Title
Finding Critical and Gradient-Flat Points of Deep Neural Network Loss Functions

Author
Frye, Charles Gearhart

Publication Date
2020

Peer reviewed|Thesis/dissertation

Finding Critical and Gradient-Flat Points of Deep Neural Network Loss Functions

by

Charles Gearhart Frye

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Neuroscience

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Associate Professor Michael R DeWeese, Co-chair
Adjunct Assistant Professor Kristofer E Bouchard, Co-chair
Professor Bruno A Olshausen
Assistant Professor Moritz Hardt

Spring 2020

Finding Critical and Gradient-Flat Points of Deep Neural Network Loss Functions

Abstract

Finding Critical and Gradient-Flat Points of Deep Neural Network Loss Functions

by

Charles Gearhart Frye

Doctor of Philosophy in Neuroscience

University of California, Berkeley

Associate Professor Michael R DeWeese, Co-chair

Adjunct Assistant Professor Kristofer E Bouchard, Co-chair

Despite the fact that the loss functions of deep neural networks are highly non-convex, gradient-based optimization algorithms converge to approximately the same performance from many random initial points. This makes neural networks easy to train, which, combined with their high representational capacity and implicit and explicit regularization strategies, leads to machine-learned algorithms of high quality with reasonable computational cost in a wide variety of domains.

One thread of work has focused on explaining this phenomenon by numerically characterizing the local curvature at *critical points* of the loss function, where gradients are zero. Such studies have reported that the loss functions used to train neural networks have no local minima that are much worse than global minima, backed up by arguments from random matrix theory. More recent theoretical work, however, has suggested that bad local minima do exist.

In this dissertation, we show that one cause of this gap is that the methods used to numerically find critical points of neural network losses suffer, ironically, from a bad local minimum problem of their own. This problem is caused by *gradient-flat points*, where the gradient vector is in the kernel of the Hessian matrix of second partial derivatives. At these points, the loss function becomes, to second order, linear in the direction of the gradient, which violates the assumptions necessary to guarantee convergence for second-order critical point-finding methods. We present evidence that approximately gradient-flat points are a common feature of several prototypical neural network loss functions.

To the victims of empires, hegemonies, and masters, large and small, visible and invisible, across time and around the world. A better tomorrow is possible.

I found it hard, it's hard to find.
Oh well, whatever, nevermind.

Kurt Cobain,
Smells Like Teen Spirit, May 1991

Contents

List of Figures

List of Algorithms

Acknowledgments

The first and greatest debt I owe is to my mother, Dr. Anne Teresa Gearhart. She nurtured my love of inquiry, knowledge, and education, fighting to preserve it in a harsh and unwelcoming environment. This degree is the culmination of that love and so the product of hers. Against her heart but not her judgment, she let me leave home at just 15 to attend the boarding school of my dreams, the Illinois Math and Science Academy, and obtain the space to grow, at times away, but never too far. Mom: I love you and appreciate everything you have done for me. I will do my best to return the favor and to make you proud.

She also raised my two wonderful siblings, Teddy and Willow. Without them I would be a much less full person. Willow: here's to at least three more decades of disputation and cardiovascular exercise. Teddy: I miss you every day, even if we disagree about *Star Wars: The Last Jedi*.

As an undergraduate at the University of Chicago, I was blessed with incredible scientific mentorship, the value of which only becomes clearer with time. Dr. Margaret Wardle oversaw my undergraduate honors thesis and showed me how to be both a scientist and a full human being. Megin, I owe you my appreciation for the importance of a rigorous statistical method. Thanks for never asking why I was so interested in studying the psychopharmacology of party drugs. I still tell people about CyberBall. I worked simultaneously with Dr. Jason MacLean on a project that became the first scientific venture over which I felt full intellectual ownership. Jason, thanks for taking a chance on an undergraduate with more enthusiasm than sense who maybe shouldn't have been given control of a laser. You are my model of integrity, as a scientist and a person, in the face of immense pressure to publish flashy, shoddy results.

When I was deciding where to go to graduate school, Jason told me to ask myself: "where do the students seem most happy? Where is there a community you can see yourself thriving in?". I chose to do my PhD at Berkeley largely on that basis, and I can look back and see that the choice was the right one. I have been challenged to grow as a person and guided in that growth by the amazing community of students at Berkeley.

First, Dylan "ChicknMcGizzard" Paiton, PhD. From my first rotation to our latest paper, we have been a productive research team. From riding the bus at the Ward House to building yurts at the Dog House, we have been partners in crime. To more science and more crimes!

This thesis would not have been possible without Neha Wadia, who first introduced me to the problem of finding critical points of neural networks and who co-authored the papers that form the third chapter of this thesis. Neha, your wisdom, intellect, and integrity are an inspiration.

From my first days at UChicago to these last days at Berkeley, I have counted on the friendship and scholarship of Ryan Zarcone. Ryan, thank you for always being down to listen to half-finished speculations about mathematics, to give presentations, blog posts, and manuscripts trenchant feedback, and to grab some Taco Bell or a whiskey and chew the fat. To a life of Epicurean delights and Platonic contemplation in equal measure!

Dr.s Alexander Naka and Cameron Baker have been the Bubbles and Blossom to my Buttercup when they weren't busy being the SpongeBob and Patrick to my Squidward. Alex, even if you never convince me that neuroscience is not butts, you will forever be one of my closest friends. Cam: thank you for keeping me grounded, running buddy.

If not for Dr. Nicholas Ryder asking me, casually over a beer, how I might go about computing a square root, the second chapter of this thesis would be much impoverished. Nick, I treasure our conversations at least as much as I do the free food at OpenAI.

Among a blessed surplus of fabulous graduate student role models ahead of me in the program, including Dr. Brian Isett and Dr. Timothy Day, two stand out for the special contributions they made to my education in particular. Dr. Jesse Livezey, you have for years been the first person I go to when a quandary of ethics presents itself. Your and Sarah's commitment to justice and to compassion, even when it means hard work, is a bright light in a world with too much darkness. Dr. Natalia Bilenko, your light burns just as bright, and you were the best manager I ever had. Good luck and godspeed as you bring the machine learning community kicking and screaming into accepting responsibility.

It was an absolute privilege to be a part of the community of the Redwood Center for Theoretical Neuroscience. The special nature of Redwood as an interdisciplinary home for heretical neuroscientists was the other major factor that led me to choose to do my thesis work at Berkeley. My committee members, Dr.s Kristofer Bouchard, Michael DeWeese, and Bruno Olshausen, imbue the community with a peculiar sense of purpose: to question everything and demand nothing short of the right answer. It was in that heterodox and iconoclastic spirit that the work in this thesis was conducted. Dr. Yubei Chen's and Brian Cheung's commitments to scholarship, both fundamental and cutting edge, were direct role models. Pratik Sachdeva and Vasha DuTell, I'll catch you at the next stretch break. Michael Fang: if you need me, I'll be in my rabbit hole. Andrew Ligeralde and Jamie Simon, thank you for your help with this project, and good luck with your graduate work. Charles, thanks for the desk.

The community at Redwood helped me to grow into the scholar I am today. But that growth would not have been possible without the freedom afforded to me by the National Science Foundation's Graduate Research Fellowship, which funded me through half of my PhD, and the Berkeley Chancellor's Fellowship, which supported me in the first two years. I am deeply appreciative of the public's investment in me and in basic science, and I hope to repay it as best I can.

Looking outside the narrow, at times myopic, world of research, I would like to thank all of my wonderful friends, whose lives and passions and stories deeply enrich my own. To the saviors of Isger, the heroes who closed the Worldwound, and the adventurers who defeated the Forgotten Pharaoh Cult, I look forward to rolling strange polyhedra with you all a whole lot more in the future. To the members of the Hot Pony Social Club, both past and future, Daniel Mossing, Sara Popham, Storm Slivkoff, Sophie Obayashi, Dr. Nicholas Jourjine, and Dr. Kevin Doxzen: living with you was a pleasure and a privilege. To the VeryBestFriends, especially those who haven't been mentioned yet, Kelsey Howard, William Shainin, Christine Hopkins, Paul Cullen, Andres Vargas, Neil Wilson, Katherine Cording, Parker Conroy, and Maxine Marshall: I cherish you.

Chapter 1

Critical Points and the No-Bad-Local-Minima Theory of Neural Network Losses

1.1 Overview

It is typical to present only the polished final products of scientific research, rather than the process itself. This is exemplified by the advice to "write papers backwards", from the results and conclusions to the introduction and rationale. While this perhaps makes the research more digestible and certainly makes it more impressive, it hides the confusion and failure that are the day-to-day reality of research. In this brief overview, I will try to tell the story of the research project laid out in this thesis as it was experienced, warts and all, and in terms comprehensible to a wide audience. If you're only interested in the technical material, proceed to the next section, Section 1.2, to get started, or double back to the abstract for an overview.

Neural networks are machine learning systems that are, as of the writing of this thesis in 2020, widely used but poorly understood. The original purpose of the research project that became this dissertation was to understand how the architecture, dataset, and training method interact to determine which neural network training problems are easy. The approach was inspired by methods from chemical physics [9] and based on an analogy between a physical system minimizing energy and a machine learning system maximizing performance. Conceptually, the goal is to characterize all of the configurations in which the system is stable, the *critical points* or stationary points of the system. The details of this problem setup are the substance of the remainder of this chapter.

Our intent was to build on the work of [22] and [67], who had reported a characterization of critical points in some small, simple neural networks. We hoped to increase the scale of the networks closer to what is used in practice, to try more types of neural networks, and especially to examine the role of the dataset.

The first step in characterizing the critical points is finding them. In general, they can't be derived or written in elementary mathematical terms, and so need to be discov-

ered numerically, just as highly-performant neural networks have their parameters set by numerical algorithms rather than by analytical derivations. Chapter 2 is a didactic survey of algorithms for finding critical points. Our early attempts to reproduce the results in [22] and [67] appeared to be failures. The metric usually used to measure how close one is to a critical point, the squared gradient norm, stubbornly refused to decrease.

The algorithms used to find critical points are complicated — both in terms of their implementation and in terms of the number of knobs, or hyperparameters, we can twiddle to configure them. Furthermore, they behave quite differently from typical machine learning algorithms, and so intuition gained from experience working with those algorithms can be misleading. We had implemented these critical point-finding algorithms ourselves, due to the absence, at the beginning of this research project, of important technical tools in typical neural network software packages. We furthermore had limited expertise and experience in this domain, so our first thought was that we had either implemented the algorithms incorrectly or weren't configuring them properly.

As it turned out, both of those hypotheses were correct, but verifying them became a research project in itself. The key innovation was the introduction of the *deep linear autoencoder* as a test problem. For this very special neural network, the critical points actually are known mathematically, and have been since the late 80s [8]. With these "correct answers" in hand, we can check the work of our algorithms. These results were written up for the arXiV in [30] and rejected from ICML2019. They form the first part of Chapter 3.

Unfortunately, the process of debugging and tuning critical point-finding algorithms on the deep linear autoencoder did not solve our performance problems on non-linear networks. It remained the case that the squared gradient norm metric was abnormally high, along with other signatures of bad behavior on the part of our algorithms.

However, the exercise of verifying our algorithms on the deep linear autoencoder gave us the confidence to consider other, more fundamental causes for failure. In reviewing the literature on the methods used to find critical points, it became clear that a particular failure mode for these methods was not well-appreciated. Implicit in the literature on critical point-finding was the fact that, whenever a certain vector (the gradient) was mapped to 0 by a certain matrix (the Hessian), critical point-finding would fail [34, 12, 71]. We named this condition *gradient-flatness* and, on reviewing the outputs of our critical point-finding algorithms when applied to neural networks, we observed it ubiquitously. The concept of, evidence for, and consequences of gradient-flatness in neural networks are the focus of the second part of Chapter 3. These results were written up separately for the arXiV in [29].

The biggest take-home message of our observations for the field is that the famous results in [22] and [67] need an asterisk: the points characterized by those papers appear to be gradient-flat points, not critical points, which has distinct consequences for our understanding of neural networks.

In the remainder of this chapter, I will set up the problem of training neural networks and describe the critical point-based perspective on it, the no-bad-local-minima theory.

1.2 Neural Network Losses

Neural networks are a highly flexible class of differentiable functions suitable to a wide variety of machine learning tasks [47]. *Neural networks* are constructed by interleaving parameterized linear transformations with (optionally parameterized) non-linear transformations. In order to be able to apply calculus to our networks, we will assume that the parameters are stored in a vector, θ, and converted into a function by a network constructor[a] NN:

$$\mathrm{NN}(\theta)\colon \mathcal{X} \to \mathcal{Y} \tag{1.1}$$

While neural networks can be used for any machine learning task, we will focus on the important special case of supervised learning. In supervised learning, a collection of inputs X and targets Y are provided, and the goal is to find a function from $\mathcal{X}$ to $\mathcal{Y}$ such that the value of some cost function c is low, on average, when applied to matching pairs of targets and inputs after being passed through the network. Optionally, certain parameter values may be considered more "costly" than others, and this is enforced by a regularizer r that is added to the cost. The result is called the *loss function* and is defined below.

