(\eta)$ tolerance.

For full-batch gradient descent, the instability criterion is exactly the largest eigenvalue of the Hessian and the threshold is $2/\eta$

$$C_{\text{GD}}(\theta) = \lambda_1(H(\theta))$$
$$\tau_{\text{GD}}(\eta) = 2/\eta$$

If $C_{\text{GD}}(\theta) > \tau_{\text{GD}}(\eta)$, gradient descent becomes unstable and catapults to other regions of the loss landscape. Because of the self stabilization dynamics discussed in the Section 9, $\lambda_1(H(\theta))$ can not grow much larger than $2/\eta$. The inequality is saturated ($C(\theta) \approx 2/\eta$) and the optimizer cannot enter regions that are sharper than this threshold.

For SGD, determining whether an instability criterion is saturated is more complicated. One possibility is to do this empirically by observing if catapults and loss spikes occur when the

hyperparameters are changed. Andreyev and Beneventano [2] shows in Appendix B.2 that the following three observations:

1. **breaking the instability criterion**,
2. **the optimizer experiencing a catapult**,
3. **the loss exhibiting a sudden spike**

are all just indications of the same phenomenon: the optimizer diverges on the local quadratic approximation U. In Lemma 2, they show that under the following three assumptions an instability criterion indicates catapult behavior when certain hyperparameters are changed. Let $C(\theta; h)$ depend on both θ and the hyperparameters $h = (\eta, \|\mathcal{B}\|)$ and let $\tau(h)$ depend on just η and the batch size $(\eta, \|\mathcal{B}\|)$. These are the assumptions:

1. $C(\cdot; h)$ is a **valid instability criterion** for the quadratic model on U.
2. $C(\theta; h)$ is **monotone in a destabilizing direction** of h. A destabilizing direction refers to a direction where change in hyperparameters h moves the optimizer toward a more unstable regime. For SGD, this happens when increasing η or decreasing $\|\mathcal{B}\|$. This monotonicity assumption of $C(\theta; h)$ means that for a destabilizing direction of hyperparameters, the gap $C(\theta; h) - \tau(h)$ increases.
3. for a fixed h_0 there is a t such that $C(\theta_t; h_0)$ **is saturated**.

With these assumptions in place, Lemma 2 in Andreyev and Beneventano [2] says that any sufficiently small destabilizing perturbation of h_0 results in hyperparameters h where $C(\theta_t; h) > \tau(h)$. By the instability criterion 8, this means if we calculated the trajectory of the quadratic approximation of the loss function starting from θ_t, we would leave every compact subset of U_t in finite time. The algorithm catapults at time t.

A catapult also has a big influence on the loss. Catapulting out of the locally quadratic region generally results in a loss spike of sufficient size. Formally, on the quadratic model, this corresponds to a spike of arbitrarily large relative size. In practice, because of self stabilization, the loss spike is not arbitrarily large, so we call it a loss spike of sufficient size if the loss increase exceeds the standard fluctuations inherent to stochastic optimization.

Now it is time to find an appropriate instability criterion for SGD. It is important that the criterion saturates during SGD training. We need to include high enough cumulants of the Hessian to find a metric that hits its limit before other measures. We also need to make sure the criterion is easily computable in higher dimensions. Andreyev and Beneventano [2] defined a measure called batch sharpness that fulfills all of these criteria.

13.4 Batch Sharpness

For a given batch $\mathcal{B}$, we can calculate the curvature along the direction SGD took a step in a similar way we did for GD in the interaction-aware sharpness. The step sharpness (SS) is:

$$SS_{\mathcal{B}}(\theta) = \frac{\nabla L_{\mathcal{B}}(\theta)^\top H_{\mathcal{B}}(\theta) \nabla L_{\mathcal{B}}(\theta)}{\|\nabla L_{\mathcal{B}}(\theta)\|^2} \tag{51}$$

Similar to the IAS (45), step sharpness is a measure of directed curvature. We do not look for the greatest curvature in any direction, but only the curvature in the direction of the gradient of a specific batch $\nabla L_{\mathcal{B}}(\theta)$.

To get a metric independent of the randomly chosen current batch $\mathcal{B}$, we take the expected value over all batches. We define the batch sharpness (BS) [2]:

$$BS(\theta) := \mathbb{E}_{\mathcal{B}\sim\mathcal{P}}\left[\frac{\nabla L_{\mathcal{B}}(\theta)^\top H_{\mathcal{B}}(\theta) \nabla L_{\mathcal{B}}(\theta)}{\|\nabla L_{\mathcal{B}}(\theta)\|^2}\right] \tag{52}$$

The batch sharpness can be derived heuristically by Taylor expanding $L_B(\theta_{t+1})$ and using similiar reasoning to how we derived the IAS in Section 12. Note that [21] defined the IAS for SGD in a really similar fashion, but used the full Hessian H instead of $H_{\mathcal{B}}$. It is denoted as $\|H\|_{S_{\|\mathcal{B}\|}}$, where $S_{\|\mathcal{B}\|} = \mathbb{E}_{\mathcal{B}\sim\mathcal{P}}[\nabla L_{\mathcal{B}} \nabla L_{\mathcal{B}}^\top]$

$$\|H\|_{S_{\|\mathcal{B}\|}} = \frac{tr(HS_{\|\mathcal{B}\|})}{tr(S_{\|\mathcal{B}\|})} = \frac{\mathbb{E}_{\mathcal{B}\sim\mathcal{P}}\left[\nabla L_{\mathcal{B}}(\theta)^\top H(\theta) \nabla L_{\mathcal{B}}(\theta)\right]}{\mathbb{E}_{\mathcal{B}\sim\mathcal{P}}\left[\|\nabla L_{\mathcal{B}}(\theta)\|^2\right]} \tag{53}$$

The main difference between the definition is that batch sharpness uses $H_{\mathcal{B}}(\theta)$, while the IAS is defined with the averaged, full dataset Hessian $H(\theta)$. Andreyev and Beneventano [2] argue that batch sharpness is a better notion of curvature along the step of SGD. Using the averaged Hessian $H(\theta)$ does not accurately represent the landscape of the mini-batch loss. The paper even gives an example that the stability of SGD can not depend solely on full-batch Hessian in Appendix L. Another difference is the expectation being taken separately for the nominator and denominator in the IAS.

When using SGD, the curvature-driven oscillations that characterize the EoS are not triggered when full-batch sharpness reaches $2/\eta$, but rather when batch sharpness reaches $2/\eta$, independently of the batch size. The regime is termed edge of stochastic stability (EoSS) [2]. Like in the GD case, the SGD algorithm does not diverge when reaching the EoSS and loss still decreases over longer periods of time. The batch sharpness hovers right at or slightly above $2/\eta$. An empirical verification for different learning rates can be seen in Figure 13.

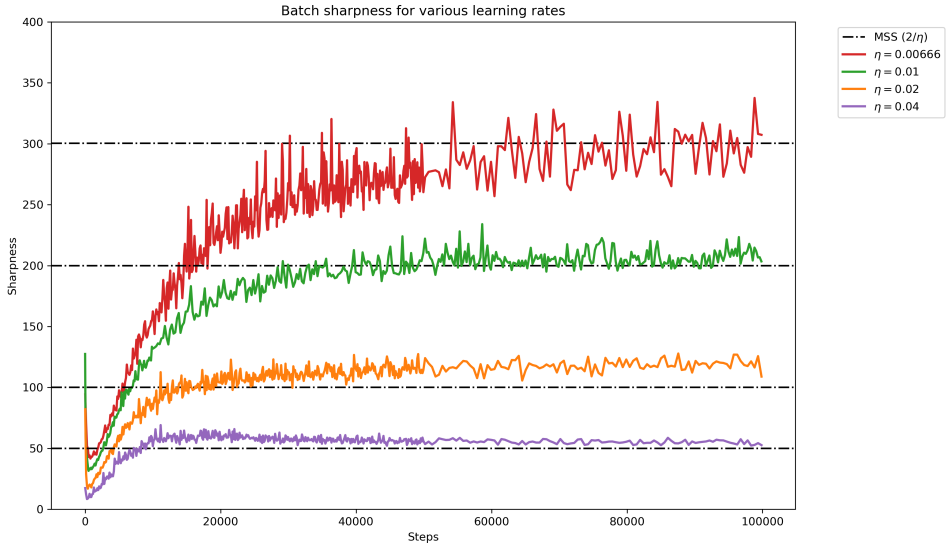


Figure 13: For different learning rates, the batch sharpness stabilizes at different MSS $2/\eta$. Interestingly, smaller learning rates result in larger oscillations ion the sharpness.

13.4.1 Random Catapults

The EoSS regime is in general more chaotic than the EoS regime. This can lead to the catapult effect we described earlier happening, even without the hyperparameters being changed. If the random sampling of mini-batches results in the sharpness being really high for a sequence of mini-batches, a catapult is triggered. Like mentioned before, this means the algorithm overshoots, launching the optimizer into a new, flatter region. This new point generally has higher loss and lower batch sharpness, so the algorithm goes back to progressive sharpening and simultaneously decreasing the loss again.

13.4.2 Batch Sharpness and IAS

We compare batch sharpness and IAS to see what statistical phenomena happen when we average over the batches. Note that full-batch gradient and full-batch Hessian can be seen as the average versions over all batches

$$\nabla L(\theta) = \mathbb{E}_{\mathcal{B}\sim\mathcal{P}}(L_{\mathcal{B}}(\theta)), \quad H(\theta) = \mathbb{E}_{\mathcal{B}\sim\mathcal{P}}(H_{\mathcal{B}}(\theta)) \tag{54}$$

We call the sharpness of a single batch $S_{\mathcal{B}}(\theta)$, not to be confused with the step sharpness $SS_{\mathcal{B}}(\theta)$.
In Section L.2 of [2] the authors constructed a toy example of a diagonal linear network trained on two samples. Each data point results in a loss surface with high sharpness, but the eigenvectors corresponding to these largest eigenvalues are orthogonal. When averaging over both samples, these sharpest directions for one data point gets dimmed because the loss surface of

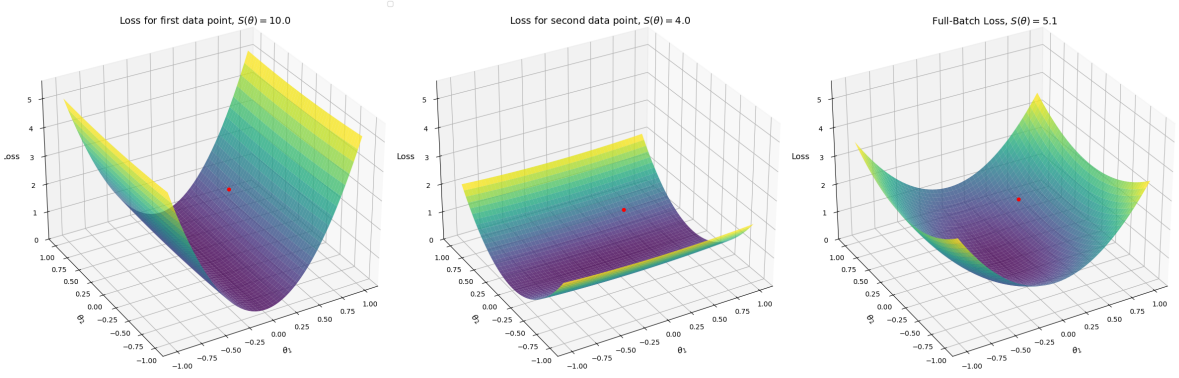


Figure 14: The average of sharpness of the loss surface for the first two samples is $S(\theta) = 7.0$, while the sharpness of the averaged loss surface is $S(\theta) = 5.1$.

the other data point is quite flat in that direction. Therefore the (full-batch) sharpness $S(\theta)$ over both samples is lower than the mean of sharpness of the two batches $S_{\mathcal{B}}(\theta)$, as can be seen in Figure 14. From this we can follow that the 'averaged sharpness' is higher than the 'sharpness of averages'.

$$\underbrace{\mathbb{E}_{\mathcal{B}\sim\mathcal{P}}(\lambda_1(H_{\mathcal{B}}(\theta)))}_{\text{averaged sharpness}} \geq \underbrace{\lambda_1(\mathbb{E}_{\mathcal{B}\sim\mathcal{P}}(H_{\mathcal{B}}(\theta)))}_{\text{sharpness of averages}}$$

If the eigenvectors corresponding to these largest eigenvalues were aligned with each other, $S(\theta)$ and the average of $S_{\mathcal{B}}(\theta)$ would be the same.

In high-dimensional neural networks, the mini-batch Hessians are generally not aligned, resulting again in a gap between averaged sharpness and sharpness averages. When calculating batch sharpness, we first calculate the directional sharpness for each individual mini-batch like in (51), and then calculate the expected value over all batches. This results in batch sharpness generally being much higher than its full-batch counterpart IAS (45).

13.4.3 Batch Sharpness and Full-Batch Sharpness

Empirical experiments show that the relation between batch sharpness $BS(\theta)$ and sharpness $S(\theta)$ is similar to that of batch sharpness and IAS. Batch sharpness, which averages over Hessians, is generally much higher than sharpness. When batch sharpness reaches the $2/\eta$ boundary, $S(\theta)$ stops increasing as well and settles at a value generally much lower than $2/\eta$. The value $S(\theta)$ settles at is dependent on the batch size, where lower batch sizes lead to lower values of sharpness stabilization. Higher batch sizes are more similar to the full-batch regime (where sharpness and batch sharpness both stabilize round $2/\eta$), so they generally have higher sharpness when batch sharpness reaches $2/\eta$ [2].

An interesting phenomenon occurs when the hyperparameters step size η or batch size $\|\mathcal{B}\|$ are changed at some point during the training process. When we increase the step size η during

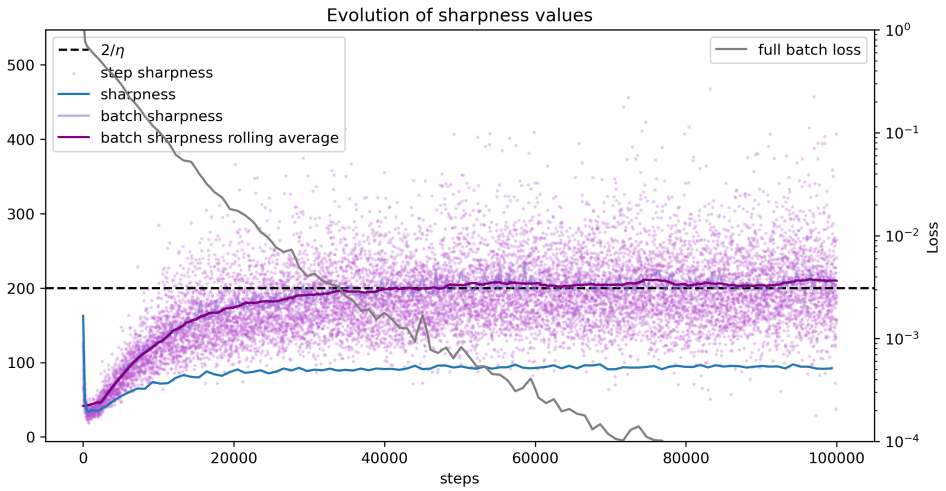


Figure 15: The batch sharpness (purple) hovers at the MSS while the standard sharpness (blue) stabilized at a much lower value.

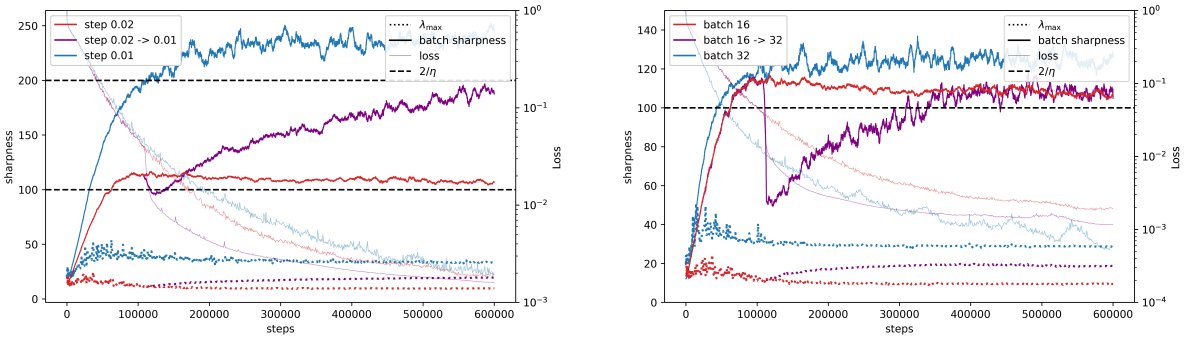
training, the maximum stable sharpness (MSS) equal to $2/\eta$ instantly drops to a lower value. This results in the batch sharpness dropping until it reaches a level close to the new MSS. The full-batch sharpness $S(\theta)$ also drops to a lower value during this process.

If we do the opposite and decrease η during training, the new MSS is higher, so batch sharpness enters the progressive sharpening phase again. Full-batch sharpness also grows, but stabilizes at a way lower value than if we started with the decreased step size η from the start, see Figure 16. This shows again that $S(\theta)$ can not be the deciding metric that governs the edge of stability in the SGD case. If the batch size is increased, larger batches lead to reduced variance of said batches resulting in the batch sharpness dropping significantly. A new phase of progressive sharpening starts. Like in the case where we decreased step size, the full-batch sharpness $S(\theta)$ increases as well in this stage, but stabilizes at a lower value than if the training were started with the higher batch size, see Figure 16. Decreasing the batch size leads to a sudden spike in batch sharpness, which then gradually lowers again. The sharpness stabilizes at a lower level than it hovered at before [2].

Changing the batch size does not change the full-batch loss landscape at an instant, but still causes strong changes in the behavior of the optimization process. This indicates that the stability of SGD is not governed by the full-batch loss landscape, and rather the mini-batch landscape.

13.5 Stability

There are multiple theoretical results and numerical findings that show that the stability of SGD is not governed by the standard sharpness $S(\theta)$ [2]. Instead, batch sharpness is a good indicator



(a) Learning rate decrease

(b) Batch size increase

Figure 16: Analysis of changing hyperparameters during training. (a) shows the effect of decreasing the learning rate during training, which results in the optimizer reentering the progressive sharpening phase. Notably, the standard sharpness (λ_{max}) stabilizes at a lower value than if the training was started with the lower learning rate from the beginning. (b) shows the effect of increasing batch size during training, which suddenly lowers the batch sharpness. The optimizer enters the progressive sharpening phase again while the behavior of standard sharpness is similar to (a). Figure is taken from [2]

of stability. Theorem 1 in [2] says that if for an $\epsilon > 0$:

$$BS(\theta_t) > \frac{2+\epsilon}{\eta} \tag{55}$$

Then, the expected squared norm of the mini-batch gradients grows by a factor of at least $(1+\epsilon)^2$ in the next step:

$$\mathbb{E}_{\mathcal{B}}\left[\frac{\|\nabla L_{\mathcal{B}}(\theta_{t+1})\|^2}{\|\nabla L_{\mathcal{B}}(\theta_t)\|^2}\right] > (1+\epsilon)^2 \tag{56}$$

If the batch sharpness remains above the threshold, the gradients of the loss L will explode, which results in the trajectory becoming unstable on the local quadratic approximation. Andreyev and Beneventano [2] were not able to prove the reverse statement: We do not know if instability implies that the batch sharpness exceeds $2/\eta$.

In the following, we will provide the rigorous statement and proof of Theorem 1 in [2]. Compared to the original, this proof is explained in greater detail, a Lipschitz constant was improved and an error in the derivation of the minimum unstable window was corrected.

For each mini-batch $\mathcal{B}_i$, let $L_{\mathcal{B}_i} : \mathbb{R}^d \to \mathbb{R}$ be twice differentiable and have a positive semidefinite Hessian $H_{\mathcal{B}_i}$. We assume the loss function for a single batch to be quadratic, which makes $H_{\mathcal{B}_i}$ a constant matrix. This "constant curvature assumption" is used often to study local stability around a minimum, which is $c_i \in \mathbb{R}^d$ in our case. We use $G_i(\theta)$ to refer to the gradients.

$$G_i(\theta) := \nabla L_{\mathcal{B}_i}(\theta) = H_{\mathcal{B}_i}(\theta - c_i)$$

Table 2: Overview of Sharpness Dynamics under Hyperparameter Changes

Action	**MSS ($2/\eta$)**	**Batch Sharpness**	**Full-Batch $S(\theta)$**
Increase η	Decreases	Drops to new MSS gradually	Drops to lower value
Decrease η	Increases	Enters progressive sharpening again	Increases
Increase $\|\mathcal{B}\|$	No Change	Drops and enters progressive sharpening	Increases
Decrease $\|\mathcal{B}\|$	No Change	Spikes and lowers gradually	Stabilizes at lower level

We use the step sharpness $r_i(\theta)$ (equal to the Rayleigh quotient of the Hessian of $L_{\mathcal{B}_i}(\theta)$ along the direction of its gradient) and we define the squared directional curvature $q_i(\theta_t)$.

$$r_i(\theta_t) := \frac{G_i(\theta_t)^\top H_{\mathcal{B}_i} G_i(\theta_t)}{\|G_i(\theta_t)\|^2} \in [0, \infty)$$
$$q_i(\theta_t) := \frac{G_i(\theta_t)^\top H_{\mathcal{B}_i}^2 G_i(\theta_t)}{\|G_i(\theta_t)\|^2} = \frac{\|H_{\mathcal{B}_i} G_i(\theta_t)\|^2}{\|G_i(\theta_t)\|^2}$$

$r_i(\theta_t)$ is the directional curvature along the gradient for a batch B_i and $q_i(\theta_t)$ is the squared norm of the gradients rate of change, a measure for the total intensity the Hessian applies to the gradient vector. We randomly sample batches out of a distribution $\mathcal{B}_i \sim \mathcal{P}$. Let $\mathbb{E}_i[\cdot]$ be shorthand for $\mathbb{E}_{\mathcal{B}_i \sim \mathcal{P}}[\cdot]$, then the batch sharpness is defined like this:

$$BS(\theta_t) = \mathbb{E}_i[r_i(\theta_t)]$$

At each t, we draw a random mini-batch $j_t \sim \mathcal{P}$ i.i.d.. Let $\mathcal{F}_t$ be the σ-algebra generated by the starting point θ_0 and the sequence of random mini-batches up to time $t-1$. $\mathcal{F}_t$ is called a filtration, it contains all the information about the trajectory of the path the optimizer took up to time $t-1$.

Now we want to calculate the expected growth of the gradient norm after one SGD step, assuming our locally quadratic model. We define the on-batch factor

$$R_t := \mathbb{E}_{j_t}\left[\frac{\|G_{j_t}(\theta_{t+1})\|^2}{\|G_{j_t}(\theta_t)\|^2}\middle|\mathcal{F}_t\right] \tag{57}$$

The factor quantifies how much the squared gradient norm of the current batch j_t changes if we re-evaluate at the updated parameters θ_{t+1} after a step has been taken in the direction of $G_{j_t}(\theta_t)$. A value of $R_t < 1$ indicates a stable decline toward the batch's local minimum, whereas $R_t > 1$ signifies an overshoot where the gradient norm increases.