Definition 1.1: Loss Function

Given a neural network constructor NN, a cost function c, a regularizer r, and input data X and targets Y, we define the *loss function* as

$$L(\theta) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{x_i, y_i \in X, Y} c(\mathrm{NN}(\theta)(x_i), y_i) + r(\theta) \tag{1.2}$$

Note that, because the parameters are the thing over which we have control, we think of this as a function of the parameters θ, even though it might in other contexts be considered a function of the network architecture, the data, or both.

For example, a *fully-connected network* has as its parameters a tuple of weight matrices W_i and applies a non-linear function σ after applying each in turn:

$$\mathrm{NN}\left(W_1, \ldots W_{k-1}, W_k\right) = W_k \circ \sigma \circ W_{k-1} \circ \sigma \circ \ldots \sigma \circ W_1 \tag{1.3}$$

The process of "training" a neural network is the process of selecting a value of the parameters, $\theta^\star$, that minimizes the loss:

$$\theta^\star \in \operatorname*{argmin}_{\theta \in \Theta} L(\theta) \tag{1.4}$$

That is, we treat the process of inferring the best parameters for our network, the process of programming our machine-learned algorithm, as an optimization problem. This is the *variational approach*, which is a ubiquitous method in mathematics[b]. One might think

[a] The notation for this setup is summarized in Table 1.1.

[b] E.g. the Courant-Fischer-Weyl characterization of eigenvalues, the variational approach to inference [77], the Lagrangian approach to mechanics, and even the universal construction approach in category theory [57].

Name	Symbol : Type
Inputs	$X \colon \mathcal{X}^n \stackrel{\text{def}}{=} \mathbb{R}^{m \times n}$
Targets, Outputs	$Y \colon \mathcal{Y}^n \stackrel{\text{def}}{=} \mathbb{R}^{p \times n}$
Parameters	$\theta \colon \Theta \stackrel{\text{def}}{=} \mathbb{R}^N$
Loss Function	$L \colon \Theta \to \mathbb{R}$
Network Constructor	$\text{NN} \colon \Theta \to \mathcal{Y}^{\mathcal{X}}$
Cost Function	$c \colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$
Regularizer	$r \colon \Theta \to \mathbb{R}$

Table 1.1: **Definitions of Terms and Symbols for Neural Network Loss Functions**. The "type" of an object is either set of which it is an element or, for a function, the types of its inputs and outputs, denoted by input type $\to$ output type. The symbol $\mathcal{Y}^{\mathcal{X}}$ denotes the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$.

of machine learning in general as a variational approach to programming computers. An element of an argmin is known as a *global minimum*. Finding global minima is generically a hard problem in the strictest sense, precisely because almost any problem in mathematics can be formulated as an optimization problem. The variational approach is particularly useful when the resulting optimization problem has a fast solution algorithm.

Almost all methods for optimizing neural networks are gradient-based. That is, they use the gradient function, which satisfies the following relation:

Definition 1.2: Gradient Function

The *gradient function* of a function $L \colon \Theta \to \mathbb{R}$ is denoted $\nabla L \colon \Theta \to \Theta$ and satisfies

$$L(\theta + \varepsilon) = L(\theta) + \langle \nabla L(\theta), \varepsilon \rangle + o(\varepsilon) \tag{1.5}$$

for all $\theta, \varepsilon \in \Theta$. The values returned by this function are called *gradients* or *gradient vectors*.

If a method only uses the function and the gradient, we call it a *first-order* method.

The gradient at a point θ is a vector that can be used as a linear functional applied to a perturbation ε that approximates the value of L at $\theta + \varepsilon$. If we drop the little-o term, we obtain this linear approximation, also known as a first-order Taylor expansion:

$$\widehat{L}(\theta + \varepsilon) = L(\theta) + \langle \nabla L(\theta), \varepsilon \rangle \tag{1.6}$$

Since $\widehat{L}$ is unbounded, minimizing it would mean selecting an infinitely-long step ε in a direction with negative inner product with the gradient. But our goal is to minimize L, not $\widehat{L}$, and as ε grows, so does the approximation error $o(\varepsilon)$, and so we select a finite length vector. We choose the one that makes that inner product most negative, for its length. This is the negative gradient. We then typically apply a scaling factor η, called the *learning rate* or *step size*. The resulting optimization method, defined in Algorithm 1,

is known as *gradient descent*.

Algorithm 1: Gradient Descent

Require $\theta_0 \in \Theta, \eta \in \mathbb{R}^+, \nabla L \colon \Theta \to \Theta, T \in \mathbb{N}^+$
while $t < T$ **do**
  $\theta_{t+1} \leftarrow \theta_t - \eta \nabla L(\theta_t)$
  t $\leftarrow t + 1$
end

This method and its stochastic and accelerated variants have some hope of working on smooth functions because whenever the parameter is a minimizer, the gradient is 0:

$$\theta^\star \in \operatorname*{argmin}_{\theta \in \Theta} L(\theta) \Rightarrow \nabla L(\theta^\star) = 0 \tag{1.7}$$

and so, if initialized from a minimizer, any gradient-based algorithm will stay there. Below, we will demonstrate that for convex smooth functions, this algorithm, for an appropriate choice of η, will converge to a minimizer from a random initial point.

Unfortunately, neural network loss surfaces are not convex, and so the theory built up around convex optimization (see [16, 12]) would suggest that training neural networks should be hard. And indeed, the experience of practitioners working on neural networks in the 80s and 90s was that training them was difficult. Nowadays, however, it is recognized that training large neural networks with gradient-based methods is actually quite easy, in that many problems can be avoided with a few generic tricks [75]. One key hypothesis as to why is the no-bad-local-minima theory. To understand it, we need to consider the kinds of structures that can appear in a non-convex function, and which of them are compatible with gradient-based optimization.

1.3 Critical Points

One approach to analyzing the behavior of optimization algorithms is to split the task of determining convergence into two steps: first, identify the points which are *stationary*, at which the update is 0, and then determine which of those points are actual targets of convergence. We call the stationary points of the loss for the gradient descent algorithm its critical points.

Definition 1.3: Critical Points

The set of all *critical points* of a loss function L on a domain Θ is denoted Θ_{cp}^L and defined as

$$\Theta_{\text{cp}}^L \stackrel{\text{def}}{=} \{\theta \in \Theta \colon \nabla L(\theta) = 0\} \tag{1.8}$$

When unambiguous, the super-script L will be omitted.

In a naïve first pass, it would seem that all $\theta \in \Theta_{\text{cp}}$ are also targets of convergence. If we initialize Algorithm 1 to one of these points ($\theta_0 \in \Theta_{\text{cp}}$), then $\theta_t = \theta_0$ for all t.

Therefore, if initialized from a critical point, the algorithm will converge to that critical point.

This picture is mis-leading for practical purposes, but even this coarse approach is sufficient to guarantee that gradient-based methods converge on smooth convex functions. A smooth function f is convex iff $f(y) \geqslant f(x) + \langle \nabla f(x)(y-x) \rangle$ for all x and y in its domain Ω. The loss functions for linear regression and logistic regression are convex, including when convex regularization is applied, e.g. LASSO or ridge [36]. See Figure 1.1.

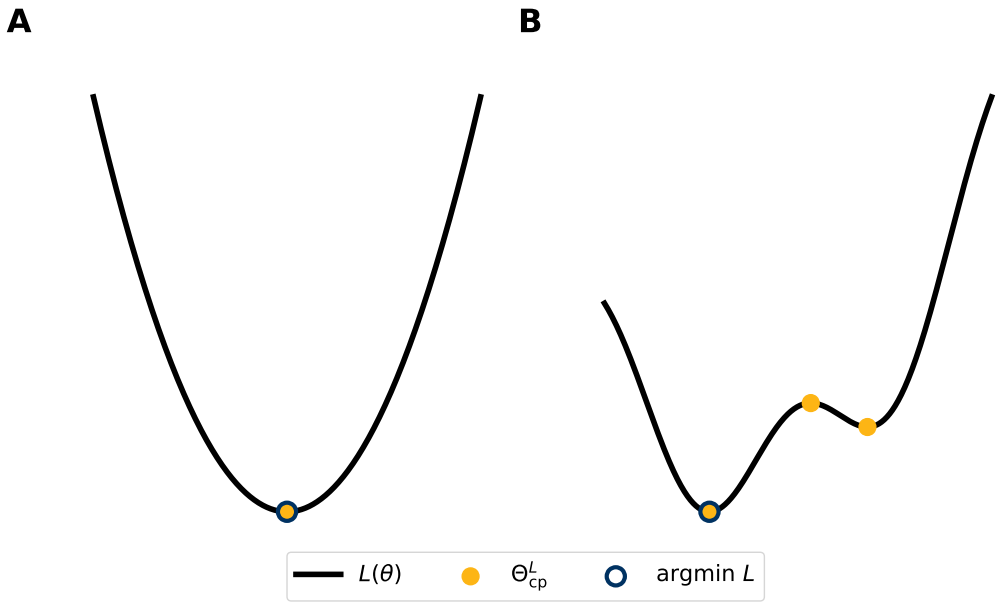


Figure 1.1: **The Critical Points and Global Minima of a Convex and a Non-Convex Function**
A: The convex function $L(\theta) = \theta^2$ (black) and its critical points (gold) and minimizers (blue outline). **B**: Same as in A, but for the non-convex function $L(\theta) = \cos(\theta) + \theta + \theta^2$.

Applying this definition of convexity at a point x_{cp} among the critical points of f, Ω^f_{cp}, we have that

$$f(y) \geqslant f(x_{\text{cp}}) + \langle \nabla f(x_{\text{cp}}), y - x_{\text{cp}} \rangle \tag{1.9}$$
$$f(y) \geqslant f(x_{\text{cp}}) + 0 = f(x_{\text{cp}}) \tag{1.10}$$

for *any* point y. This is part of the power of convexity: local information (encoded in the gradient) gives global information (in the form of a global lower bound). With it, we can improve 1.7 from a one-way implication to a biconditional, from "if-then" to "if and only if":

$$L \text{ smooth, convex } \Rightarrow \theta \in \underset{\theta \in \Theta}{\text{argmin}}\, L(\theta) \Leftrightarrow \nabla L(\theta) = 0 \Leftrightarrow \theta \in \Theta_{\text{cp}} \tag{1.11}$$

Another way to characterize smooth convex functions is through their Hessian function. The Hessian function returns matrices that satisfy the relation below:

Definition 1.4: Hessian Function

The *Hessian function* of $L\colon \Theta \to \mathbb{R}$ is denoted $\nabla^2 L\colon \Theta \to \mathbb{R}^{N\times N}$, where N is the dimensionality of Θ, is the function that satisfies

$$\nabla L\left(\theta+\varepsilon\right) = \nabla L\left(\theta\right) + \nabla^2 L\left(\theta\right)\varepsilon + o(\varepsilon) \tag{1.12}$$

for all $\theta, \varepsilon \in \Theta$. The matrices returned by this function are called *Hessians* or *Hessian matrices*.

It is the "gradient of the gradient", in that it returns a linear function (a matrix) that approximates the gradient function, which itself returns a linear functional (a vector) that approximates the original scalar function.

Combined, the gradient function and the Hessian function produce a quadratic approximation of the original function L:

$$L\left(\theta+\varepsilon\right) = L(\theta) + \left\langle \nabla L\left(\theta\right), \varepsilon\right\rangle + \frac{1}{2}\varepsilon^\top \nabla^2 L\left(\theta\right)\varepsilon + o(\|\varepsilon\|^2) \tag{1.13}$$

Note that the Hessian appears in Equation 1.13 as a *quadratic form*: a symmetric matrix pre- and post-multiplied with the same vector. Quadratic forms are classified, up to a change of basis, by the eigenvalue spectrum of the underlying matrix: the number of positive, negative, and 0 eigenvalues[c]. We will later classify critical points in the same fashion.

Smooth convex functions are precisely those functions whose Hessian matrix has no negative eigenvalues at any point. Such a matrix M is called *positive semi-definite*, denoted[d] $M \succeq 0$. If $M \succ 0$, then its smallest eigenvalue is positive, and the matrix is *positive definite*.

If the Hessian M at a point is positive definite then $x^\top M x$ is always positive. This implies that the second-order term in Equation 1.13 is positive. Since the second-order term dominates the higher-order terms for sufficiently small ε, at a point θ^* where the gradient is 0 and the Hessian is positive definite, we have that

$$\begin{aligned} L\left(\theta^*+\varepsilon\right) &= L(\theta^*) + \left\langle \nabla L\left(\theta^*\right), \varepsilon\right\rangle + \varepsilon^\top \nabla^2 L\left(\theta^*\right)\varepsilon + o(\|\varepsilon\|^2) && (1.14)\\ &= L(\theta^*) + 0 + \varepsilon^\top \nabla^2 L\left(\theta^*\right)\varepsilon + o(\|\varepsilon\|^2) && (1.15)\\ &\geqslant L(\theta^*) && (1.16) \end{aligned}$$

Such a θ^* is a called a *local minimum*, since it is a minimizer of L in all its neighborhoods under a given size.

[c] Because the matrices are real and symmetric, there are no complex eigenvalues.

[d] Specifically, $\succeq$ is the Loewner partial order on symmetric matrices. $A \succeq B$ if the smallest eigenvalue of $A - B$ is greater than or equal to 0. $\succ$ is defined using strict inequality.

Definition 1.5: Local Minima

The set of all *local minima* of a scalar function L on a domain Θ is denoted Θ^L_{lm} and defined as

$$\Theta^L_{\text{lm}} \stackrel{\text{def}}{=} \{\theta \in \Theta \colon L(\theta + \varepsilon) \geqslant L(\theta)\} \tag{1.17}$$

for some $\varepsilon > 0$. When unambiguous, the super-script L will be omitted.

By a small extension of the above argument[e], we come to a final characterization of why gradient descent converges on smooth, convex functions[f]: all critical points are local minima and all local minima are also global minima, i.e. elements of the argmin:

$$f \colon \Omega \to \mathbb{R} \text{ smooth, convex } \Rightarrow \Omega_{\text{lm}} = \operatorname*{argmin}_{\Omega} f = \Omega_{\text{cp}} \tag{1.18}$$

From this, we can deduce that any algorithm that converges to a generic critical point will converge, on smooth convex functions, to a minimizer.

This condition is sufficient, but not necessary, for gradient descent to converge to a local minimizer. The sticking point is when there are critical points which are not local minimizers: $\Theta_{\text{cp}} \supset \Theta_{\text{lm}}$. Does gradient descent converge to non-minimizing critical points, or only to local minimizers?

Reviewing the second order approximation of the loss in Equation 1.13, we see that, at a critical point θ_{cp}, the first order term vanishes

$$L(\theta_{\text{cp}} + \varepsilon) = \frac{1}{2}\varepsilon^\top \nabla^2 L\left(\theta_{\text{cp}}\right) \varepsilon + o(\|\varepsilon\|^2) \tag{1.19}$$

leaving only the terms of second order and above. We can therefore classify critical points according to the eigenvalue spectrum of their associated Hessian. The fraction of negative eigenvalues is known as the *index* of the critical point.

Definition 1.6: Index

For a critical point $\theta \in \Theta^L_{\text{cp}}$, we define the *index* as the number of negative eigenvalues of the Hessian of L at θ:

$$I(\theta) = \frac{1}{N} \sum_{\lambda_i \in \Lambda(\nabla^2 L(\theta))} \mathbb{I}\left(\lambda_i < 0\right) \tag{1.20}$$

where $\Lambda \colon S\mathbb{R}^{k \times k} \to \mathbb{R}^k$ is a function that takes in a symmetric real matrix and returns its eigenvalues as a vector and $\mathbb{I}$ is the indicator function.

[e]The extension demonstrates that, for convex functions, points with positive semi-definite Hessians are still minimizers. See [12]. This is untrue in the non-convex case, and checking whether a point is a minimum becomes NP-Hard at worst, see [58].

[f]In fact the class of functions for which the implication 1.18 holds is broader. They are known as smooth *invex* functions, see [39].

Points with index strictly between 0 and 1 are *strict saddle points*. These are points where the gradient is 0 and the local curvature is upwards in some directions and downwards in others. See Figure 1.2 for an example. If all eigenvalues are non-zero, then critical points with index 0 are local minima and critical points with index 1 are local maxima. If some eigenvalues are zero and the index is 0, then the critical point may be a local minimum or may be a (non-strict) saddle point, but higher-order derivatives are needed to disambiguate.

If all saddle points are strict saddle points, a condition known as the *strict saddle property*, then gradient descent converges to a local minimizer [50, 49]. Furthermore, convergence for a stochastic version of Algorithm 1 is fast [42] and can be accelerated via momentum [41].

This is a convenient property, but is it satisfied by any losses of practical interest? As an illustrative example, consider the quartic loss function of two variables, Example 1.1 below, which is based on perhaps the simplest neural network. The example is artificial—it is using machine learning to multiply by 1, which is excessive even in the contemporary era of ML hype —but it is closely related to principal components analysis (PCA), as we will see in Chapter 2. More generally, the strict-saddle property is satisfied by tensor decomposition problems [31], which covers a number of latent variable models, including PCA, independent components analysis (ICA [11, 21]), sparse coding [62] and other forms of dictionary learning, and Hidden Markov models [3].

In many of these cases, all local minima are also global minima, and so gradient descent is sufficient for those optimization problems. The question of whether this holds for neural network loss functions is the subject of the next section.

1.4 The No-Bad-Local-Minima Theory

When all local minima of a function are approximately global minima, we will say that the function has the *no-bad-local-minima property*, or the NBLM property. While Example 1.1 has the NBLM property, the function in panel B of Figure 1.1 does not, due to the presence of a non-global local minimum. We will refer to the hypothesis that the loss functions of large neural networks satisfy the NBLM property as the *no-bad-local-minima theory*.

Informally, the argument goes as follows: at any critical point, imagine that the eigenvalues of the Hessian are drawn randomly. If there is even a small chance that any eigenvalue is negative, then for a sufficiently large network, there will be almost surely at least one negative eigenvalue, by the strong law of large numbers. When the value of the loss is low, we might expect that the probability of a positive eigenvalue becomes 1, while when the value of the loss is high, negative and positive values are both possible. A random loss function drawn according to these rules will have the NBLM property with a probability that rapidly increases with dimension.

In Section 1.4.1, we will cover the first model of neural network loss functions meant to formalize this intuition, the Gaussian random field model of [22]. Then, in Section 1.4.2, we will explain how this model was improved by incorporating observations about the spectrum of the Hessians of neural networks [67]. Then, we will discuss, in Section 1.4.3,

Example 1.1: 1-Dimensional Deep Linear Autoencoder

In Definition 1.1, take $X = Y = 1$, $c(y, \widehat{y}) = (y - \widehat{y})^2$, $r(\theta) = 0$, and $\mathrm{NN}(\theta) = \theta_2\theta_1$ for $\Theta = \mathbb{R}^2$. The resulting loss function L is

$$L(\theta_1, \theta_2) = (\theta_2\theta_1 - 1)^2 \tag{1.21}$$

In more standard terminology, it corresponds to the choice of the mean squared error cost function and no regularization for a linear autoencoder network with one-dimensional inputs and a one-dimensional hidden layer.

The function is not convex, because the gradient does not provide a global lower bound at some points, e.g., at the origin where it is the zero vector. The origin is a critical point that is not a minimizer (the loss takes on value $1 > 0$) but instead a strict saddle point. However, this function can still be optimized effectively with gradient-based methods.

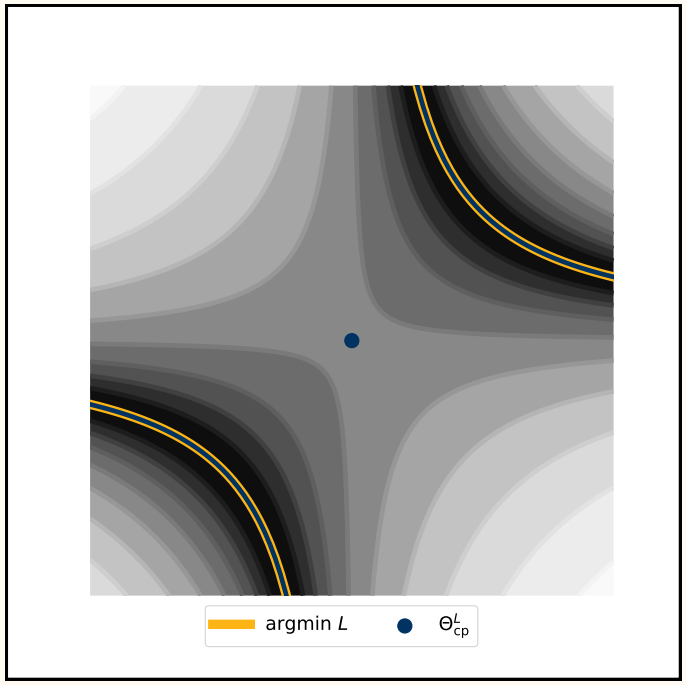


Figure 1.2: **A Loss Function that Satisfies the Strict Saddle Property**. Values of the function defined in Example 1.1 on a domain centered at the origin. The value of L is represented by color, with low values in black and high values in white. Contours are logarithmically-spaced for illustrative purposes. Critical points in blue, and global minima in gold. The isolated critical point at the origin is a strict saddle.

weaknesses of and negative analytical results for the broadest versions of the NBLM theory, and close with a review of the alternatives, in Section 1.4.4.

1.4.1 The Gaussian Random Field Model of Neural Network Losses

One of the simplest interesting classes of random functions that have the NBLM property is the class of *Gaussian random fields*. Gaussian random fields are random functions from $\mathbb{R}^n$ to $\mathbb{R}$. An example of a single draw from a Gaussian random field defined on $\mathbb{R}^2$ appears in Figure 1.3.

With most random variables, discussion can proceed directly from writing down a density. But with random functions, the situation is more complicated, because it is difficult to consider densities and integrals over the space of functions. For one, it necessitates that we typically visualize random functions by looking at a single example realization, as in Figure 1.3, and hoping that any properties we notice are "typical", rather than drawing a histogram or density. The difficulty in working with densities further necessitates a somewhat strange definition for the Gaussian random field. The definition is clearer if we first introduce an analogous definition for multivariate Gaussian random variables.

Definition 1.7: Multivariate Gaussian

A random k-dimensional vector v is *multivariate Gaussian* if the random variable $w^\top v$ is Gaussian-distributed for all $w \in \mathbb{R}^k$.

As a corollary, the family of multivariate Gaussians is closed under all linear transformations, since the composition of a linear transformation and an inner product is equal to an inner product with another vector.

We can similarly define a Gaussian random field by requiring that the vectors obtained by evaluating the random function at any set of test points have the same property.

Definition 1.8: Gaussian Random Field

A random function f from $\mathbb{R}^k \to \mathbb{R}$ is a *Gaussian random field* if the random variable $w^\top f^{\otimes n}(x_1 \dots x_n)$ is Gaussian-distributed for all $w \in \mathbb{R}^n, x_i \in \mathbb{R}^k$ and for every $n \in \mathbb{N}$.

Here $f^{\otimes n}$ is the n-fold product of the function f, which applies the function f to n separate inputs, returning n outputs[g].

There is one example of a Gaussian random field that is familiar to machine learning practitioners: the Gaussian process.

[g]Specifically, $f^{\otimes n}(x_1, \dots, x_n)_i = f(x_i)$ for all i from 1 to n.

Example 1.2: Well-Known Gaussian Processes

Choose $k = 1$ in the Definition 1.8. This special case of a Gaussian random field is called a *Gaussian process*.

If $f(x_i)$ is independent of $f(x_j)$ for all $x_i, x_j \in \mathbb{R}$, then f is a *white noise Gaussian process*.

The cumulative integral of a white noise Gaussian process is also a Gaussian process, called a *Wiener process* or a *random walk*.

Consider a single unit in a neural network with independent, Gaussian random weights θ. The family of Gaussian random variables is closed under linear combinations, so the random variable $\theta^\top x$ is Gaussian for all x. Such a neuron is therefore a Gaussian process. With care, this can be extended into a model for deep nonlinear neural networks with very large layers and non-Gaussian random weights, see [40].

Usefully, a Gaussian random field can be defined in terms of its mean function $\mu \colon \mathbb{R}^k \to \mathbb{R}$ and covariance function, or kernel, $K \colon \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$, which must be positive semi-definite. For many choices of kernel, the field is smooth (and its partial derivatives are also Gaussian random fields). See Figure 1.3, which depicts a single draw from the ensemble of smooth Gaussian random fields with a particular squared exponential, or Gaussian, kernel. Notice that it has local minima, saddles, and maxima. If we are to use this ensemble as a model of neural network losses that can explain their optimizability, we would like a statistical description of these critical points.

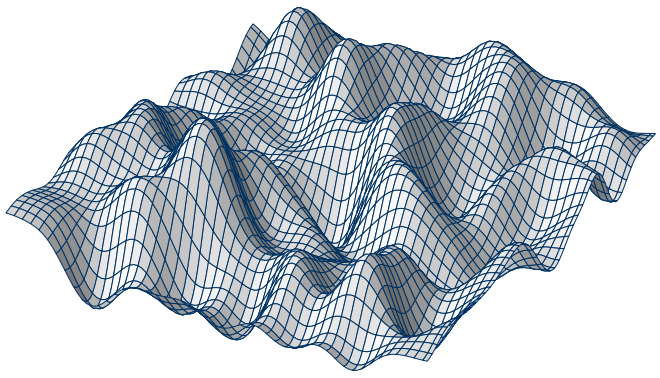


Figure 1.3: **2-Dimensional Gaussian Random Field**.
Note that this is a single realization of a Gaussian random field, akin to a single observation of a random variable.

The key work on the critical points of smooth Gaussian random fields is [13]. Their core result showed that the expected index (Definition 1.6) of critical points is a tight function of the field's value at that point[h]. Furthermore, for many kernels, including the squared exponential kernel, the expected index is an increasing function of the loss, and the deviations are even smaller[i].