Proposition 1 (Multiplicative Explosion if Batch Sharpness $\geq 2/\eta$)**.** *Assume the mini-batch losses follow the local quadratic model, and that the Hessians and expected squared gradients are uniformly bounded (ess sup$_{\mathcal{B}\in\mathcal{P}}\|H_{\mathcal{B}}\| < \infty$ and $\mathbb{E}_i[\|G_i(\theta)\|^2] < \infty$). Then, for all t:*

$$R_t \geq \left(1 - \eta BS(\theta_t)\right)^2$$

For an initial step $t_0 \in \mathbb{N}$, window size $T \geq 1$, and $\epsilon > 0$, if

$$BS(\theta_t) \geq \frac{2+\epsilon}{\eta}, \quad \text{for all } t \in \{t_0, \ldots, t_0 + T - 1\}$$

Then the expectation of the product of the on-batch factors grows exponentially.

$$\mathbb{E}\left[\prod_{s=t_0}^{t_0+T-1} \frac{\|G_{j_s}(\theta_{s+1})\|^2}{\|G_{j_s}(\theta_s)\|^2}\right] \geq (1+\epsilon)^{2T}$$

Proof. Fix a time t with chosen batch $\mathcal{B}_i$. Then

$$\begin{aligned} G_i(\theta_{t+1}) &= H_{\mathcal{B}_i}(\theta_{t+1} - c_i) \\ &= H_{\mathcal{B}_i}(\theta_t - \eta G_i(\theta_t) - c_i) \\ &= H_{\mathcal{B}_i}(\theta_t - c_i) - \eta H_{\mathcal{B}_i} G_i(\theta_t) \\ &= (I - \eta H_{\mathcal{B}_i}) G_i(\theta_t) \end{aligned}$$

Therefore

$$\begin{aligned} \frac{\|G_i(\theta_{t+1})\|^2}{\|G_i(\theta_t)\|^2} &= \frac{\|(I - \eta H_{\mathcal{B}_i})G_i(\theta_t)\|^2}{\|G_i(\theta_t)\|^2} \\ &= \frac{G_i(\theta_t)^\top \left(I - 2\eta H_{\mathcal{B}_i} + \eta^2 H_{\mathcal{B}_i}^2\right) G_i(\theta_t)}{G_i(\theta_t)^\top G_i(\theta_t)} \\ &= 1 - 2\eta \frac{G_i(\theta_t)^\top H_{\mathcal{B}_i} G_i(\theta_t)}{G_i(\theta_t)^\top G_i(\theta_t)} + \eta^2 \frac{G_i(\theta_t)^\top H_{\mathcal{B}_i}^2 G_i(\theta_t)}{G_i(\theta_t)^\top G_i(\theta_t)} \\ &= 1 - 2\eta r_i(\theta_t) + \eta^2 q_i(\theta_t) \end{aligned} \tag{58}$$

The Cauchy-Schwarz inequality states that for any two vectors w, z

$$(w^\top z)^2 \leq \|w\|^2 \|z\|^2$$

By setting $w = G_i(\theta_t)$ and $z = H_{\mathcal{B}_i} G_i(\theta_t)$ we see that

$$\begin{aligned} \frac{\left(G_i(\theta_t)^\top (H_{\mathcal{B}_i} G_i(\theta_t))\right)^2}{\|G_i(\theta_t)\|^4} &\leq \frac{\|G_i(\theta_t)\|^2}{\|G_i(\theta_t)\|^2} \frac{\|H_{\mathcal{B}_i} G_i(\theta_t)\|^2}{\|G_i(\theta_t)\|^2} \\ r_i(\theta_t)^2 &\leq q_i(\theta_t) \end{aligned} \tag{59}$$

Plugging this into equation (58) gives us

$$\frac{\|G_i(\theta_{t+1})\|^2}{\|G_i(\theta_t)\|^2} \geq 1 - 2\eta r_i(\theta_t) + \eta^2 r_i(\theta_t)^2 = (1 - \eta r_i(\theta_t))^2$$

Since this equation is true for any batch $\mathcal{B}_i$, it also holds for the expected value:

$$R_t = \mathbb{E}_i \left[\frac{\|G_i(\theta_{t+1})\|^2}{\|G_i(\theta_t)\|^2} \right] \geq \mathbb{E}_i \left[(1 - \eta r_i(\theta_t))^2 \right]$$

Jensen's inequality states that for any random variable X and any convex function ϕ

$$\phi(\mathbb{E}[X]) \leq \mathbb{E}[\phi(X)]$$

Applying Jensen's inequality for $\phi(X) = X^2$:

$$\mathbb{E}_i \left[(1 - \eta r_i(\theta_t))^2 \right] \geq \mathbb{E}_i \left[(1 - \eta r_i(\theta_t)) \right]^2 = (1 - \eta \mathbb{E}_i[r_i(\theta_t)])^2 = (1 - \eta BS(\theta_t))^2$$

In Proposition 1, we assume $BS(\theta_t) \geq \frac{2+\epsilon}{\eta}$ for $t \in \{t_0, \ldots, t_0 + T - 1\}$. An easy algebraic computation confirms that this means $(1 - \eta BS(\theta_t))^2 \geq (1 - \eta \frac{2+\epsilon}{\eta})^2$. For R_t we get:

$$R_t \geq (1 - \eta \frac{2 + \epsilon}{\eta})^2 = (1 + \epsilon)^2 > 1 + 2\epsilon \text{ for } t \in \{t_0, \ldots, t_0 + T - 1\} \tag{60}$$

Now we want to prove that the expectation of the product of the gradient norm ratios grows exponentially. For this, we need the tower property:

$$\mathbb{E}[X] = \mathbb{E}\big[\mathbb{E}[X \mid \mathcal{F}_t]\big]$$

In the product, we isolate the final step $t = t_0 + T - 1$ and condition it on $\mathcal{F}_{t_0+T-1}$. The conditional expected value we get is exactly R_{t_0+T-1}.

$$\begin{aligned}
\mathbb{E}\left[\prod_{t=t_0}^{t_0+T-1} \frac{\|G_{j_t}(\theta_{t+1})\|^2}{\|G_{j_t}(\theta_t)\|^2} \right] &= \mathbb{E}\left[\left(\prod_{t=t_0}^{t_0+T-2} \frac{\|G_{j_t}(\theta_{t+1})\|^2}{\|G_{j_t}(\theta_t)\|^2} \right) \mathbb{E}\left[\frac{\|G_{j_{t_0+T-1}}(\theta_{t_0+T})\|^2}{\|G_{j_{t_0+T-1}}(\theta_{t_0+T-1})\|^2} \middle| \mathcal{F}_{t_0+T-1} \right] \right] \\
&= \mathbb{E}\left[\left(\prod_{t=t_0}^{t_0+T-2} \frac{\|G_{j_t}(\theta_{t+1})\|^2}{\|G_{j_t}(\theta_t)\|^2} \right) R_{t_0+T-1} \right]
\end{aligned}$$

Since the batch sharpness exceeds the critical threshold for all steps in this interval, we can plug in $R_t \geq (1 + \epsilon)^2$:

$$\mathbb{E}\left[\prod_{t=t_0}^{t_0+T-1} \frac{\|G_{j_t}(\theta_{t+1})\|^2}{\|G_{j_t}(\theta_t)\|^2} \right] \geq (1 + \epsilon)^2 \cdot \mathbb{E}\left[\prod_{t=t_0}^{t_0+T-2} \frac{\|G_{j_t}(\theta_{t+1})\|^2}{\|G_{j_t}(\theta_t)\|^2} \right]$$

Doing this iteratively proves our statement.

$$\mathbb{E}\left[\prod_{t=t_0}^{t_0+T-1} \frac{\|G_{j_t}(\theta_{t+1})\|^2}{\|G_{j_t}(\theta_t)\|^2}\right] \geq (1+\epsilon)^{2T} \tag{61}$$

Now we need to show that in the quadratic setting, batch sharpness actually stays above $2/\eta$ once it crossed that border. We assume the gradient norm and the Hessian to have a certain minimum and maximum norm

$$\|G_i(\theta)\| \geq g_{\min} > 0, \quad \|H_{\mathcal{B}_i}\| \leq \Lambda$$

This shows that $G_i(\theta) = H_{\mathcal{B}_i}(\theta - c_i)$ is Λ-Lipschitz. We want to show that $r_i(\theta)$ is also Lipschitz. For this, we define $v_i(\theta) := \frac{G_i(\theta)}{\|G_i(\theta)\|}$.

$$|r_i(\theta_1) - r_i(\theta_2)| = \left|v(\theta_1)^\top H_{\mathcal{B}_i} v(\theta_1) - v(\theta_2)^\top H_{\mathcal{B}_i} v(\theta_2)\right|$$

We add and subtract $v(\theta_1)^\top H_{\mathcal{B}_i} v(\theta_2)$ inside the absolute value, factor the expression, and use the Cauchy-Schwarz inequality:

$$\begin{aligned} |r_i(\theta_1) - r_i(\theta_2)| &\leq \left|v(\theta_1)^\top H_{\mathcal{B}_i}\big(v(\theta_1) - v(\theta_2)\big)\right| + \left|\big(v(\theta_1) - v(\theta_2)\big)^\top H_{\mathcal{B}_i} v(\theta_2)\right| \\ &\leq \|v(\theta_1)\|\|H_{\mathcal{B}_i}\|\|v(\theta_1) - v(\theta_2)\| + \|v(\theta_1) - v(\theta_2)\|\|H_{\mathcal{B}_i}\|\|v(\theta_2)\| \\ &= 2\Lambda\|v(\theta_1) - v(\theta_2)\| \end{aligned} \tag{62}$$

For determining the Lipschitz constant in terms of θ, we add and subtract $\frac{G_i(\theta_2)}{\|G_i(\theta_1)\|}$ inside the norm $\|v(\theta_1) - v(\theta_2)\|$.

$$\|v_i(\theta_1) - v_i(\theta_2)\| = \left\|\frac{G_i(\theta_1) - G_i(\theta_2)}{\|G_i(\theta_1)\|} + G_i(\theta_2)\left(\frac{1}{\|G_i(\theta_1)\|} - \frac{1}{\|G_i(\theta_2)\|}\right)\right\|$$

We use the triangle inequality ($\|a+b\| \leq \|a\| + \|b\|$) to separate the two terms:

$$\begin{aligned} \|v_i(\theta_1) - v_i(\theta_2)\| &\leq \frac{\|G_i(\theta_1) - G_i(\theta_2)\|}{\|G_i(\theta_1)\|} + \|G_i(\theta_2)\|\left|\frac{\|G_i(\theta_2)\| - \|G_i(\theta_1)\|}{\|G_i(\theta_1)\|\|G_i(\theta_2)\|}\right| \\ &= \frac{\|G_i(\theta_1) - G_i(\theta_2)\|}{\|G_i(\theta_1)\|} + \frac{\big|\|G_i(\theta_2)\| - \|G_i(\theta_1)\|\big|}{\|G_i(\theta_1)\|} \end{aligned}$$

We use the reverse triangle inequality ($\big|\|a\| - \|b\|\big| \leq \|a-b\|$) on the numerator of the second term:

$$\frac{\big|\|G_i(\theta_1)\| - \|G_i(\theta_2)\|\big|}{\|G_i(\theta_1)\|} \leq \frac{\|G_i(\theta_1) - G_i(\theta_2)\|}{\|G_i(\theta_1)\|}$$

Combining this we get:

$$\|v_i(\theta_1) - v_i(\theta_2)\| \leq \frac{2}{\|G_i(\theta_1)\|}\|G_i(\theta_1) - G_i(\theta_2)\|$$

Using assumptions $\|G_i(\theta)\| \geq g_{\min}$ and the Λ-Lipschitz continuity of $G_i(\theta)$, we get:

$$\|v_i(\theta_1) - v_i(\theta_2)\| \leq \frac{2}{g_{\min}} \cdot \Lambda\|\theta_1 - \theta_2\| = \frac{2\Lambda}{g_{\min}}\|\theta_1 - \theta_2\|$$

This together with eq (62) shows that $r_i(\theta)$ is Lipschitz with constant $L_S = \frac{2\Lambda^2}{g_{\min}}$:

$$|r_i(\theta_1) - r_i(\theta_2)| \leq 2\Lambda\|v(\theta_1) - v(\theta_2)\| \leq \frac{2\Lambda^2}{g_{\min}}\|\theta_1 - \theta_2\|$$

Averaging of the batches does not change this bound.

$$|BS(\theta_1) - BS(\theta_2)| \leq L_S\|\theta_1 - \theta_2\| \tag{63}$$

We additionally assume a uniform bound for the length of each step. Let $g_{\max} > 0$:

$$\|\theta_{t+1} - \theta_t\| \leq \eta g_{\max}$$

If for a time t_0:

$$BS(\theta_{t_0}) \geq \frac{2+\epsilon}{\eta}$$

For all k with $0 \leq k \leq T_\star := \lfloor \frac{\epsilon}{2\eta^2 L_S g_{\max}} \rfloor$ batch sharpness stays above $2/\eta$, to be precise. Note that the bound $\lfloor \frac{\epsilon}{2\eta^2 L_S g_{\max}} \rfloor$ was wrongly stated as $\lfloor \frac{\epsilon}{2\eta L_S g_{\max}} \rfloor$ in paper [2].

$$BS(\theta_{t_0+k}) \geq \frac{2+\epsilon/2}{\eta}$$

By unpacking the absolute value in equation (63) and applying the bound of $\|\theta_{t+1} - \theta_t\|$:

$$BS(\theta_{t+1}) \geq BS(\theta_t) - L_S\eta g_{\max}$$

By doing this k times we get:

$$\begin{aligned} BS(\theta_{t_0+k}) &\geq BS(\theta_{t_0}) - kL_S\eta g_{\max} \\ &\geq \frac{2+\epsilon}{\eta} - \frac{\epsilon}{2L_S\eta^2 g_{\max}} L_S\eta g_{\max} \\ &= \frac{2}{\eta} + \frac{\epsilon}{\eta} - \frac{\epsilon}{2\eta} \\ &= \frac{2+\epsilon/2}{\eta} \end{aligned}$$

By applying Proposition (1) with $T = T_\star$ we get:

$$\mathbb{E}\left[\prod_{t=t_0}^{t_0+T_\star-1} \frac{\|G_{j_t}(\theta_{t+1})\|^2}{\|G_{j_t}(\theta_t)\|^2}\right] \geq (1+\epsilon/2)^{2T_\star} \tag{64}$$

This shows there is a window of time where the iterates exponentially grow. A single unstable step triggers a catapult. □

14 Batch Sharpness for Adaptive Optimizers

To the best of my knowledge, the concept of batch sharpness has not yet been extended to adaptive optimizers such as RMSProp and Adam in current literature. [5] covered Adam in the mini-batch setting, but only investigated standard effective sharpness, whereas [4] researched RMSProp in great detail but only in the full-batch case.

14.1 Batch Sharpness for Scalar RMSProp

In this Section, we will derive the batch sharpness for scalar RMSProp. Let $\mathcal{B}_t$ be the mini-batch drawn at time t. For mini-batch scalar RMSProp, the update formula is:

$$\begin{aligned} \nu_t &= \beta_2 \nu_{t-1} + (1-\beta_2)\|\nabla L_{\mathcal{B}_t}(\theta_t)\|^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\nu_t}+\epsilon}\nabla L_{\mathcal{B}_t}(\theta_t) \end{aligned} \tag{65}$$

Similar to the derivation of IAS for standard gradient descent in Section 12, we evaluate local stability using a second-order Taylor expansion. Expanding the mini-batch loss $L_{\mathcal{B}_t}(\theta_{t+1})$ around θ_t gives us:

$$L_{\mathcal{B}_t}(\theta_{t+1}) \approx L_{\mathcal{B}_t}(\theta_t) + \nabla L_{\mathcal{B}_t}(\theta_t)^\top(\theta_{t+1}-\theta_t) + \frac{1}{2}(\theta_{t+1}-\theta_t)^\top H_{\mathcal{B}_t}(\theta_t)(\theta_{t+1}-\theta_t) \tag{66}$$

Now we substitute the scalar RMSProp step $\theta_{t+1} - \theta_t = -\frac{\eta}{\sqrt{\nu_t}+\epsilon}\nabla L_{\mathcal{B}_t}(\theta_t)$:

$$\begin{aligned} L_{\mathcal{B}_t}(\theta_{t+1}) \approx & L_{\mathcal{B}_t}(\theta_t) + \nabla L_{\mathcal{B}_t}(\theta_t)^\top \left(-\frac{\eta}{\sqrt{\nu_t}+\epsilon}\nabla L_{\mathcal{B}_t}(\theta_t)\right) \\ & + \frac{1}{2}\left(-\frac{\eta}{\sqrt{\nu_t}+\epsilon}\nabla L_{\mathcal{B}_t}(\theta_t)\right)^\top H_{\mathcal{B}_t}(\theta_t)\left(-\frac{\eta}{\sqrt{\nu_t}+\epsilon}\nabla L_{\mathcal{B}_t}(\theta_t)\right) \end{aligned} \tag{67}$$

Simplifying the terms, we get:

$$L_{\mathcal{B}_t}(\theta_{t+1}) - L_{\mathcal{B}_t}(\theta_t) \approx -\frac{\eta}{\sqrt{\nu_t}+\epsilon}\|\nabla L_{\mathcal{B}_t}(\theta_t)\|^2 + \frac{\eta^2}{2(\sqrt{\nu_t}+\epsilon)^2}\nabla L_{\mathcal{B}_t}(\theta_t)^\top H_{\mathcal{B}_t}(\theta_t)\nabla L_{\mathcal{B}_t}(\theta_t) \tag{68}$$

For the mini-batch loss to be locally stable for a single step, we want $L_{\mathcal{B}_t}(\theta_{t+1}) - L_{\mathcal{B}_t}(\theta_t) \leq 0$. Rearranging gives us the following condition:

$$\frac{\frac{1}{(\sqrt{\nu_t}+\epsilon)^2}\nabla L_{\mathcal{B}_t}(\theta_t)^\top H_{\mathcal{B}_t}(\theta_t)\nabla L_{\mathcal{B}_t}(\theta_t)}{\frac{1}{\sqrt{\nu_t}+\epsilon}\|\nabla L_{\mathcal{B}_t}(\theta_t)\|^2} \leq \frac{2}{\eta} \tag{69}$$

This simplifies to exactly $1/(\sqrt{\nu_t}+\epsilon)$ times the standard batch sharpness for SGD:

$$\frac{1}{\sqrt{\nu_t}+\epsilon}\frac{\nabla L_{\mathcal{B}_t}(\theta_t)^\top H_{\mathcal{B}_t}(\theta_t)\nabla L_{\mathcal{B}_t}(\theta_t)}{\|\nabla L_{\mathcal{B}_t}(\theta_t)\|^2} \leq \frac{2}{\eta} \tag{70}$$

Now we want to take the expected value of this, but this leads to some problems. The optimizer is no longer memoryless because ν_t depends on the batched gradients of all the steps before. Let $\mathcal{F}_t$ be the σ-algebra generated by the starting parameters θ_0 and ν_0, and the sequence of random mini-batches drawn up to time $t-1$.

To get the effective scalar RMSProp batch sharpness, we need to condition it on the training history $\mathcal{F}_t$. Additionally, because ν_t depends on the current batch $\mathcal{B}_t$, we cannot use it in our expectation. If we assume that β_2 is sufficiently close to 1, ν_t updates rather slowly. We can make the assumption $\nu_{t-1} \approx \nu_t$ and use ν_{t-1} instead of ν_t, which gives us the following effective batch sharpness for scalar RMSProp:

$$\mathrm{BS}_{\mathrm{scRMSP}}(\theta_t, \nu_{t-1}) = \frac{1}{\sqrt{\nu_{t-1}}+\epsilon}\mathbb{E}_{\mathcal{B}_t\sim\mathcal{P}}\left[\frac{\nabla L_{\mathcal{B}_t}(\theta_t)^\top H_{\mathcal{B}_t}(\theta_t)\nabla L_{\mathcal{B}_t}(\theta_t)}{\|\nabla L_{\mathcal{B}_t}(\theta_t)\|^2}\,\middle|\,\mathcal{F}_t\right] \tag{71}$$

14.2 Batch Sharpness for RMSProp

The derivation of batch sharpness for standard RMSProp is very similar. The in the update formulas are:

$$\begin{aligned}\nu_t &= \beta_2\nu_{t-1} + (1-\beta_2)\left(\nabla L_{\mathcal{B}_t}(\theta_t)\right)^{\odot 2}\\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\nu_t}+\epsilon}\odot\nabla L_{\mathcal{B}_t}(\theta_t)\end{aligned} \tag{72}$$

Repeating the previous calculation with the RMSProp update formula that includes the symmetric preconditioner P_t results in the following stability criterion for a single step:

$$\frac{\nabla L_{\mathcal{B}_t}(\theta_t)^\top P_t^{-1} H_{\mathcal{B}_t}(\theta_t) P_t^{-1}\nabla L_{\mathcal{B}_t}(\theta_t)}{\nabla L_{\mathcal{B}_t}(\theta_t)^\top P_t^{-1}\nabla L_{\mathcal{B}_t}(\theta_t)} \leq \frac{2}{\eta} \tag{73}$$

Again, assuming $P_{t-1} \approx P_t$ yields the following definition of effective batch sharpness for RMSProp:

$$\mathrm{BS}_{\mathrm{RMSP}}(\theta_t, P_{t-1}) = \mathbb{E}_{\mathcal{B}_t \sim \mathcal{P}} \left[\frac{\nabla L_{\mathcal{B}_t}(\theta_t)^\top P_{t-1}^{-1} H_{\mathcal{B}_t}(\theta_t) P_{t-1}^{-1} \nabla L_{\mathcal{B}_t}(\theta_t)}{\nabla L_{\mathcal{B}_t}(\theta_t)^\top P_{t-1}^{-1} \nabla L_{\mathcal{B}_t}(\theta_t)} \;\middle|\; \mathcal{F}_t \right] \tag{74}$$

Numerical experiments confirm that both $\mathrm{BS}_{\mathrm{scRMSP}}$ and $\mathrm{BS}_{\mathrm{RMSP}}$ commonly stabilize at $2/\eta$ for neural network training with mini-batch scalar RMSProp and mini-batch RMSProp, respectively. See Section 15 for details.

14.3 Batch Sharpness for Adam

Remember the Adam optimizer uses the following update formula:

$$\theta_{t+1} = \theta_t - \eta P_t^{-1} m_{t+1} \tag{75}$$

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1) \nabla L_{\mathcal{B}_t}(\theta_t) \tag{76}$$

Substituting the Adam step $\theta_{t+1} - \theta_t = -\eta P_t^{-1} m_{t+1}$ into the Taylor expansion of the mini-batch loss (66) yields:

$$\begin{aligned} L_{\mathcal{B}_t}(\theta_{t+1}) \approx & L_{\mathcal{B}_t}(\theta_t) + \nabla L_{\mathcal{B}_t}(\theta_t)^\top \left(-\eta P_t^{-1} m_{t+1} \right) \\ & + \frac{1}{2} \left(-\eta P_t^{-1} m_{t+1} \right)^\top H_{\mathcal{B}_t}(\theta_t) \left(-\eta P_t^{-1} m_{t+1} \right) \end{aligned} \tag{77}$$

Because P_t^{-1} is symmetric, after rearranging we get:

$$L_{\mathcal{B}_t}(\theta_{t+1}) - L_{\mathcal{B}_t}(\theta_t) \approx -\eta \nabla L_{\mathcal{B}_t}(\theta_t)^\top P_t^{-1} m_{t+1} + \frac{\eta^2}{2} m_{t+1}^\top P_t^{-1} H_{\mathcal{B}_t}(\theta_t) P_t^{-1} m_{t+1} \tag{78}$$

Forcing the loss to decrease results in the following stability criterion for a single step:

$$\frac{m_{t+1}^\top P_t^{-1} H_{\mathcal{B}_t}(\theta_t) P_t^{-1} m_{t+1}}{\nabla L_{\mathcal{B}_t}(\theta_t)^\top P_t^{-1} m_{t+1}} \leq \frac{2}{\eta}$$

But this criterion is fundamentally flawed. Because momentum-based optimizers are not memoryless, they can and often do temporarily violate the monotonic descent condition without the overall trajectory diverging. Furthermore, to be in line with the other measures of directed curvature like the IAS and batch sharpness, we want this measure to be a Rayleigh quotient.