As a consequence, if a high-dimensional loss function is a typical member of the this random field ensemble, then when the loss is high, the number of negative eigenvalues at critical points is high, and so the critical points are all saddles or maxima. Below a certain value of the loss, all of the critical points will be minima. Furthermore, the minima, including the global minima, will have approximately the same loss. This is an effective restatement of the no-bad-local-minima property.

One benefit of this model is that it applies only to high-dimensional loss functions. This would explain the difference in optimizability of contemporary neural networks, where parameter counts ($\dim \Theta$) are in the hundreds of thousands or millions, and neural networks in the 80s and 90s, which were orders of magnitude smaller in size.

The hypothesis that neural network loss surfaces might be well-modeled by typical members of the Gaussian random field ensemble was first put forward in [22]. They reported numerical results on the critical points of two neural network losses that were in qualitative agreement with the NBLM property of Gaussian random fields. We will later see (Chapter 3) that these results have a caveat to them, due to weaknesses of the numerical methods. Because [22] predated, and to some extent inspired, results on the convergence of gradient descent in the presence of saddles [50, 42], it focused on defining an optimization method, saddle-free Newton, that was clearly repelled by saddles.

1.4.2 Improving the Gaussian Random Field Model with Wishart Matrices

In this section, we will see how the Gaussian random field model misses the mark in predicting the spectrum of neural network loss Hessians and develop a better model, following [67]. We begin by considering the random matrix ensembles from which the Hessians of the Gaussian random field model are drawn. For a fuller treatment of the theory of random matrices, including proofs of the statements below regarding spectra, see [27, 76].

The eigenvalues of random matrices enjoy a similar phenomenon to the Central Limit Theorem for sums of random variables: when the right independence assumption is made, the details of the random variables don't matter, and the result has a stereotypical distribution. For averages of random variables, that stereotypical distribution is the Gaussian. For eigenvalues of random matrices, it is the Wigner semi-circle distribution.

First, let's define the ensemble of random matrices associated with this distribution: the symmetric Wigner random matrices.

[h] Deviations of order $O(1/\sqrt{N})$, where N is the dimension of the field.

[i] Deviations of order $O(1/N)$.

Definition 1.9: Symmetric Wigner Random Matrix

A random square $n \times n$ matrix M is *symmetric Wigner* if the random entries M_{ij} satisfy the following:

$$\begin{aligned} &\forall i, j \colon 1 \leqslant i < j \leqslant n \\ &\quad \mathbb{E}\left[M_{i,j}\right] = 0 \\ &\quad \mathbb{E}\left[M_{i,j}^2\right] = 1/n \\ &\quad M_{j,i} = M_{i,j} \text{ and } M_{i,j} \text{ iid} \\ &\forall i \\ &\quad \mathbb{E}\left[M_{i,i}\right] = \gamma \\ &\quad \mathbb{E}\left[M_{i,i}^2\right] = 1/n \\ &\quad M_{i,i} \text{ i.i.d} \end{aligned}$$

where iid is short-hand for independent and identically-distributed and $\gamma \in \mathbb{R}$. Note that the entries along the diagonal, defined in the second block, may have a different distribution than the entries off the diagonal, defined in the first block, but both groups are iid.

A single draw from this ensemble is pictured in Figure 1.4, along with its observed eigenvalue distribution and cumulative distribution. As the size of the matrix n goes to ∞, the observed eigenvalue distribution converges, in the almost sure sense, to a Wigner semicircular distribution.

Definition 1.10: Wigner Semicircular Distribution

The (γ-shifted) *Wigner semicircular distribution* is the distribution associated with the density $\nu \colon [-2+\gamma, 2+\gamma] \to \mathbb{R}$, defined as

$$\nu(x) = \frac{1}{2\pi}\sqrt{4 - (x-\gamma)^2} \tag{1.22}$$

for $\gamma \in \mathbb{R}$. When the shift parameter γ is 0, we drop the prefix of "shifted".

This distribution is symmetric about the control parameter γ. For $\gamma = 0$, this means that, for a large symmetric Wigner matrix, approximately half of the eigenvalues will be negative and half of the values will be positive. As γ changes, the fraction of eigenvalues above and below zero changes.

We can now be a bit more specific about the results of [13]. They found that the Hessian spectra of critical points of Gaussian random fields have γ-shifted Wigner semicircular distributions whose shift parameter γ depends on the value of the random field at that critical point. This gives an index that is similarly dependent.

How well does this spectrum match up against the observed eigenvalues of neural network loss Hessians? The expected distribution is a density: i.e. it has no atoms, or

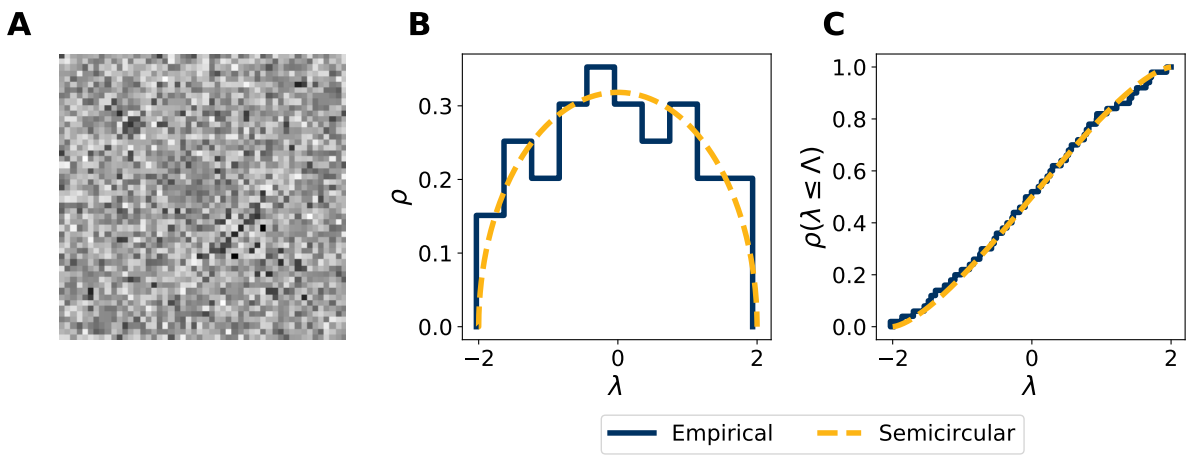


Figure 1.4: **The Wigner Semicircular Distribution**.
A: An example 50×50 symmetric Wigner random matrix drawn from the ensemble in Definition 1.9. Entries are Gaussian-distributed. More negative values in black, more positive values in white. **B**: The expected spectral density of the matrix in A (gold), which is given by Equation 1.22, and the histogram of observed eigenvalues (blue). Note the close match. **C**: The expected cumulative spectral distribution of the matrix in A (gold), which is the integral of the density in B, and the cumulative fraction of observed eigenvalues (blue). Again, the match is close even for a fairly small matrix.

single points with finite probability mass, as is typical for "well-behaved" continuous random variables[j]. In particular, there is no atom at 0, and so the probability that a random matrix has an *exactly* zero eigenvalue is 0. This means that their kernel, defined below, consists only of the zero vector, and the matrices almost surely have inverses.

Definition 1.11: Kernel and Co-Kernel

For a matrix $M \in \mathbb{R}^{m \times n}$, we define the *kernel* of M, denoted $\ker M$, as the set of all vectors mapped to 0 by M:

$$\ker M \stackrel{\text{def}}{=} \{v \in \mathbb{R}^n : Mv = 0\} \tag{1.23}$$

We define the *co-kernel* of M, denoted $\operatorname{co}\ker M$, as the subspace spanned by all vectors not mapped to 0 by M:

$$\operatorname{co}\ker M \stackrel{\text{def}}{=} \{v \in \mathbb{R}^n : Mv \neq 0\} \cup \{0\} \tag{1.24}$$

These two subspaces are *complementary*: their union is $\mathbb{R}^n$ and their dimensions add up to n. The *rank* of M, $\operatorname{rk} M$, is equal to the dimension of $\operatorname{co}\ker M$. A matrix with rank less than the minimum of its input and output dimension is said to have *low rank*.

[j]In rigorous terms, the distribution is *absolutely continuous* with respect to the Lebesgue measure.

The full rank of Wigner random matrices is in stark contrast to that of Hessian matrices of typical neural network loss functions [72]. There are many 0 eigenvalues, along with some eigenvalues at much larger values, with fewer and lower magnitude negative eigenvalues. Furthermore, the low-rank structure of the Hessian matrix can be read off from the clear patterns in the values, unlike the clearly independent (aside from symmetry) values in the matrix in Figure 1.4.

There is an alternative random matrix ensemble that has some of these properties: the Wishart random matrix ensemble.

Definition 1.12: Wishart Random Matrix

A random square $n \times n$ matrix M is *Wishart* if it can be constructed as

$$M = XX^\top \tag{1.25}$$

where X is a random $n \times k$ matrix with iid Gaussian entries

$$X_{ij} \sim \text{Normal}\,(0, 1/k) \tag{1.26}$$

It is the sample covariance matrix of a k-element sample of an n-dimensional Gaussian random vector with iid components.

The rank of a Wishart random matrix is almost surely equal to the minimum of n and k. When k is less than n, the matrix is low-rank.

The expected spectrum of a Wishart random matrix follows the Marçenko-Pastur distribution with parameter $\gamma = \frac{n}{k}$, defined below. This distribution places non-zero probability mass at 0, and so produces non-invertible matrices. It also includes a density over positive values, and so its definition is somewhat technically involved. A single draw from this ensemble is pictured in Figure 1.5, along with its observed eigenvalue distribution and cumulative distribution. Note the presence of a large bulk at 0.

Definition 1.13: Marçenko-Pastur Distribution

The *Marçenko-Pastur distribution* with parameter γ is defined as

$$\nu(X) = \begin{cases} \mathbb{I}\,(0 \in X)\left(1 - \frac{1}{\gamma}\right) + \mu(X), & \text{if } \gamma > 1 \\ \mu(X), & \text{if } \gamma \leqslant 1 \end{cases} \tag{1.27}$$

where the measure μ, which is absolutely continuous, is defined by its associated density $d\mu\colon\ [\lambda_-, \lambda_+] \to \mathbb{R}$:

$$d\mu(\lambda) = \frac{1}{2\pi} \frac{\sqrt{(\lambda_+ - \lambda)\,(\lambda - \lambda_-)}}{\gamma\lambda} d\lambda \tag{1.28}$$

where $\lambda_\pm = \left(1 \pm \sqrt{\gamma}\right)^2$.

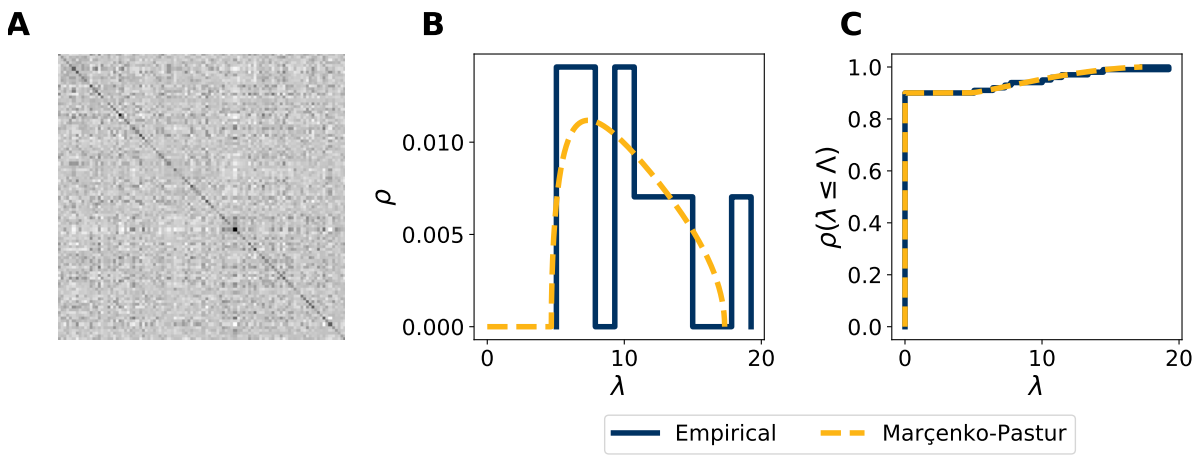


Figure 1.5: **The Marçenko-Pastur Distribution**.
A: An example 100×100 Wishart random matrix with rank $k = 10$ drawn from the ensemble in Definition 1.12. More negative values in black, more positive values in white. **B**: The expected spectral density of the matrix in A (gold) outside of the bulk at 0, which is given by Equation 1.28, with parameter $\gamma = 100/10 = 10$. and the histogram of observed eigenvalues (blue). Note the close match. **C**: The expected cumulative spectral distribution of the matrix in A (gold), which is the value of $\nu([0, \lambda])$ (defined in Equation 1.27) and the cumulative fraction of observed eigenvalues (blue).