We can derive a stability criterion that is a Rayleigh quotient by mapping the system into the preconditioned coordinate space $\tilde{\theta} = P_t^{1/2} \theta$. As demonstrated in Appendix A.2 of [5], running a preconditioned momentum-based optimizer like Adam is mathematically isomorphic to running standard Heavy Ball momentum on this reparameterized loss landscape. The update rule

for Adam becomes

$$\tilde{\theta}_{t+1} = \tilde{\theta}_t - \eta \tilde{m}_{t+1} \tag{79}$$

The direction of the Adam step is now the preconditioned momentum $\tilde{m}_{t+1} = P_t^{-1/2} m_{t+1}$, and the preconditioned Hessian is $\tilde{H}_{\mathcal{B}_t} = P_t^{-1/2} H_{\mathcal{B}_t}(\theta_t) P_t^{-1/2}$. Evaluating the Rayleigh quotient along the direction of the step the optimizer takes gives us

$$\text{Step Sharpness}_{\text{Adam}} = \frac{\tilde{m}_{t+1}^\top \tilde{H}_{\mathcal{B}_t} \tilde{m}_{t+1}}{\tilde{m}_{t+1}^\top \tilde{m}_{t+1}} = \frac{m_{t+1}^\top P_t^{-1} H_{\mathcal{B}_t}(\theta_t) P_t^{-1} m_{t+1}}{m_{t+1}^\top P_t^{-1} m_{t+1}}$$

To get an expectation, we again assume $P_t \approx P_{t-1}$. Because the β_1 used for momentum is typically 0.9, we cannot make this same assumption for m_{t+1}. Instead, we calculate it using the randomly sampled batch and the old momentum with the formula $m_{t+1} = \beta_1 m_t + (1 - \beta_1)\nabla L_{\mathcal{B}_t}(\theta_t)$. Let $\mathcal{F}_t$ again be the σ-algebra of previous steps, this time also including the initial EMA ν_0 and momentum m_0.

The effective batch sharpness for Adam is therefore:

$$\text{BS}_{\text{Adam}}(\theta_t, m_t, P_{t-1}) = \mathbb{E}_{\mathcal{B}_t \sim \mathcal{P}} \left[\frac{m_{t+1}^\top P_{t-1}^{-1} H_{\mathcal{B}_t}(\theta_t) P_{t-1}^{-1} m_{t+1}}{m_{t+1}^\top P_{t-1}^{-1} m_{t+1}} \;\middle|\; \mathcal{F}_t \right] \tag{80}$$

Evaluating the expectation of this entire fraction leads to problems in Proof 14.4. Andreyev and Beneventano [2] introduced the "modified batch sharpness", which is the batch sharpness but with separate expectations for both parts of the fraction:

$$\tilde{\text{BS}}_{\text{SGD}}(\theta_t) = \frac{\mathbb{E}_{\mathcal{B}_t \sim \mathcal{P}} \left[\nabla L_{\mathcal{B}_t}(\theta_t)^\top H_{\mathcal{B}_t}(\theta_t) \nabla L_{\mathcal{B}_t}(\theta_t) \right]}{\mathbb{E}_{\mathcal{B}_t \sim \mathcal{P}} \left[\|\nabla L_{\mathcal{B}_t}(\theta_t)\|^2 \right]} \tag{81}$$

They proved that $\tilde{\text{BS}}_{\text{SGD}}$ is a valid instability criterion and stabilizes at a value higher than $2/\eta$ in practice. Analogous to this formulation, we define the modified batch sharpness for Adam:

$$\tilde{\text{BS}}_{\text{Adam}}(\theta_t, m_t, P_{t-1}) = \frac{\mathbb{E}_{\mathcal{B}_t \sim \mathcal{P}} \left[m_{t+1}^\top P_{t-1}^{-1} H_{\mathcal{B}_t}(\theta_t) P_{t-1}^{-1} m_{t+1} \mid \mathcal{F}_t \right]}{\mathbb{E}_{\mathcal{B}_t \sim \mathcal{P}} \left[\|m_{t+1}\|_{P_{t-1}^{-1}}^2 \mid \mathcal{F}_t \right]} \tag{82}$$

Since the sharpness for full-batch Adam must be bounded by $\text{MSS}_{\text{HB}^{\text{EMA}}}$, a logical choice is to adopt the same threshold for Adam batch sharpness. The resulting condition for stability would be:

$$\text{BS}_{\text{Adam}}(\theta_t, m_t, P_{t-1}) \leq \frac{2 + 2\beta_1}{\eta(1 - \beta_1)} \quad \text{or} \quad \tilde{\text{BS}}_{\text{Adam}}(\theta_t, m_t, P_{t-1}) \leq \frac{2 + 2\beta_1}{\eta(1 - \beta_1)} \tag{83}$$

Unfortunately, in numerical experiments with mini-batch Adam, BS_{Adam} normally does not reach $\text{MSS}_{\text{HB}^{\text{EMA}}}$. See Section 15 for details. However, under some assumptions, we can prove that if

$\widetilde{\text{BS}}_{\text{Adam}}$ were consistently above $\text{MSS}_{\text{HB}^{\text{EMA}}}$, the expected root-mean-square of the momentum would grow exponentially, as we will see in the next section.

14.4 Proof of Divergence of Adam if Batch Sharpness Exceeds the MSS

In this section, we derive a proof for Adam becoming unstable when the modified effective batch sharpness is above $\text{MSS}_{\text{HB}^{\text{EMA}}}$. The proof is similar to the proof of Proposition 1, where we showed SGD diverges if the batch sharpness exceeds $(2+\epsilon)/\eta$. There, we analyzed the fraction $\frac{\|\nabla L_{\mathcal{B}_t}(\theta_{t+1})\|^2}{\|\nabla L_{\mathcal{B}_t}(\theta_t)\|^2}$ corresponding to investigating how the magnitude of the gradients changes after the step from θ_t to θ_{t+1} if evaluated on the same batch $\mathcal{B}_t$. For Adam, we want to study the behavior of the squared preconditioned momentum norm $\|m_t\|^2_{P^{-1}}$ instead of the squared gradient norm $\|\nabla L_{\mathcal{B}_t}(\theta_{t+1})\|^2$. Analogous to the proof of Proposition 1, instead of tracking the true momentum m_{t+2}, we define the virtual on-batch momentum $\check{m}_{t+2}$. It represents what the momentum would be if evaluated on the same batch $\mathcal{B}_t$ at the new position θ_{t+1}:

$$\check{m}_{t+2} = \beta_1 m_{t+1} + (1-\beta_1)\nabla L_{\mathcal{B}_t}(\theta_{t+1}) \tag{84}$$

For the proof of local instability of Adam, the following assumptions are needed

1. **Locally Quadratic Landscape:** As in the proof for batch sharpness for SGD, we assume that for any batch the loss function is locally quadratic with a constant Hessian $H_{\mathcal{B}_t}$.
2. **Slowly Changing Preconditioner:** We assume that once $\widetilde{\text{BS}}_{\text{Adam}}(\theta_t, m_t, P_{t-1})$ crosses the stability threshold $\text{MSS}_{\text{HB}^{\text{EMA}}}$, the preconditioner P_t remains quasi-static for K steps. Because β_2 is typically 0.999, we can justify $P_{t+k} \approx P_t := P$ for $k \in [0, K]$.
3. **Virtual Momentum Equivalence:** We assume that for any step $k \in [0, K-2]$ within the quasi-static window, the conditionally expected squared norm of the virtual momentum is the same as the conditionally expected squared norm of the true momentum:

$$\mathbb{E}[\|\check{m}_{t+k+2}\|^2_{P^{-1}} \mid \mathcal{F}_t] = \mathbb{E}[\|m_{t+k+2}\|^2_{P^{-1}} \mid \mathcal{F}_t] \tag{85}$$

   where the conditional expectation $\mathbb{E}[\cdot \mid \mathcal{F}_t]$ is taken over the future random mini-batch draws given $\mathcal{F}_t$. This assumption means that on average across all possible future trajectories projected from our current state, the squared norm of the momentum vector is the same whether we re-used the old batch in the most recent iteration or drew a new one.

Theorem (Local Instability of Adam at the Edge of Stability)
Consider a loss function $L(\theta)$ that is optimized by Adam with learning rate $\eta > 0$, momentum parameter $\beta_1 \in (0,1)$, and a positive-definite preconditioner P. Let $\mathcal{F}_t$ be the filtration with the optimization history up to step t, and let $N_{t+k} = \sqrt{\mathbb{E}[\|m_{t+k}\|^2_{P^{-1}} \mid \mathcal{F}_t]}$ be the conditionally expected root-mean-square norm of the momentum at step $t+k$.

Suppose Assumptions 1-3 hold for a quasi-static window of K steps starting at step t. If the initial momentum norm does not shrink by more than Adam's momentum decay parameter

($N_{t+1} \geq \beta_1 N_t$), and there exists a constant $\epsilon > 0$ such that for all steps from t to $t+K$ the modified Adam batch sharpness exceeds $\text{MSS}_{\text{HB}^{\text{EMA}}}$:

$$\tilde{\text{BS}}_{\text{Adam}}(\theta_{t+k}, m_{t+k}, P) \geq \frac{2+2\beta_1}{\eta(1-\beta_1)} + \epsilon$$

then the expected momentum norm N_{t+k} grows exponentially for the duration of the quasi-static window.

Remark: $N_{t+1} \geq \beta_1 N_t$ excludes the rare scenario where an incoming mini-batch gradient strongly opposes the historical momentum. Because stochastic gradient noise ensures that such strong opposition cannot be sustained indefinitely, we can set the start of the quasi-static window at a step where this condition is satisfied.

Proof. Let the quasi-static window begin at step t, where we assume $\tilde{\text{BS}}_{\text{Adam}}$ crosses $\text{MSS}_{\text{HB}^{\text{EMA}}}$. Without loss of generality, we shift the time coordinate to set $t = 0$. We analyze the local stability in the window $[0, K]$.

By Assumption 1 (Locally Quadratic Landscape), substituting the Adam step into the gradient yields:

$$\nabla L_{\mathcal{B}_k}(\theta_{k+1}) = \nabla L_{\mathcal{B}_k}(\theta_k) - \eta H_{\mathcal{B}_k} P^{-1} m_{k+1}$$

We plug this into our virtual momentum equation $\check{m}_{k+2} = \beta_1 m_{k+1} + (1-\beta_1)\nabla L_{\mathcal{B}_k}(\theta_{k+1})$ and rearrange using $(1-\beta_1)\nabla L_{\mathcal{B}_k}(\theta_k) = m_{k+1} - \beta_1 m_k$:

$$\check{m}_{k+2} + \beta_1 m_k = \Big[(1+\beta_1)I - \eta(1-\beta_1)H_{\mathcal{B}_k}P^{-1}\Big] m_{k+1} \tag{86}$$

Now we define $x := P^{-1/2} m_{k+1}$ (P is a positive-definite diagonal matrix, so $P^{-1/2}$ is well-defined) and the symmetric matrix $A := (1+\beta_1)I - \eta(1-\beta_1)P^{-1/2}H_{\mathcal{B}_k}P^{-1/2}$. Equation (86) can be rewritten as $P^{-1/2}(\check{m}_{k+2} + \beta_1 m_k) = Ax$. Squaring both sides yields:

$$\|\check{m}_{k+2} + \beta_1 m_k\|_{P^{-1}}^2 = \|Ax\|^2$$

Now we want to apply the conditional expectation $\mathbb{E}[\cdot \mid \mathcal{F}_0]$, where $\mathcal{F}_0$ captures all the previous optimization history up to time $t = 0$. First, we use the Cauchy-Schwarz inequality for random variables ($\mathbb{E}[UV]^2 \leq \mathbb{E}[U^2]\mathbb{E}[V^2]$) with $U = \|x\|$ and $V = x^\top Ax/\|x\|$, and we use the standard vector inequality $(x^\top Ax)^2 \leq \|x\|^2\|Ax\|^2$ to get the following result:

$$\mathbb{E}[x^\top Ax \mid \mathcal{F}_0]^2 \leq \mathbb{E}[\|x\|^2 \mid \mathcal{F}_0]\mathbb{E}[\|Ax\|^2 \mid \mathcal{F}_0]$$

Rearranging this gives us a lower bound for the conditional expectation of $\|Ax\|^2$:

$$\mathbb{E}[\|Ax\|^2 \mid \mathcal{F}_0] \geq \mathbb{E}[\|x\|^2 \mid \mathcal{F}_0] \left(\frac{\mathbb{E}[x^\top Ax \mid \mathcal{F}_0]}{\mathbb{E}[\|x\|^2 \mid \mathcal{F}_0]}\right)^2 \tag{87}$$