But this matrix ensemble is on its own inadequate for modeling neural network loss Hessians, since its members are always positive semi-definite, as the support of the density $d\mu$ in Equation 1.28 is non-negative. A random function whose Hessians are members of this ensemble will always be convex.

The authors of [67] proposed a parameterized mixture of Wigner and Wishart random matrices as a better qualitative model of neural network loss Hessians. The random Hessians are a sum of random matrices: a Wishart part that contributes singularity and large positive eigenvalues and a Wigner part that contributes moderately-sized positive and negative eigenvalues. Just as the Fourier transform allows us to calculate the distribution of a sum of independent random variables from their separate distributions, the Stieltjes and $\mathcal{R}$ transforms combine to allow the calculation of the expected spectrum of a sum of freely independent random matrices from knowledge of the separate expected spectra. In this model, the relative weight of the Wishart part increases as the loss decreases, so that at low loss the Hessian is almost surely positive semi-definite, while it is almost surely indefinite at higher values of the loss. The details of this derivation are in [67], along with a more sophisticated model for the loss of a single hidden-layer ReLU network.

Even without any additions, this model is able to better match the qualitative features of neural network loss Hessians (singularity, large positive eigenvalues) while retaining the NBLM property. Furthermore, [67] includes numerical experiments indicating some match

between the model and the loss and index of critical points in a moderately-sized neural network (though we will see, in Chapter 3, that this evidence is somewhat weak).

1.4.3 Criticism of the No-Bad-Local-Minima Theory

While the NBLM theory agrees qualitatively and in some cases quantitatively with observations of neural network loss functions, it is not without its flaws.

First, the approach based on critical points and smooth Gaussian random fields fails in the case of non-smooth activations or cost functions. The popular ReLU activation function, $\text{ReLU}(x) = \max(x, 0)$, is non-smooth because it is not differentiable at 0, as is the *hinge* cost function for binary classification, $c(y, \widehat{y}) = \max(1 - y \cdot \widehat{y}, 0)$, where the two class labels are ± 1. Non-smooth functions may have local minima without having any points where the gradient is 0, as in the Euclidean norm $\|v\|$, which specializes to the absolute value in the one-dimensional case. Because neural networks rely on gradients for optimization, it is typical for activations and costs to be differentiable almost everywhere, so it might seem that this is a mere technicality. However, it can be shown that for networks with that activation and cost, all local minima are either non-differentiable or constant [46]. An analysis based on differentiable minima would only consider the constant minima, which would give an overly-optimistic picture of the loss for many non-smooth losses.

Second, it can be shown that the strict form of the NBLM theory is false. For networks with ReLU neurons, the theory is clearly false: if the parameters are such that all of the inputs to the ReLU function are 0 in one layer, e.g. if the biases are large and negative, then the loss is locally constant but may have arbitrarily high value. Recent work has further shown that, if biases are intialized to small values, ReLU networks converge to bad local minima [37].

For some time, it was believed that in the smooth case, despite negative results for small networks, e.g. [7], a sufficiently-large network had no bad local minima, thanks to a proof in [80]. However, this proof was incorrect, as shown in [51], and only implies the non-existence of *strict* local minima, where the Hessian is positive definite. Further work demonstrated that non-strict bad local minima are generically present for a wide variety of activation functions, not just non-smooth activations, for arbitrarily large widths [23].

This suggests that the rosy picture painted by the numerical results in [22] and [67] was somewhat mis-leading. It remains possible, however, that with the right intialization, these bad local minima are avoided, and so some form of the NBLM theory holds, perhaps for the loss restricted to a region in the vicinity of the initial point.

1.4.4 Alternatives to the No-Bad-Local-Minima Theory

The alternative method to demonstrate the convergence of gradient-based methods on neural network losses is to prove it directly, rather than relying on generic optimization proofs by demonstrating that the loss function falls in some privileged class. There are two closely-related approaches that have born fruit in recent years: an asymptotic

approach, based on the Neural Tangent Kernel, and a direct approach, based on over-parameterization.

1.4.4.1 Over-Parameterization

The random matrix theory approach outlined in Section 1.4.1 and Section 1.4.2 was motivated by the observation that larger networks were easier to train. To some, this suggested that there were special properties of generic (random, from the right ensemble) high-dimensional functions.

Alternatively, it could be the case that the optimizability of large neural networks follows directly from their size. Both [1] and [25] prove that gradient descent applied to a neural network loss converges to within ε of a global minimum from a random initialization, provided that the number of neurons in each layer is polynomial in a problem-dependent set of parameters. Furthermore, this occurs within a polynomial number of steps in terms of a similar set of parameters. In both cases, the convergence rate in terms of ε is the same as in traditional proofs for gradient methods (i.e. linear, [12]). The result in [25] has an exponential dependence on depth for networks without residual connections, which is absent in [1].

While these are useful proofs of principle, polynomial dependence is still too great to be of practical use if the degree of the polynomial is high, as is noted in [1]. For example, one of the polynomial terms for the width in [25], which is applied to the number of samples, has degree 4. For the MNIST dataset [48], which has 6e4 examples, this implies a minimum width of more than 1e19 neurons, up to linear factors. The same parameter in [1] has degree 30. Another term in [25], $\lambda_{\min}$, can be exponentially small in terms of other natural parameters. The regime in which these claims operate is clearly outside the realm of practically implementable networks.

1.4.4.2 The Neural Tangent Kernel

With widths so large, they might as well be infinite. In the infinite-width limit, neural networks with random weights become Gaussian random fields as in Section 1.4.1, thanks to a form of Central Limit Theorem. Though the correspondence between shallow Bayesian neural networks and Gaussian processes dates back to the 1990s [59], it was only recently that this correspondence was extended to an extremely broad class of neural networks, dubbed *Tensor Programs* [79]. This view enabled the automated computation of posteriors from these networks [61], akin to the automated computation of gradients provided by automatic differentiation packages. With the right tricks [52], these posteriors can be competitive with more traditional networks.

Importantly for the optimization perspective, this approach can be used to analyze the convergence of non-Bayesian neural networks. Indeed, in this view, neural networks trained by stochastic gradient descent from random initialization are actually undergoing *kernel gradient descent*, which descends a convex function [40], even when the loss is a non-convex function of the parameters.

While this approach is exciting and provides a fundamentally different way to think about and train neural networks, it is asymptotic. Whether the claims apply to finite-sized networks was at first unclear. The authors of [6] demonstrated equivalence between certain finite-width networks and their limit with high probability. However, the minimum layer width was again polynomial in certain problem parameters, and again the degrees of the polynomials were high (4+). It remains to be seen whether tighter bounds can be proven.

1.5 Conclusion

Neural networks are, in many cases, easily trained to approximate global minima from random initializations. This suggests that their loss functions may have a no-bad-local-minima property: all local minima are nearly global minima. This NBLM property is shared by a large class of random functions, the Gaussian random fields [13, 22]. Though the typical Wigner and Wishart ensembles of random matrices are not sufficient alone to model neural network loss functions, even qualitatively, an ensemble derived by mixing them is [67].

However, analytical results [23] indicate that this picture is incorrect and that, analytically speaking, neural network losses have bad local minima, even if they aren't found by typical training procedures. Alternative approaches based on overparameterization [1] and its infinite-width limit [40] have suggested a different path to understanding the trainability of neural networks, but they require unreasonably large hidden layers.

In the absence of strong theoretical results, it is important to obtain better empirical evidence, building on the work of [22] and [67]. In Chapter 2, we will develop algorithms for examining the critical points of neural network loss functions. In Chapter 3, we will apply these algorithms to some example neural network losses, and observe something interesting: due to numerical and analytical issues, previously-used algorithms may not have been characterizing the critical points of loss functions, but instead another class of points, gradient-flat points.

Chapter 2

Second-Order Critical Point-Finding Algorithms

2.1 Chapter Summary

In order to characterize the local curvature properties of neural network loss functions at critical points, we need first to find critical points. In this chapter, we review two classes of algorithms for finding critical points: gradient norm minimization methods and Newton methods. All of these algorithms are numerical, iterative methods. They are based on finding the zero values of linear approximations of the gradient function. These linear approximations are given by the Hessian function. We refer to them as *second-order* critical point-finding algorithms.

While the analytical critical points exactly satisfy the analytical system of (possibly non-linear) equations

$$\nabla L(\theta) = 0 \tag{2.1}$$

numerical approaches cannot guarantee exact equality. Instead, they can at best guarantee *approximate* equality,

$$\|\nabla L(\theta)\|^2 \leqslant \varepsilon \tag{2.2}$$

for some $\varepsilon > 0$. So our algorithms will actually recover approximate critical points:

Definition 2.1: ε-Critical Points

The set of all *ε-critical points* of a loss function L on a domain Θ for a fixed $\varepsilon \geqslant 0$ is denoted $\Theta^L_{\varepsilon\text{-cp}}$ and defined as

$$\Theta^L_{\varepsilon\text{-cp}} \stackrel{\text{def}}{=} \left\{\theta \in \Theta \colon \|\nabla L(\theta)\|^2 \leqslant \varepsilon\right\} \tag{2.3}$$

When unambiguous, the super-script L will be omitted. See Definition 1.3.

If only introduced in the case of large-dimensional Θ and complex loss function L, this definition and the problem setup can be somewhat intimidating and intuition can be hard to build. Section 2.2 demonstrates a close analogy between finding critical points and a

fundamental arithmetic operation: taking the square root. Specifically, taking the square root will require us to define approximate square roots, set up a surrogate optimization problem based on an inequality that relaxes an equality, and develop an inexact, iterative method for scalar inversion (Section 2.5.1).

This running example is used to intuitively motivate each of three algorithm classes in turn: gradient norm minimization methods (Section 2.3), exact Newton methods (Section 2.4), and inexact Newton methods (Section 2.5). Practical versions of Newton methods for high-dimensional and non-linear problems require additional tricks, which are reviewed in Section 2.6.

The presentation of the Newton methods is intentionally didactic and lengthy. This is because these methods are relatively unfamiliar to the neural network community. First, second-order optimization methods, like Newton methods for convex functions, are not commonly used to train neural networks. Second, Newton methods for critical point-finding are different in motivation and implementation than those used in convex optimization. It is hoped that the thorough elaboration of these methods in this chapter will help ease the incorporation of these methods into the toolkits of future neural network researchers.

2.2 Optimization Approach to Taking Square Roots

Addition ($+$) and multiplication ($\times$) are simple operations, in the following sense: given exact binary representations for two numbers, an exact binary representation for the result of $+$ or $\times$ applied to those two numbers can be obtained in finite time[a]. The symbols $a+b$ and $a \times b$ represent the exact, finite output of a concrete, finite-time algorithm. That is, both operations define closed monoids over binary strings of (importantly!) finite length.

This is not true of division ($\div$), inversion ($^{-1}$), or taking the square root[b] ($\sqrt{}$). In these cases, the operation is actually defined in terms of a promise regarding what happens when the output of this operation is subjected to $\times$:

$$b = a \div c \implies b \times c = a \tag{2.4}$$

$$b = a^{-1} \implies b \times a = 1 \tag{2.5}$$

$$b = \sqrt{a} \implies b \times b = a \tag{2.6}$$

and for an exact representation of a number a, the number that fulfills this promise might not have an exact representation, as is famously the case for $\sqrt{2}$. This makes algorithm design for these operations more complex than for $+$ and $\times$.

There are individual strategies for each, but one general idea that turns out to be very powerful is *relaxation to an optimization problem*. That is, we take the exact promises

[a]Specifically, for n-digit numbers addition is $O(n)$ and multiplication is $O(n \log n)$ [35], courtesy of the Fast Fourier Transform and the convolution theorem.

[b]We take the type signature of $\sqrt{}$ to be $\sqrt{}: \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$, i.e. we do not allow non-real or negative square roots.

made above, recognize them as statements of the optimality of the output for some criterion, and then use that criterion to define an approximate promise, true up to some tolerance ε.

For $\sqrt{}$, we rearrange the promise 2.6, denoting its exact solution with a $\star$, into:

$$b^\star \stackrel{\text{def}}{=} \sqrt{a} \tag{2.7}$$

$$b^\star \times b^\star = a \tag{2.8}$$

$$b^\star \times b^\star - a = 0 \tag{2.9}$$

$$\|b^\star \times b^\star - a\|^2 = 0 \tag{2.10}$$

and then recognize that, due to the non-negativity of the norm, Equation 2.10 is a statement of optimality about $b^\star$. That is, the value $b^\star$ that we are looking for is the argument that minimizes the expression on the LHS:

$$b^\star = \operatorname*{argmin}_{b \in \mathbb{R}_{\geqslant 0}} \|b \times b - a\|^2 \tag{2.11}$$

Exactly minimizing this expression to arbitrary precision might be impossible, so we instead consider a set of approximate square roots, to a tolerance ε:

$$B_\varepsilon \stackrel{\text{def}}{=} \left\{ b \in \mathbb{R}_{\geqslant 0} \colon \|b \times b - a\|^2 \leqslant \varepsilon \right\} \tag{2.12}$$

and so the problem of square root finding is the problem of finding a member of B_ε.

Analogously, the problem of critical point-finding is the problem of *gradient root finding*: instead of trying to find a value b such that $b \times b - a$ is approximately 0, we are trying to find a value θ such that $\nabla L(\theta)$ is approximately 0. This θ is a member of $\Theta^L_{\varepsilon\text{-cp}}$, Definition 2.1.

2.3 Gradient Norm Minimization

The simplest way to attack the problem of finding a point in the sets defined by Equation 2.3 or Equation 2.12 is to apply a first-order optimization algorithm. In the case of critical point-finding, we define an auxiliary function G, equal to half the squared gradient norm:

$$G(\theta) = \frac{1}{2} \|\nabla L(\theta)\|^2 \tag{2.13}$$

and apply our first-order optimization algorithm to it.

This approach turns out to work poorly, so it is not widely disseminated enough to have a standard name. However, it is straightforward enough to be repeatedly discovered. The earliest use, to the best of the author's knowledge, was in chemical physics [54], where the problem of finding saddle points arises when computing transition states of chemical reactions. They gave it the name *gradient norm minimization*, which we use here. For several reasons, an alternative method for solving the problem of finding transition states called eigenvector-following became popular instead [18]. Gradient norm minimization

was then reinvented by two groups simultaneously in 2000 ([4, 14]) and its poor performance quickly noted [24]. The method was apparently then reinvented by the authors of [67] and applied to the problem of finding critical points of neural networks.

While automatic differentiation can compute the derivatives of g with no issue, it is still instructive to know its gradients, so let's derive them by hand.

Theorem 2.1: Gradient of Squared Gradient Norm

Let L be a twice-differentiable function from Θ to $\mathbb{R}$ and define $G(\theta) = \frac{1}{2}\|\nabla L(\theta)\|^2$. Then the gradient function of G, $\nabla G(\theta)$, is given by

$$\nabla G(\theta) = \nabla^2 L(\theta)\, \nabla L(\theta) \tag{2.14}$$

Proof of Theorem 2.1:
If we apply G to a point $\theta + \varepsilon$,

$$G(\theta + \varepsilon) = \frac{1}{2}\|\nabla L(\theta + \varepsilon)\|^2 \tag{2.15}$$

we see that the right-hand-side contains the same expression as the left-hand-side of the definition of the Hessian function, Equation 1.12. We can therefore replace that expression with the decomposition on the right-hand-side:

$$G(\theta + \varepsilon) = \frac{1}{2}\|\nabla L(\theta) + \nabla^2 L(\theta)\, \varepsilon + o(\varepsilon)\|^2 \tag{2.16}$$

Then, we expand the squared norm into its constituent terms, group like terms[c], and simplify:

$$G(\theta + \varepsilon) = \frac{1}{2}\|\nabla L(\theta)\|^2 + \underbrace{\frac{1}{2}\|\nabla^2 L(\theta)\, \varepsilon\|^2 + \frac{1}{2}\|o(\varepsilon)\|^2}_{o(\varepsilon)} + \nabla L(\theta)^\top \nabla^2 L(\theta)\, \varepsilon + o(\varepsilon) \tag{2.17}$$

$$G(\theta + \varepsilon) = G(\theta) + \left(\nabla^2 L(\theta)\, \nabla L(\theta)\right)^\top \varepsilon + o(\varepsilon) \tag{2.18}$$

which implies, by pattern-matching to Definition 1.2, that the gradient function of g is

$$\nabla G(\theta) = \nabla^2 L(\theta)\, \nabla L(\theta) \tag{2.19}$$

■

The minimal form of gradient norm minimization applies Algorithm 1 with these gradients, giving rise to Algorithm 2.

[c] Importantly, any ε inside a squared norm makes that term $o(\varepsilon)$, as does any $o(\varepsilon)$.

Algorithm 2: Gradient Norm Minimization by Gradient Descent

Require $T \in \mathbb{N}$, $\alpha \in \mathbb{R}_{>0}$, $\theta_0 \in \mathbb{R}^N$, $\nabla f \colon \mathbb{R}^N \to \mathbb{R}^N$, $\nabla^2 f \colon \mathbb{R}^N \to \mathbb{R}^{N \times N}$

$t \leftarrow 0$

while $t < T$ **do**

  $g \leftarrow \nabla f(\theta_t)$

  $H \leftarrow \nabla^2 f(\theta_t)$

  $\theta_{t+1} \leftarrow \theta_t - \alpha H g$

  $t \leftarrow t + 1$

end

However, one advantage of the gradient norm minimization approach is that, because it is framed as the optimization of a scalar-valued function, any number of tricks from the world of optimization can be applied to it: stochastic methods, momentum, adaptive gradients, and so on. This is in contrast to the Newton methods reviewed below. In Chapter 3, we will apply back-tracking line search [5] along gradient descent directions with the Wolfe conditions for line search termination [78] as our optimization algorithm. For more on back-tracking line search, see Section 2.6.2 and [12, Section 9.2].

2.4 Exact Newton Methods

Gradient norm minimization does not take any advantage of the special structure of our optimization problem, namely that it was introduced as a proxy for solving a system of equations. The other major class of algorithms for critical point-finding, *Newton methods*, are specifically designed to solve systems of equations. In this section, we review the fundamental, textbook forms of the method. Then, in the following sections, Section 2.5 and Section 2.6, we introduce extensions that make it feasible for problems with large dimension and higher-order nonlinearity, like many neural network loss functions.

2.4.1 The Babylonian Algorithm

The very earliest form of Newton's method dates back perhaps to the 17th century BCE, based on the evidence of an extremely precise (to within 2e-6) approximation to the square root of 2 on a Babylonian clay tablet. It was certainly known as a method for computing square roots by the first century CE, when it appears in the work of Hero of Alexandria. See [15] for details.

The intuition for this algorithm is geometric: if a square of area x is smaller (larger) than a square with sides of length θ, then it is larger (smaller) than the square with sides of length $x \div \theta$. In either case, the square with sides equal in length to the average of θ and $x \div \theta$ is closer in area to x (see Figure 2.1). Therefore, its side length is a closer approximation to $\sqrt{x}$. This improvement can be applied iteratively, yielding the below algorithm:

Algorithm 3: Babylonian Algorithm

Require $T \in \mathbb{N}$, $x, \theta_0 \in \mathbb{R}^2_{>0}$ 1
$t \leftarrow 0$ 2
while $t < T$ **do** 3
  $\theta_{t+1} \leftarrow \frac{\theta_t}{2} + \frac{x \div \theta_t}{2}$ 4
end 5

Recall that the motivation for considering the square root problem was that $\sqrt{}$, unlike $+$ and $\times$, does not have an exact finite-time algorithm. This is also true of $\div$, which appears in Algorithm 3. We will come back to this in Section 2.5.1, in order to better understand inexact Newton methods. Note that this algorithm converges extremely quickly, as indicated by Figure 2.1B-C. The error goes from order 1 to order 1e-10 in just 5 iterations, then drops to 0.

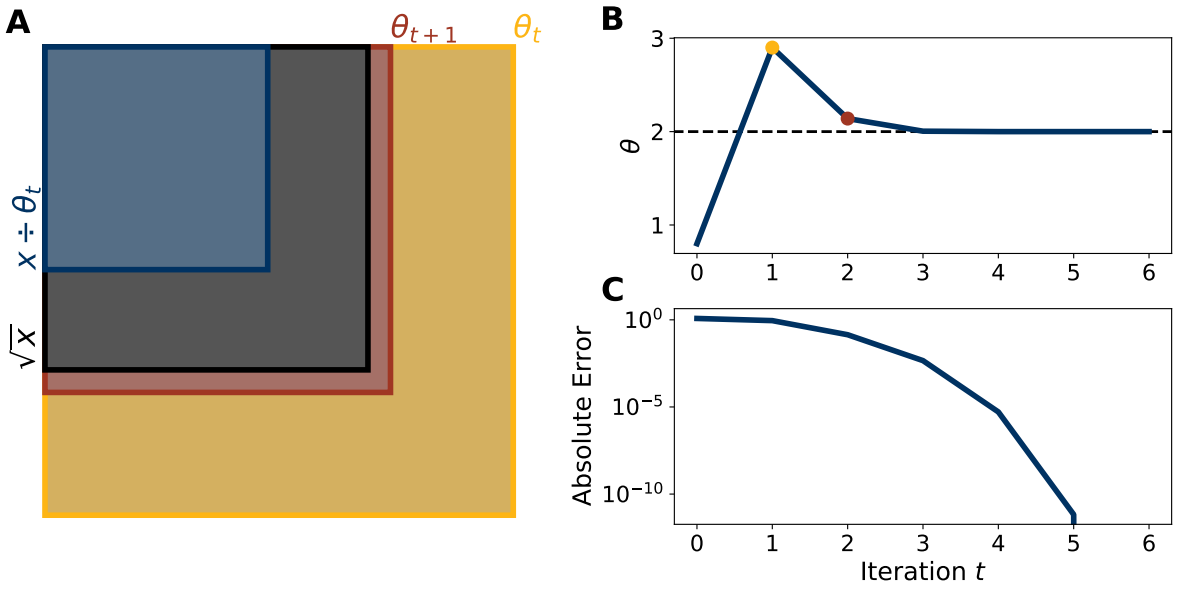


Figure 2.1: **The Babylonian Algorithm for Computing Square Roots**
A: Depiction of geometric intuition for the method. The square with side length θ_t (gold) is larger than that with side length $\sqrt{x}$ and area x (black), while that with side length $x \div \theta_t$ (blue) is smaller. The average of these two values is the value θ_{t+1}, which results in a square closer in area to x (red). **B**: Value of θ across iterations of Algorithm 3 for arguments $T = 6$, $x = 4$, and $\theta_0 = 0.8$. The value of 4 was chosen because the exact square root is known, unlike e.g. 2. The iterations indicated by color correspond to values in A. x-axis is shared with C. **C**: Absolute value of the difference between θ_t and $\sqrt{x}$ for Babylonian algorithm as in B, plotted on log-scale. Difference at iteration 6 is 0, and so is plotted off the chart.

2.4.2 Newton-Raphson

If we define

$$f(\theta) = x - \theta^2 \tag{2.20}$$

then the update in Algorithm 3 can be written

$$\theta_{t+1} = \theta_t - \frac{x - \theta_t^2}{2\theta_t} \tag{2.21}$$

$$= \theta_t - \frac{f(\theta_t)}{\nabla f\left(\theta_t\right)} \tag{2.22}$$

which can be defined for any differentiable f, so long as $\nabla f\left(\theta_t\right) \neq 0$. In this form, it is known as the *Newton method* or as *Newton-Raphson*. This is an interesting example of Stigler's law of eponymy [73]: Thomas Simpson was the first to notice that this algorithm could be generalized using calculus, in 1740. However, in part because Isaac Newton had developed an algebraic version of the method for generic polynomials and Joseph Raphson had refined it, Joseph Fourier, Carl Runge, and other prominent mathematicians mistakenly credited them with the technique. See [44] for details.

In the case of critical point-finding, the equivalent of f is a vector and already a gradient, and so ∇f becomes the Hessian matrix and the scalar division becomes multiplication by an inverse matrix. The resulting algorithm is as follows:

Algorithm 4: Newton-Raphson

Require $T \in \mathbb{N}$, $\theta_0 \in \mathbb{R}^N$, $\nabla f \colon \mathbb{R}^N \to \mathbb{R}^N$, $\nabla^2 f \colon \mathbb{R}^N \to \mathbb{R}^{N \times N}$ 1
$t \leftarrow 0$ 2
while $t < T$ **do** 3
  $g \leftarrow \nabla f\left(\theta_t\right)$ 4
  $H \leftarrow \nabla^2 f\left(\theta_t\right)$ 5
  $\theta_{t+1} \leftarrow \theta_t - H^{-1} g$ 6
  $t \leftarrow t + 1$ 7
end 8

While the problem setup here was for equation solving, this algorithm, in particular under the name Newton's method, is better known as an optimization algorithm. It works as such for the same reason that gradient descent can be a global optimizer on smooth convex functions: whenever all of the stationary points are also global optima, an algorithm for finding stationary points is also an optimization algorithm. Unlike gradient descent, it is a second-order algorithm: in addition to a gradient function, ∇f, it also needs a Hessian function, $\nabla^2 f$.

In the absence of geometric intuition to explain why this is a sensible algorithm, we turn to calculus. First, we show that this is the best method possible for functions with constant Hessian function.

Theorem 2.2: Optimality of Newton Step

Let f be a twice-differentiable function from Θ to $\mathbb{R}$ with constant, non-singular Hessian function. From any point θ, a point $\theta^\star$ that solves the gradient equations 2.1 for f can be obtained by computing

$$\theta^\star = \theta - \nabla^2 f(\theta)^{-1} \nabla f(\theta) \tag{2.23}$$

Proof of Theorem 2.2:
The value of the gradient at a point $\theta + p$ can be approximated by applying the Hessian at θ to p:

$$\nabla f(\theta + p) = \nabla f(\theta) + \nabla^2 f(\theta)\, p + o(p) \tag{2.24}$$

By Taylor's theorem, $o(p)$ is governed by the integral of the third derivative of f evaluated from p to θ. But the Hessian is constant, and so the third derivative is a zero tensor and $o(p)$ can be replaced with 0:

$$\nabla f(\theta + p) = \nabla f(\theta) + \nabla^2 f(\theta)\, p \tag{2.25}$$

To solve the gradient equations, we need the right-hand side here to be equal to 0. Denoting the solution $p^\star$, we find that

$$0 = \nabla f(\theta) + \nabla^2 f(\theta)\, p^\star \tag{2.26}$$

$$p^\star = -\nabla^2 f(\theta)^{-1} \nabla f(\theta) \tag{2.27}$$

from which the claim follows, with $\theta^\star = \theta + p^\star$.