We plug in the following definitions:

$$\|x\|^2 = \|m_{k+1}\|_{P^{-1}}^2$$
$$x^\top Ax = (1+\beta_1)\|m_{k+1}\|_{P^{-1}}^2 - \eta(1-\beta_1)m_{k+1}^\top P^{-1}H_{\mathcal{B}_k}P^{-1}m_{k+1}$$

We can see the ratio inside the square contains the modified Adam batch sharpness evaluated at step k, but conditioned on filtration $\mathcal{F}_0$ instead of $\mathcal{F}_k$:

$$\tilde{\text{BS}}^{(0)}_{\text{Adam}}(\theta_k, m_k, P) = \frac{\mathbb{E}\left[m_{k+1}^\top P^{-1}H_{\mathcal{B}_k}P^{-1}m_{k+1} \mid \mathcal{F}_0\right]}{\mathbb{E}\left[\|m_{k+1}\|_{P^{-1}}^2 \mid \mathcal{F}_0\right]} \tag{88}$$

Substituting x, A, and the modified Adam batch sharpness into (87) yields:

$$\mathbb{E}\left[\|\check{m}_{k+2} + \beta_1 m_k\|_{P^{-1}}^2 \mid \mathcal{F}_0\right] \geq \mathbb{E}\left[\|m_{k+1}\|_{P^{-1}}^2 \mid \mathcal{F}_0\right]\left((1+\beta_1) - \eta(1-\beta_1)\tilde{\text{BS}}^{(0)}_{\text{Adam}}\right)^2 \tag{89}$$

By the Tower Property, $\mathbb{E}[\mathbb{E}[X \mid \mathcal{F}_k] \mid \mathcal{F}_0] = \mathbb{E}[X \mid \mathcal{F}_0]$. Since the local stability bound holds almost surely and the expectation of the constant $\text{MSS}_{\text{HB}^{\text{EMA}}}$ is simply itself, $\tilde{\text{BS}}^{(0)}_{\text{Adam}}$ inherits the threshold:

$$\tilde{\text{BS}}^{(0)}_{\text{Adam}}(\theta_k, m_k, P) \geq \frac{2+2\beta_1}{\eta(1-\beta_1)} + \epsilon$$

We substitute this into (89):

$$(1+\beta_1) - \eta(1-\beta_1)\tilde{\text{BS}}^{(0)}_{\text{Adam}} \leq -(1+\beta_1) - \eta(1-\beta_1)\epsilon$$

Let $\tilde{\epsilon} = \eta(1-\beta_1)\epsilon$. Because both sides are negative, squaring reverses the inequality:

$$\left((1+\beta_1) - \eta(1-\beta_1)\tilde{\text{BS}}^{(0)}_{\text{Adam}}\right)^2 \geq (1+\beta_1+\tilde{\epsilon})^2$$

Plugging this into equation (89) yields:

$$\mathbb{E}\left[\|\check{m}_{k+2} + \beta_1 m_k\|_{P^{-1}}^2 \mid \mathcal{F}_0\right] \geq \mathbb{E}\left[\|m_{k+1}\|_{P^{-1}}^2 \mid \mathcal{F}_0\right](1+\beta_1+\tilde{\epsilon})^2$$

Taking the square root of both sides gives:

$$\sqrt{\mathbb{E}\left[\|\check{m}_{k+2} + \beta_1 m_k\|_{P^{-1}}^2 \mid \mathcal{F}_0\right]} \geq (1+\beta_1+\tilde{\epsilon})\sqrt{\mathbb{E}\left[\|m_{k+1}\|_{P^{-1}}^2 \mid \mathcal{F}_0\right]} \tag{90}$$

Now we use Minkowski's Inequality, which is the triangle inequality for L^p spaces:

$$\|U+V\|_{L^p} \leq \|U\|_{L^p} + \|V\|_{L^p}$$

Since for any random variable X, $\|X\|_{L^2} = (\mathbb{E}[|X|^2])^{1/2}$, and the Tower Property ensures this inequality holds for conditional expectations, we can separate the conditional expectation of the sum on the left side:

$$\sqrt{\mathbb{E}\left[\|\check{m}_{k+2}\|_{P^{-1}}^2 \mid \mathcal{F}_0\right]} + \beta_1 \sqrt{\mathbb{E}\left[\|m_k\|_{P^{-1}}^2 \mid \mathcal{F}_0\right]} \geq \sqrt{\mathbb{E}\left[\|\check{m}_{k+2} + \beta_1 m_k\|_{P^{-1}}^2 \mid \mathcal{F}_0\right]}$$

Recall our definition of N_k, which again conditions only on $\mathcal{F}_0$ for any k:

$$N_k = \sqrt{\mathbb{E}\left[\|m_k\|_{P^{-1}}^2 \mid \mathcal{F}_0\right]}$$

By Assumption 3 (Virtual Momentum Equivalence), we get $\mathbb{E}[\|\check{m}_{k+2}\|_{P^{-1}}^2 \mid \mathcal{F}_0] = \mathbb{E}[\|m_{k+2}\|_{P^{-1}}^2 \mid \mathcal{F}_0]$, meaning the first term evaluates to N_{k+2}.

We plug the definition into (90) and apply Minkowski's inequality to get a clean formula for N_{k+2}:

$$N_{k+2} + \beta_1 N_k \geq (1 + \beta_1 + \tilde{\epsilon}) N_{k+1}$$

Rearranging the terms, we obtain a second-order linear recurrence inequality, which governs the local momentum dynamics for any step $k \in [0, K]$:

$$N_{k+2} - (1 + \beta_1 + \tilde{\epsilon}) N_{k+1} + \beta_1 N_k \geq 0 \tag{91}$$

To solve linear recurrence inequalities of this form, we analyze the corresponding equality, whose fundamental solutions are λ_+^k and λ_-^k. λ_+ and λ_- are the roots of the characteristic polynomial $f(\lambda) = \lambda^2 - (1 + \beta_1 + \tilde{\epsilon})\lambda + \beta_1$. By Vieta's formulas, the sum and product of the roots correspond to the coefficients of the linear recurrence inequality:

$$\begin{aligned} \lambda_+ + \lambda_- &= 1 + \beta_1 + \tilde{\epsilon} \\ \lambda_+ \lambda_- &= \beta_1 \end{aligned}$$

Evaluating the polynomial at 1 yields $f(1) = -\tilde{\epsilon} < 0$. Since the parabola opens upwards, this guarantees that one root is strictly greater than 1 ($\lambda_+ > 1$) while the other satisfies $0 < \lambda_- < \beta_1$.

Using these roots, we can cleanly factor and rewrite the recurrence inequality (91) to decouple the growing and decaying components:

$$\begin{aligned} N_{k+2} - (\lambda_+ + \lambda_-) N_{k+1} + (\lambda_+ \lambda_-) N_k &\geq 0 \\ N_{k+2} - \lambda_- N_{k+1} &\geq \lambda_+ (N_{k+1} - \lambda_- N_k) \end{aligned}$$

Let $Z_k := N_{k+1} - \lambda_- N_k$, which simplifies the inequality to:

$$Z_{k+1} \geq \lambda_+ Z_k$$

By the hypothesis of the theorem, $N_1 \geq \beta_1 N_0$. Because $\lambda_- < \beta_1$, this guarantees $N_1 > \lambda_- N_0$ and therefore $Z_0 > 0$. Given the positive initial condition, this inequality dictates exponential growth:

$$Z_k \geq \lambda_+^k Z_0$$

Since $N_{k+1} = Z_k + \lambda_- N_k$, and N_k is strictly non-negative by the definition of the RMS norm, it immediately follows that:

$$N_{k+1} \geq \lambda_+^k Z_0 + \lambda_- N_k \geq \lambda_+^k Z_0$$

Because $\lambda_+ > 1$ and $Z_0 > 0$, this confirms that N_k experiences exponential growth for the entirety of the quasi-static window, verifying the local instability of Adam. □

The proof for instability with SGD included a section where it was proven that batch sharpness stays above the stability threshold for a period of time after the edge of stability is originally reached. Finding an analogous proof for modified Adam batch sharpness is challenging because, even if we assume a constant preconditioner P, the modified Adam batch sharpness depends on a joint state space: the parameters θ and the momentum vector m_t, both of which must be considered. Assuming local Lipschitz continuity with respect to both variables yields:

$$|\widetilde{\mathrm{BS}}_{\mathrm{Adam}}(\theta_{t+1}, m_{t+1}, P) - \widetilde{\mathrm{BS}}_{\mathrm{Adam}}(\theta_t, m_t, P)| \leq L_\theta \|\theta_{t+1} - \theta_t\| + L_m \|m_{t+1} - m_t\|$$

The drift of θ is throttled by the strictly small learning rate η, but the drift of the momentum is only throttled by the factor $(1 - \beta_1)$. For a standard value of $\beta_1 = 0.9$, this factor is 0.1, which is not sufficiently small to guarantee a slow transition. It is possible that the time window where $\widetilde{\mathrm{BS}}_{\mathrm{Adam}}$ stays above $\mathrm{MSS}_{\mathrm{HB}^{\mathrm{EMA}}}$ is shorter than a single iteration. As we will see in the next section, when training mini-batch Adam in a realistic setting, $\widetilde{\mathrm{BS}}_{\mathrm{Adam}}$ normally does not even reach $\mathrm{MSS}_{\mathrm{HB}^{\mathrm{EMA}}}$.

15 Numerical Experiments

The following section outlines the numerical experiments conducted in this thesis, mainly focusing on the evaluation of the behavior and stability of adaptive optimizers in the mini-batch setting. The experimental framework was adapted from the existing codebase of Andreyev and Beneventano [2], but heavily modified to add support for adaptive optimizers and to accommodate the strict VRAM constraints of the hardware used. All of the corresponding code can be found in the following GitHub repository: `https://github.com/JakobFabianRohner/eos-stochastic-adaptive-optimizers`. All experiments were executed on a workstation equipped with an NVIDIA RTX 3070 Ti GPU and an Intel Core i5-12600K CPU. All experiments in this section were conducted using a multilayer perceptron (MLP) trained on a four-class subset of the CIFAR-10 dataset, restricted to 8,192 data points.

- **Sharpness** (13): Computed iteratively using SciPy's Lanczos algorithm (`eigsh`) by calculating Hessian-vector products via the Pearlmutter trick [25] incrementally over memory-safe physical batches.
- **Step sharpness** (51): The step sharpness of a single mini-batch, calculated as the Rayleigh quotient $\frac{g^\top Hg}{g^\top g}$. The matrix-vector product Hg is again computed via the Pearlmutter trick.
- **Batch sharpness** (52): Approximated numerically via Monte Carlo sampling by calculating the average step sharpness for 10 to 50 randomly sampled mini-batches, terminating early if the relative standard error of the mean is low.
- **Effective sharpness** (41, 42): For scalar RMSProp, this is implemented as a direct scalar scaling: $S(\theta_t)/(\sqrt{\nu_{t-1}}+\epsilon)$. For standard RMSProp and Adam, we calculate the maximum eigenvalue of the preconditioned matrix $P_t^{-1/2} H P_t^{-1/2}$ using the Lanczos method.
- **Effective step sharpness and effective batch sharpness** (74): For scalar RMSProp, we again simply scale the standard step sharpness or batch sharpness accordingly. For RMSProp and Adam, we compute $\frac{p^\top Hp}{g^\top p}$ where $p = P_t^{-1} g$ for effective step sharpness. Effective batch sharpness is again calculated using Monte Carlo sampling.
- **Effective batch sharpness momentum** (80): Specific to Adam, we replace the gradient g with the bias-corrected momentum vector m_{t+1}. The momentum step sharpness is computed as $\frac{p^\top Hp}{m_{t+1}^\top p}$ where $p = P_t^{-1} m_{t+1}$, and the corresponding expected batch sharpness is again approximated via Monte Carlo sampling.