■

We call the value $p^\star$ the *pure Newton step* or the *exact Newton step*. Note the absence of any step size. We call the linear system of equations in p defined by

$$0 = \nabla f(\theta) + \nabla^2 f(\theta)\, p \tag{2.28}$$

the *Newton system* of equations.

For functions with non-constant Hessian, i.e. those which cannot be represented by a degree two polynomial, this argument doesn't hold, and the pure Newton step is not optimal in the same sense. However, it holds approximately for functions whose Hessian doesn't change too quickly. In fact, the rate of convergence for this algorithm, once sufficiently close to a critical point, is quadratic: for each iteration, the number of bits of precision doubles[d]. This is to be contrasted with gradient descent, which has at best linear improvement: with each iteration, the number of bits of precision increases by a fixed amount[e]. This rapid convergence is visible in Figure 2.1C, where the quality of the solution doubles, from approximately 1e-5 to approximately 1e-10, between iterations 4 and 5. This rate is so fast that for practical purposes, the number of steps required during this phase can be bounded from above by 6 [12, Section 9.5]. See [60, Chapters 2 & 11] for proofs of this convergence rate in various settings (in particular, Theorem 11.7).

[d] In terms of the error, this is much faster than quadratic: it is squared exponential.

[e] Again, in terms of the error, this is much faster: it is exponential.

2.4.3 Pseudo-Inverse Newton

In defining the Babylonian algorithm, Algorithm 3, it was important that the starting point not be 0, or else the update would be undefined due to division by 0. Equivalently, the full Newton method Algorithm 4 required that the Hessian was non-singular, else the inverse is undefined.

But this is not a fundamental restriction, which is lucky because the loss functions of neural networks are highly singular [72]. The key property of the update $p^\star$ was that it solved the Newton system

$$0 = \nabla f(\theta) + \nabla^2 f(\theta)\, p \tag{2.29}$$

which has a solution whenever $-\nabla f(\theta)$ is in the image of $\nabla^2 f(\theta)$, the linear subspace of possible outputs of $\nabla^2 f(\theta)$, defined below.

Definition 2.2: Image and Co-Image of a Matrix

For a matrix M in $\mathbb{R}^{m\times n}$, we define the *image* of M, denoted $\operatorname{im} M$, as

$$\operatorname{im} M \stackrel{\text{def}}{=} \{v \in \mathbb{R}^m \colon \exists w \colon Mw = v\} \tag{2.30}$$

and the *co-image* of M, denoted $\operatorname{co\,im} M$, as

$$\operatorname{co\,im} M \stackrel{\text{def}}{=} \{v \in \mathbb{R}^m \colon !\exists w \colon Mw = v\} \tag{2.31}$$

When the rank (Definition 1.11) of M is at least m, the co-image of M is empty.

In this case, the solution $p^\star$ can be obtained by applying the (Moore-Penrose) pseudo-inverse [68] of the Hessian to the gradient. We will define this pseudo-inverse by means of the singular value decomposition. We will need several properties of this matrix later, so we do this in detail.

2.4.3.1 Pseudo-Inverses and the Singular Value Decomposition

The singular value decomposition of a matrix breaks down a matrix into the product, or composition, of three matrices.

Definition 2.3: Singular Value Decomposition

For a matrix M in $\mathbb{R}^{m\times n}$ with rank r, we define the *singular value decomposition* or SVD of M as the triple of matrices $V^\top$, Σ, U such that

$$M = U\Sigma V^\top \tag{2.32}$$

where $V^\top \in \mathbb{R}^{r\times n}$, $\Sigma \in \mathbb{R}^{r\times r}$, and $U \in \mathbb{R}^{m\times r}$ and U and V are orthonormal and Σ is diagonal. The columns of U are called the *left-singular* vectors of M, the columns of V the *right-singular*. The diagonal entries of Σ are the *singular values* of M. This form is sometimes called the *compact SVD*.

It can be shown that every matrix has a singular value decomposition [74]. The singular value decomposition arises from the specialization of the First Isomorphism Theorem (aka the canonical decomposition, see [2, Theorem 2.7]) to the category of vector spaces and linear maps. We state it here in terms of functions, for readers not familiar with category theory, but it is more appropriately set in terms of morphisms.

Theorem 2.3: The First Isomorphism Theorem

Let $\mathcal{C}$ be a collection of sets and all functions between them. For any function $f\colon A \to B$, where f, A, and B are members of the collection, there is a triple of functions s, b, and i such that

$$f = i \circ b \circ s \tag{2.33}$$

where $s\colon A \twoheadrightarrow A/\sim$ is a surjection, $b\colon A/\sim \xrightarrow{\sim} \operatorname{im} A$ is a bijection, and $i\colon \operatorname{im} A \hookrightarrow B$ is an injection (surjectivty and injectivity indicated by arrow type). The elements of the equivalence relation $\sim$ are given by pairs $(a, a') \in A \times A$ where $f(a) = f(a')$.

The First Isomorphism Theorem states that every function can be decomposed into three constituent pieces: an onto mapping, or surjection, that classifies its inputs according to which output they are mapped to; a one-to-one mapping, or bijection, that picks the output corresponding to each class of input; and an into mapping, or injection, that inserts this output into the target set.

Though this decomposition is of limited use for generic functions, as opposed to structure-preserving functions like group homomorphisms or linear maps, it is fruitful to consider a concrete example from the world of generic functions before connecting to the SVD.

Example 2.1: Decomposition of `is_odd`

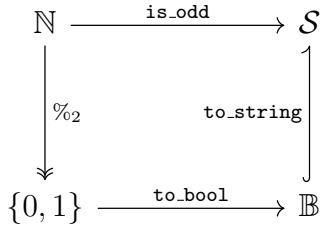
Consider the function $\texttt{is\_odd}\colon \mathbb{N} \to \mathcal{S}$, where $\mathcal{S}$ is the set of all strings in the English alphabet, that returns "True" if the number is odd, "False" otherwise. This function can be decomposed as

$$\texttt{is\_odd} = \texttt{to\_string} \circ \texttt{to\_bool} \circ \%_2 \tag{2.34}$$

where $\%_2\colon \mathbb{N} \twoheadrightarrow \{0, 1\}$ computes the value mod 2 and

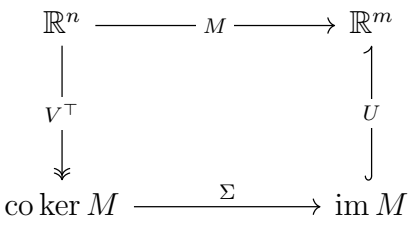
$$\texttt{to\_string}(b) = \begin{cases} \text{"False"} & \text{if } b = \bot \\ \text{"True"} & \text{if } b = \top \end{cases} \qquad \texttt{to\_bool}(k) = \begin{cases} \bot & \text{if } k = 0 \\ \top & \text{if } k = 1 \end{cases}$$

This decomposition is summarized neatly by the commutative diagram below:



Saying that this is a commutative diagram means that any path from one domain to another results in the same function.

Now, let us view the SVD through this lens. First, $V^\top$ is a surjection because it is orthonormal and $r \leqslant n$. Second, U is an injection because it is orthonormal and $m \geqslant r$. Finally, Σ is a bijection because it is a diagonal matrix with non-zero diagonal entries. This establishes that the (compact) SVD is the same decomposition as in the First Isomorphism Theorem. Furthermore, for linear functions, two inputs can only be mapped to the same output if they are both mapped to 0. If we take 0 to be the representative of that equivalence class, the target of the surjection becomes the co-kernel of M (recall Definition 1.11). The target of the bijection is the image of M. This set of relationships is summarized in the commutative diagram below.



We are now ready to define the pseudo-inverse. We first define it for two simple classes of matrices, then extend it for all matrices by means of the SVD.

Definition 2.4: Moore-Penrose Pseudo-Inverse

Let O be an orthogonal matrix and let D be a diagonal matrix of full rank. We define the pseudo-inverses of O and D, denoted O^+ and D^+, as

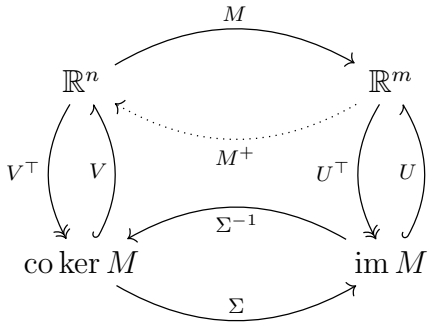
$$O^+ \stackrel{\text{def}}{=} O^\top,\ D^+ \stackrel{\text{def}}{=} D^{-1} \tag{2.35}$$

Let M be a matrix in $\mathbb{R}^{m\times n}$ and let its singular value decomposition be the triple V, Σ, U. We define the *pseudo-inverse* of M, denoted $M^+ \colon \mathbb{R}^m \to \mathbb{R}^n$, as

$$M^+ \stackrel{\text{def}}{=} \left(V^\top\right)^+ \Sigma^+ U^+ \tag{2.36}$$

$$= V\Sigma^{-1}U^\top \tag{2.37}$$

This is sometimes known as the Moore-Penrose pseudo-inverse. The relationships between these matrices are summarized by the commutative diagram below.



2.4.3.2 Defining Pseudo-Inverse Newton

Let's return to the problem of solving Equation 2.29 for an update $p^\star$. We see that whenever a vector g is in the image of the Hessian, $\nabla^2 f(\theta)$, we can apply the pseudoinverse, $\nabla^2 f(\theta)^+$, to $-g$ to obtain a vector that is the pre-image[f] of $-g$ with respect to the Hessian. We choose this as our update, obtaining the algorithm below.

Algorithm 5: Pseudo-Inverse Newton

Require $T \in \mathbb{N}$, $\theta_0 \in \mathbb{R}^N$, $\nabla f \colon \mathbb{R}^N \to \mathbb{R}^N$, $\nabla^2 f \colon \mathbb{R}^N \to \mathbb{R}^{N \times N}$ 1
$t \leftarrow 0$ 2
while $t < T$ **do** 3
  $g \leftarrow \nabla f(\theta_t)$ 4
  $H \leftarrow \nabla^2 f(\theta_t)$ 5
  $\theta_{t+1} \leftarrow \theta_t - H^+ g$ 6
  $t \leftarrow t + 1$ 7
end 8

We will return to the problem of what happens when the gradient is not in the image of the Hessian in Chapter 3. This will turn out to be critically important for understanding the behavior of Newton methods on neural network loss functions.

2.5 Inexact Newton Methods

In Section 2.4, we solved the Newton system algebraically and exactly

$$0 = \nabla f(\theta) + \nabla^2 f(\theta)\, p^\star \tag{2.38}$$

But as noted in the beginning of this chapter, numerical methods are generically incapable of providing exact solutions. While we don't often think of this as applying to algebraic operations, we have already seen that it applies to square roots. In Section 2.5.1, we will see that it also applies to division, which is a special case of the matrix inversion we need to perform exact Newton methods.

We will then consider how we might replace the matrix inversion in our exact Newton algorithms, Algorithm 4 and Algorithm 5, with an approximate optimization, based on

[f]The *pre-image* of $y \in \mathcal{Y}$ with respect to a function $f \colon \mathcal{X} \to \mathcal{Y}$ is the set of all inputs that map to y: $\{x \in \mathcal{X} \colon f(x) = y\}$.

iterative solutions to linear systems of equations. We will focus on the Krylov subspace methods (Section 2.5.3).

2.5.1 Optimization Approach to Division

The Babylonian algorithm for computing the square root of a positive real number x, Algorithm 3, had the following iteration rule:

$$\theta_{t+1} = \theta_t/2 + x \div \theta_t/2 \tag{2.39}$$

For binary representations, division by 2 can be done by an $O(1)$ bit-shift, much like division by 10 in decimal representation. However, division by other numbers is, like the square root operation, not guaranteed to result in a finite binary string: $1/3$, for example[g]. Note that this is evident in the typical "long division" algorithm taught in elementary schools: children are often told to simply stop when they hit some number of decimal places.