15.1 Scalar RMSProp

We empirically verify that effective batch sharpness serves as the correct instability criterion for scalar RMSProp. Training was performed with a batch size of 16, a learning rate of $\eta = 0.01$, and a numerical stability term of $\epsilon = 10^{-7}$. In Figure 17, we see that for this configuration, the effective batch sharpness is indeed the governing threshold. In Figure 18, we investigate further details regarding the behavior of the raw (unscaled) sharpness metrics as well as the preconditioner scalar ν_t.

15.2 RMSProp

In the RMSProp update rule (6), the effective step size for parameter i is scaled by the factor $\frac{\eta}{\sqrt{\nu_{t,i}}+\epsilon}$. When the optimizer enters a flat valley, the gradient moving average decays such that $\nu_{t,i}$ is close to 0, causing the effective learning rate to approach its upper bound of $\frac{\eta}{\epsilon}$. For a small ϵ like $\epsilon = 10^{-7}$, this causes the effective step size to be very large, which destabilizes the effective sharpness. We can prevent this by choosing a larger ϵ such as $\epsilon = 10^{-4}$, as shown in Figure 19. In the mini-batch setting, we are interested in the effective batch sharpness (74), which can be seen in Figure 20.

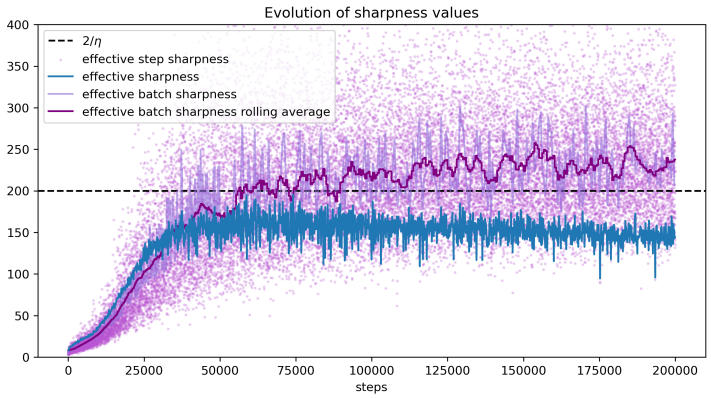


Figure 17: Scalar RMSProp: The effective step sharpness (red dots) exhibits high variance. The effective batch sharpness (purple) hovers slightly above the $2/\eta$ boundary, while the effective sharpness (blue) stays below the MSS.

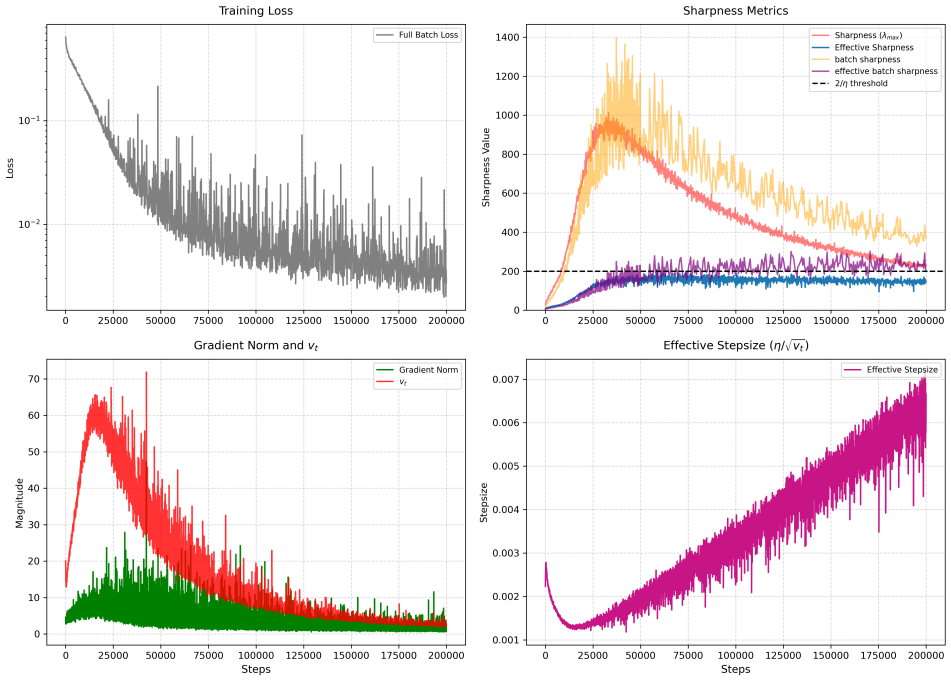
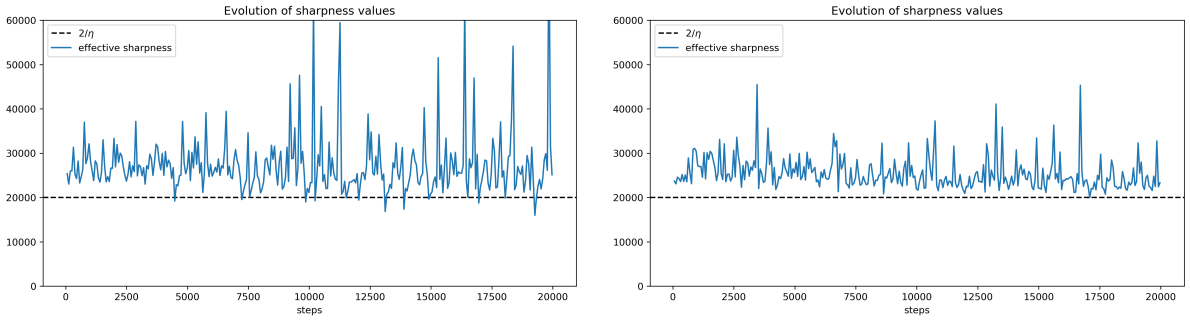


Figure 18: Training dynamics of scalar RMSProp. **Top right:** The raw sharpness metrics (both full and batch) significantly overshoot the $2/\eta$ boundary before the optimizer finds flatter regions. **Bottom left:** The moving average ν_t increases during the phase where the raw sharpness metrics increase, while both of the effective sharpness metrics increase at a much slower pace. When the raw sharpness metrics start to decrease again, ν_t decreases with them, allowing the effective batch sharpness to stabilize at the MSS and the effective sharpness to remain slightly below it. **Bottom right:** After an initial drop in the first 20,000 iterations, the effective step size $(\eta/\sqrt{\nu_t})$ steadily increases throughout the remainder of training.



(a) $\epsilon = 10^{-7}$ **(b)** $\epsilon = 10^{-4}$

Figure 19: Comparison of full-batch RMSProp stability with $\eta = 0.0001$. **Left**: For a small $\epsilon = 10^{-7}$, the effective sharpness is very unstable and experiences many catapults. **Right**: For the larger $\epsilon = 10^{-4}$, the effective sharpness is much more stable.

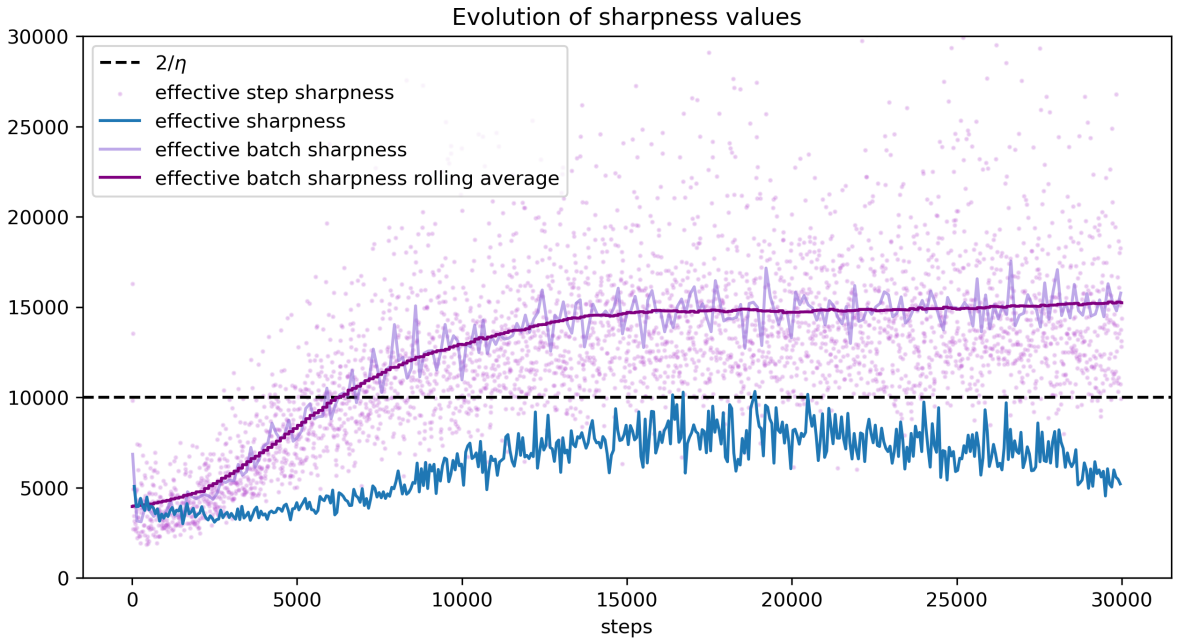


Figure 20: Effective sharpness dynamics for RMSProp trained with batch size 16, learning rate $\eta = 0.0002$ and $\epsilon = 10^{-4}$. The effective batch sharpness (purple) stabilizes slightly above the MSS of $2/\eta$, while the full-batch effective sharpness (blue) stays at a lower value.

15.3 Adam

In our experiments with the Adam optimizer, we utilized the standard hyperparameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Under these settings, the theoretical maximum stable sharpness is $\text{MSS}_{\text{HB}^{\text{EMA}}} = 38/\eta$. We analyzed both the effective batch sharpness (74) with the formula used for RMSProp as well as the momentum effective batch sharpness (80). For the experiments conducted, both of these values consistently stayed below the standard effective sharpness (42) and did not rise to $\text{MSS}_{\text{HB}^{\text{EMA}}}$. Furthermore, we observed a consistent batch-size dependence: training with smaller batch sizes resulted in lower sharpness values across all three metrics compared to training with larger batch sizes.

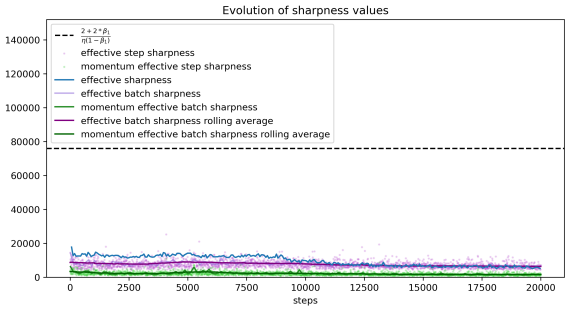
Why Adam Stays Below the Theoretical MSS

Adam's effective sharpness generally stays much lower than the theoretical edge of stability. This can likely be attributed to the momentum and the preconditioner handling oscillating gradients differently.

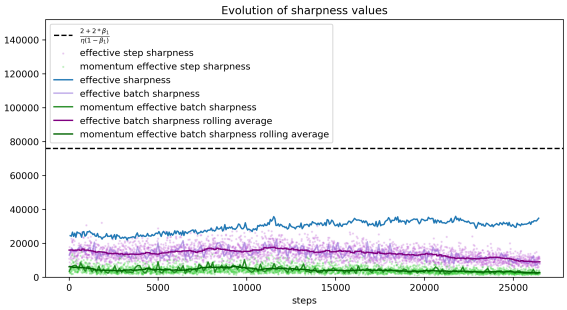
- **Momentum cancellation:** The situation is really similar to that described in Subsection 11.3. When the optimizer enters a sharp valley, the gradients along the highest-curvature eigenvectors begin to oscillate rapidly, flipping signs at each step. Because Adam's momentum vector m_t relies on a moving average of these gradients, the alternating signs effectively cancel out. This results in an exponentially decaying momentum vector in the sharpest directions.
- **Preconditioner dampening:** Contrary to the momentum, the EMA ν_t uses squared gradients. The sign-flipping does not cancel out in the update formula of ν_t, causing it to become very large. This means the values of the inverse preconditioner P_t^{-1} corresponding to the effective step size in these high-curvature directions become very small, which heavily dampens the effective step size in these directions.

Combined, these two effects heavily suppress the optimizer's movement along the sharpest eigenvectors. While the surrounding loss landscape still contains highly sharp directions, the actual trajectory of the optimizer is steered to flatter subspaces. This keeps the momentum batch sharpness below the $\text{MSS}_{\text{HB}^{\text{EMA}}}$ threshold.

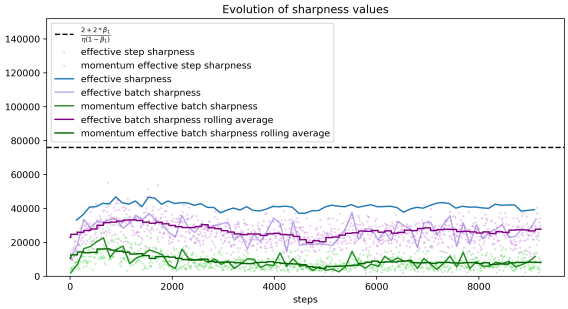
A possible explanation for why the gradient-based effective batch sharpness plateaus below the theoretical limit is the effect of stochastic noise. The high variance of mini-batch gradients inflates ν_t, which continuously scales down the preconditioned curvature. This prevents the preconditioned curvature from ever reaching $\text{MSS}_{\text{HB}^{\text{EMA}}}$.



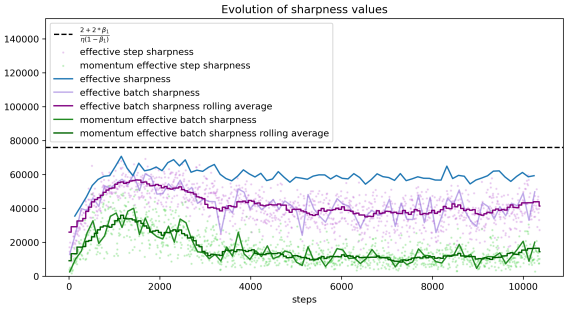
(a) Batch size 64



(b) Batch size 128



(c) Batch size 256



(d) Batch size 512

Figure 21: Various sharpness metrics for Adam trained with varying batch sizes and $\eta = 0.0005$. Across all batch sizes, none of the sharpness metrics reach the theoretical MSS of $38/\eta$. The full-batch effective sharpness (blue) maintains the highest value throughout training. The pure-gradient effective batch sharpness (purple) stabilizes at a lower value, while the momentum effective batch sharpness (green) consistently remains the lowest of the three. For smaller batch sizes, all sharpness metrics are lower compared to larger batch sizes.

16 Conclusion

This thesis provided a comprehensive overview of progressive sharpening and Edge of Stability behavior in different neural networks. Through the theoretical lenses of self-stabilization and the central flow framework, we provided an heuristic explanation for these behaviors and examined how optimizers actively seek flatter regions of the loss landscape, also including adaptive gradient methods. Via the Interaction-Aware sharpness, we bridged the gap to the more complex, stochastic regime of mini-batch optimization, where batch sharpness is the primary instability criterion for standard SGD. We were able to improve a Lipschitz constant and correct an error in the derivation of the minimum unstable window in the corresponding paper [2].

The primary novel contribution of this thesis was the extension of the Edge of Stochastic Stability framework to mini-batch adaptive optimizers. We heuristically derived new batch sharpness criteria for scalar RMSProp, standard RMSProp, and Adam. Using these newly established criteria, we provided a rigorous theoretical proof demonstrating that if Adam's modified batch sharpness exceeds the theoretical maximum stable sharpness (MSS) threshold of $\frac{2+2\beta_1}{\eta(1-\beta_1)}$, the expected root-mean-square norm of the momentum vector grows exponentially. This triggers a catapult effect, leading to the local quadratic approximation becoming unstable.

Our numerical experiments revealed a stark contrast between these theoretical instability bounds and practical optimizer dynamics. The newly derived sharpness metrics for both scalar RMSProp and standard RMSProp consistently are bounded by their respective MSS boundaries. But Adam behaves fundamentally differently. Across all tested batch sizes, Adam's effective batch sharpness and momentum batch sharpness consistently stayed well below the theoretical instability threshold, never even triggering the exponential growth outlined in our proof. Adam's internal mechanics seem to suppress its movement along the sharpest eigenvectors. Potential reasons for this are explained in the previous section.

Bibliography

[1] Kwangjun Ahn, Sébastien Bubeck, Sinho Chewi, Yin Tat Lee, Felipe Suarez, and Yi Zhang. "Learning threshold neurons via edge of stability". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 19540–19569.

[2] Arseniy Andreyev and Pierfrancesco Beneventano. "Edge of Stochastic Stability: Revisiting the Edge of Stability for SGD". In: *arXiv preprint arXiv:2412.20553* (2024).

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[4] Jeremy M Cohen, Alex Damian, Ameet Talwalkar, Zico Kolter, and Jason D Lee. "Understanding Optimization in Deep Learning with Central Flows". In: *arXiv preprint arXiv:2410.24206* (2024).

[5] Jeremy M Cohen, Behrooz Ghorbani, Shankar Krishnan, Naman Agarwal, Sourabh Medapati, Michal Badura, Daniel Suo, David Cardoze, Zachary Nado, George E Dahl, et al. "Adaptive gradient methods at the edge of stability". In: *arXiv preprint arXiv:2207.14484* (2022).

[6] Jeremy M Cohen, Simran Kaur, Yuanzhi Li, J Zico Kolter, and Ameet Talwalkar. "Gradient descent on neural networks typically occurs at the edge of stability". In: *arXiv preprint arXiv:2103.00065* (2021).

[7] Alex Damian, Eshaan Nichani, and Jason D Lee. "Self-stabilization: The implicit bias of gradient descent at the edge of stability". In: *arXiv preprint arXiv:2209.15594* (2022).

[8] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. "Sharp minima can generalize for deep nets". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1019–1028.

[9] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.7 (2011), pp. 2121–2159.

[10] Jeffrey L. Elman. "Finding structure in time". In: *Cognitive Science* 14 (1990), pp. 179–211.

[11] Sepp Hochreiter and Jürgen Schmidhuber. "Flat Minima". In: *Neural Computation* 9.1 (Jan. 1997), pp. 1–42. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.1.1. eprint: https://direct.mit.edu/neco/article-pdf/9/1/1/813385/neco.1997.9.1.1.pdf. URL: https://doi.org/10.1162/neco.1997.9.1.1.

[12] Stanislaw Jastrzebski, Maciej Szymczak, Stanislav Fort, Devansh Arpit, Jacek Tabor, Kyunghyun Cho, and Krzysztof Geras. "The break-even point on optimization trajectories of deep neural networks". In: *arXiv preprint arXiv:2002.09572* (2020).

[13] Stanislaw Jastrzebski, Maciej Szymczak, Stanislav Fort, Devansh Arpit, Jacek Tabor, Kyunghyun Cho, and Krzysztof Geras. "The break-even point on optimization trajectories of deep neural networks". In: *arXiv preprint arXiv:2002.09572* (2020).

[14] Stanisław Jastrzębski, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. “Three factors influencing minima in sgd”. In: *arXiv preprint arXiv:1711.04623* (2017).

[15] Stanisław Jastrzębski, Zachary Kenton, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. “On the relation between the sharpest directions of DNN loss and the SGD step length”. In: *arXiv preprint arXiv:1807.05031* (2018).

[16] Simran Kaur, Jeremy Cohen, and Zachary Chase Lipton. “On the maximum hessian eigenvalue and generalization”. In: *Proceedings on*. PMLR. 2023, pp. 51–65.

[17] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. “On large-batch training for deep learning: Generalization gap and sharp minima”. In: *arXiv preprint arXiv:1609.04836* (2016).

[18] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. “On large-batch training for deep learning: Generalization gap and sharp minima”. In: *arXiv preprint arXiv:1609.04836* (2016).

[19] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[20] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1 (1989), pp. 541–551.

[21] Sungyoon Lee and Cheongjae Jang. “A new characterization of the edge of stability based on a sharpness measure aware of batch gradient distribution”. In: *International Conference on Learning Representations*. 2023. URL: https://api.semanticscholar.org/CorpusID:259298833.

[22] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. “Visualizing the Loss Landscape of Neural Nets”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/a41b3bb3e6b050b6c9067c67f663b915-Paper.pdf.

[23] Warren Mcculloch and Walter Pitts. “A Logical Calculus of Ideas Immanent in Nervous Activity”. In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 127–147.

[24] Jinseong Park, Hoki Kim, Yujin Choi, and Jaewook Lee. “Differentially private sharpness-aware training”. In: *International Conference on Machine Learning*. PMLR. 2023, pp. 27204–27224.

[25] Barak Pearlmutter. “Fast Exact Multiplication by the Hessian”. In: *Neural Computation* 6 (Jan. 1994), pp. 147–160. DOI: 10.1162/neco.1994.6.1.147.

[26] Prin Phunyaphibarn, Junghyun Lee, Bohan Wang, Huishuai Zhang, and Chulhee Yun. “Gradient Descent with Polyak’s Momentum Finds Flatter Minima via Large Catapults”. In: *arXiv e-prints*, arXiv:2311.15051 (Nov. 2023), arXiv:2311.15051. DOI: 10.48550/arXiv.2311.15051. arXiv: 2311.15051 [cs.LG].

[27] Boris Polyak. “Some methods of speeding up the convergence of iteration methods”. In: *Ussr Computational Mathematics and Mathematical Physics* 4 (1964), pp. 1–17. URL: https://api.semanticscholar.org/CorpusID:120243018.

[28] Jorma Rissanen. “Modeling by shortest data description”. In: *Automatica* 14.5 (Sept. 1978), pp. 465–471. ISSN: 0005-1098. DOI: 10.1016/0005-1098(78)90005-5.

[29] Herbert E. Robbins. “A Stochastic Approximation Method”. In: *Annals of Mathematical Statistics* 22 (1951), pp. 400–407. URL: https://api.semanticscholar.org/CorpusID:16945044.

[30] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[31] Tijmen Tieleman and Geoffrey Hinton. “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude”. In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[33] Kaiyue Wen, Zhiyuan Li, and Tengyu Ma. “Sharpness minimization algorithms do not only minimize sharpness to achieve better generalization”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 1024–1035.

[34] Jiawei Zhang. “Gradient descent based optimization algorithms for deep learning models training”. In: *arXiv preprint arXiv:1903.03614* (2019).

[35] Xingyu Zhu, Zixuan Wang, Xiang Wang, Mo Zhou, and Rong Ge. “Understanding edge-of-stability training dynamics with a minimalist example”. In: *arXiv preprint arXiv:2210.03294* (2022).

Appendix

Assumptions for Theorem 6

Let $||\cdot||_{op}$ be the operator norm.
Assumption 3: Let ρ_3, ρ_4 to be the minimum constants such that for all θ, $||\nabla^3 L(\theta)||_{op} \leq \rho_3$ and $\nabla^3 L$ is ρ_4-Lipschitz with respect to $||\cdot||_{op}$. Then we assume that $\rho_4 = O(\eta\rho_3^2)$.

Assumption 4: For all t,

$$\frac{-\nabla L(\hat{\theta}_t) \cdot \nabla S(\hat{\theta}_t)}{||\nabla L(\hat{\theta}_t)||\,||\nabla S(\hat{\theta}_t)^{\perp}||} = \Theta(1) \quad \text{and} \quad ||\nabla S(\hat{\theta}_t)^{\perp}|| = \Theta(\rho_3).$$

Assumption 5: For all t and all $w, z \perp u(\hat{\theta}_t)$,

$$\frac{||\nabla^3 L(\hat{\theta}_t)(w,z)||}{||\nabla^3 L(\hat{\theta}_t)||_{op}||w||||z||}, \quad \frac{|H(\hat{\theta}_t)(\dot{v}_t^{\perp}, \dot{v}_t^{\perp})|}{||H(\hat{\theta}_t)||||\dot{v}_t^{\perp}||^2}, \quad \text{and} \quad \frac{|\lambda_{\min}(H(\hat{\theta}_t))|}{||H(\hat{\theta}_t)||_2} \leq O(\epsilon).$$