To obtain an alternate division algorithm that's more explicitly an optimization, we first rewrite the iteration rule as

$$\theta_{t+1} = \theta_t/2 + \theta_t^{-1} \times x/2 \tag{2.40}$$

which makes the connection to the Newton algorithm more clear, because it is in terms of an inverse, but doesn't solve our problem, because the inverse is also an operation that takes us outside finite binary strings (e.g., 3^{-1}). Let us denote the inverse of θ_t as $\gamma^\star$. This pair satisfies

$$\gamma^\star = \underset{\gamma \in \mathbb{R}}{\operatorname{argmin}}\, \gamma^{-1} - \theta_t \tag{2.41}$$

to which we can apply Newton-Raphson by introducing the function

$$g(\gamma) = \gamma^{-1} - \theta_t \tag{2.42}$$

with gradient

$$\nabla g\left(\gamma\right) = -\gamma^{-2} \tag{2.43}$$

This hardly seems like an improvement, but when we plug it into the Newton iteration formula Equation 2.22 and rearrange, we obtain an update entirely in terms of multiplication and subtraction:

$$\begin{aligned}
\gamma_{k+1} &= \gamma_k - \frac{\gamma_k^{-1} - \theta_t}{-\gamma_k^{-2}} && (2.44)\\
&= \gamma_k + \gamma_k^2 \times \left(\gamma_k^{-1} - \theta_t\right) && (2.45)\\
&= \gamma_k + \gamma_k - \gamma_k^2 \times \theta_t && (2.46)\\
&= \gamma_k \times \left(2 - \gamma_k \times \theta_t\right) && (2.47)
\end{aligned}$$

[g]We could commit to exact arithmetic with rational numbers, but the sizes of our number representations would grow with each operation, which is infeasible for a neural network loss defined on thousands of examples.

which results in the approximate inverse algorithm below:

Algorithm 6: Newton Inversion

Require $K \in \mathbb{N}$, $(\theta, \gamma_0) \in (\mathbb{R} \setminus \{0\})^2$
$k \leftarrow 0$
while $k < K$ **do**
  $\gamma_{k+1} \leftarrow \gamma_k \times (2 - \gamma_k \times \theta)$
  $k \leftarrow k + 1$
end

Our full square-root finding algorithm, then, looks like this: in an outer loop, update our estimate for the square root by applying Newton-Raphson to the current estimate, as in Algorithm 3. The division operation requires the computation of the inverse, which occurs in an inner loop according to Algorithm 6. Note that this inverse is approximate as well.

2.5.2 Least-Squares Inexact Newton

The motivation for inexact Newton methods is precisely the same as for the optimization approach to division: we cannot exactly perform the inversion, so we search for a sensible way to do it approximately.

We proceed slightly differently to Section 2.5.1. We first note that an exact solution to the Newton system, $p^\star$, also satisfies

$$p^\star = \underset{p}{\operatorname{argmin}} \|\nabla f(\theta) + \nabla^2 f(\theta)\, p\|^2 \tag{2.48}$$

where the expression inside the squared norm is the right-hand-side of the Newton system. This is another example of relaxation to an optimization problem, as described in Section 2.2. We can therefore re-write our exact Newton algorithms in terms of this argmin operation.

Algorithm 7: Least-Squares Exact Newton

Require $T \in \mathbb{N}$, $\theta_0 \in \mathbb{R}^N$, $\nabla f \colon \mathbb{R}^N \to \mathbb{R}^N$, $\nabla^2 f \colon \mathbb{R}^N \to \mathbb{R}^{N \times N}$
$t \leftarrow 0$
while $t < T$ **do**
  $g \leftarrow \nabla f(\theta_t)$
  $H \leftarrow \nabla^2 f(\theta_t)$
  $p^\star \leftarrow \operatorname{argmin}_p \|Hp + g\|^2$
  $\theta_{t+1} \leftarrow \theta_t + p^\star$
  $t \leftarrow t + 1$
end

Our two exact Newton methods above were different choices of solution to the problem of computing the argmin in this meta-algorithm, appropriate for different assumptions about $\nabla^2 f$. If we relax the requirement to compute the exact argmin, which cannot be satisfied in a numerical setting anyway, we obtain a new meta-algorithm, which we call

least-squares inexact Newton.

Algorithm 8: Least-Squares Inexact Newton

Require $T \in \mathbb{N}$, $\theta_0 \in \mathbb{R}^N$, $\nabla f \colon \mathbb{R}^N \to \mathbb{R}^N$, $\nabla^2 f \colon \mathbb{R}^N \to \mathbb{R}^{N \times N}$, lsq-solve

$t \leftarrow 0$

while $t < T$ **do**

  $p \leftarrow \text{lsq-solve}\,(\theta_t, \nabla^2 f, \nabla f)$

  $\theta_{t+1} \leftarrow \theta_t + p$

  $t \leftarrow t + 1$

end

Here, lsq-solve is a function that takes in the current parameter value, Hessian function, and gradient function, and applies an approximate method to minimize the least-square value of the right-hand-side of the Newton system of equations. This setup has several advantages over the exact setup.

First, it allows for direct control over numerical issues. Iterative methods for computing least-squares solutions generally come with hyperparameters for stopping before a solution within numerical tolerance is obtained. For example, when a matrix M is non-invertible, its numerical representation is typically invertible, because its eigenvalues are non-zero, but very small. The numerically-inverted matrix will have large eigenvalues and the result of applying it to some vectors will be very large. An iterative solver can detect that its solution has grown above a threshold and terminate. This is just one of many ways that iterative solvers can better handle numerical issues.

Second, it transfers immediately to the case where the Hessian matrix is non-invertible. There are two issues in this case. Firstly, the argmin in Algorithm 7 need no longer be a single value, but can instead be a set (indeed, a linear subspace!). This occurs because any portion of p in the kernel of H can take on any value. A least-squares solver will always return only a single value from this set, typically according to some criterion. The exact solution computed by pseudo-inverse Newton, Algorithm 5, corresponds to choosing the element of the argmin with minimum norm, as in the algorithm below. Secondly, the updates in the exact case were taken to be solutions of the Newton system, which need not exist in the non-invertible case. When g is in the co-image of H, no vector p satisfies the Newton system. Such a system is called *unsatisfiable*, and we will have more to say about this case in Chapter 3. When the Newton system is unsatisfiable the updates are not solutions, but merely elements of the argmin in Equation 2.48.

Algorithm 9: Least-Squares Exact Newton for Incompatible Systems

Require $T \in \mathbb{N}$, $\theta_0 \in \mathbb{R}^N$, $\nabla f \colon \mathbb{R}^N \to \mathbb{R}^N$, $\nabla^2 f \colon \mathbb{R}^N \to \mathbb{R}^{N \times N}$
$t \leftarrow 0$
while $t < T$ **do**
  $g \leftarrow \nabla f(\theta_t)$
  $H \leftarrow \nabla^2 f(\theta_t)$
  $P \leftarrow \operatorname{argmin}_p \|Hp + g\|^2$
  $p^\star \leftarrow \operatorname{argmin}_{p \in P} \|p\|^2$
  $\theta_{t+1} \leftarrow \theta_t + p^\star$
  $t \leftarrow t + 1$
end

2.5.3 Krylov Subspace Methods for Least-Squares Inexact Newton

We now turn to the choice of method for lsq-solve. This discussion closely follows that in [71].

In principle, the least-squares sub-problem could be solved by generic convex optimization methods. However, this is unwise, because it is a very special type of convex problem: it is a *linear* system of equations. Linearity is a powerful property that enables specialized algorithms to achieve very high performance. These algorithms are called *linear solvers*.

The classic choice for linear solver is the *conjugate gradient* algorithm. On each iteration, conjugate gradient produces a step that is *conjugate* to all previous steps with respect to the current Hessian H, as defined below. See [60, Chapter 5] for details on this algorithm.

Definition 2.5: Conjugate Vectors

Two vectors x and y are *conjugate* with respect to the symmetric matrix M if

$$x^\top M y = 0 \tag{2.49}$$

By the symmetry of M, conjugacy is a symmetric relation.

Conjugate gradient has several nice properties as a linear solver for inexact Newton. With exact arithmetic, it produces an exact solution in no more than N steps, where the dimension of the system is $N \times N$. This is to be contrasted with generic optimization approaches, like gradient descent, which need never produce exact solutions, even with exact arithmetic. Furthermore, conjugate gradient can be implemented entirely in terms of matrix-vector products, with no need to form any $N \times N$ matrices besides the linear system's matrix of coefficients, which reduces memory footprint. This is particularly advantageous when the matrix is a Hessian, because Hessian-vector products can also be

computed in $O(N)$ time, versus $O(N^2)$ time for generic matrix-vector products, and don't even require the explicit formation of the $N \times N$ Hessian [65].

These properties stem from the fact that it is a *Krylov subspace method*, meaning that it computes its solution in a sequence of special subspaces of increasing dimension, up to N, defined below.

Definition 2.6: Krylov Subspace Methods

Let M be a matrix in $\mathbb{R}^{N\times N}$ and x be a vector in $\mathbb{R}^N$. The *Krylov subspace* of order r for M and x, denoted $\mathcal{K}_r(M, x)$, is defined as

$$\mathcal{K}_r(M, x) \stackrel{\text{def}}{=} \text{span}\left\{x, Mx, M^2x, \dots M^{r-1}x\right\} \tag{2.50}$$

An optimization method is a *Krylov subspace method* if its iterate at step r, p_r, is an element of the Krylov subspace of order r.

Note that the Krylov subspaces are constructed iteratively, with each new vector produced by means of matrix-vector multiplication. For more on Krylov subspaces, see [60, Section 5.1].

The choice of conjugate gradient is motivated by the fact that, for a smooth, convex function $f(\theta)\colon \Theta \to \mathbb{R}$, the kth iterate of conjugate gradient, p_k, applied to the Newton system at a point θ_t satisfies

$$p_k = \underset{p\in\mathcal{K}_k(\nabla^2 f(\theta_t),\nabla f(\theta_t))}{\text{argmin}} \nabla f(\theta_t)^\top p + \frac{1}{2}p^\top \nabla^2 f(\theta_t)\, p \tag{2.51}$$

$$= \underset{p\in\mathcal{K}_k(\nabla^2 f(\theta_t),\nabla f(\theta_t))}{\text{argmin}} \widehat{f}_{\theta_t}(p) \tag{2.52}$$

where $\widehat{f}_{\theta_t}$ is the second-order Taylor expansion of f at θ_t. That is, conjugate gradient produces the point within each Krylov subspace that minimizes the quadratic approxmation of the function f at θ_t.

This is ideal for the minimization of convex functions, but our goal is different: we wish to minimize the gradient. We should therefore like, instead, that our iterates minimize the norm of our approximation of the gradient, $r(p) = \nabla f(x) + \nabla^2 f(x)\, p$, within each Krylov subspace. The quantity r is known as the *residual*, and so we call a Newton method that uses a linear solver with that property a *minimum residual Newton method*, as defined below. Note that we explicitly consider the possibility that the Newton system is unsatisfiable in these algorithms and select the final update $p^\star$ according to some criterion function c. Furthermore, for simplicity's sake, we write that the algorithm selects the optimal iterate in a Krylov subspace of a fixed order, $\mathcal{K}_K$, but practical algorithms would iteratively determine the Krylov subspace based on numerical considerations, e.g. norm of the residual or the solution.

Algorithm 10: Minimum Residual Newton for Incompatible Systems

Require $T \in \mathbb{N}$, $K \in \mathbb{N}$, $\theta_0 \in \mathbb{R}^N$, $\nabla f \colon \mathbb{R}^N \to \mathbb{R}^N$, $\nabla^2 f \colon \mathbb{R}^N \to \mathbb{R}^{N \times N}$, $c \colon \mathbb{R}^N \to \mathbb{R}$

$t \leftarrow 0$

while $t < T$ **do**

  $g \leftarrow \nabla f(\theta_t)$

  $H \leftarrow \nabla^2 f(\theta_t)$

  $\mathcal{K} \leftarrow \mathcal{K}_K(H, g)$

  $P \leftarrow \operatorname{argmin}_{p \in \mathcal{K}} \|g + Hp\|^2$

  $p^\star \leftarrow \operatorname{argmin}_{p \in P} c(p)$

  $\theta_{t+1} \leftarrow \theta_t + p^\star$

  $t \leftarrow t + 1$

end

There are two major linear solvers that have the minimum residual property[h]: MINRES [63], short for "minimum residual", and MINRES-QLP [19], also known as MR-QLP, which is similar to MINRES but uses the QLP decomposition [38] in place of the QR decomposition[i]. In addition to MR-QLP having superior performance on systems that are ill-conditioned outside of their kernel, it also corresponds to the choice of $c(p) = \|p\|^2$ in Algorithm 10 above. The exact minimum norm solution is the one computed by the pseudo-inverse method, Algorithm 9. Therefore, an inexact Newton method with MR-QLP as its linear solver is a natural relaxation of an exact pseudo-inverse Newton method.

The choice of MR-QLP as a linear solver for inexact Newton methods was proposed in [71], which noted all of the favorable numerical and analytical properties described above. The final Newton algorithm proposed in [71] is, however, outside the class of algorithms considered so far. We next turn to the additional tricks required to define Newton algorithms used in practice.

2.6 Practical Newton Methods

The definition of the Newton step in the basic Newton-Raphson algorithm in Algorithm 4 was motivated by its optimality for functions with constant Hessian. For functions with non-constant Hessian, i.e. those with non-zero third or higher partial derivatives, this step is no longer optimal. Indeed, Newton's method can diverge, for example when used as a root finding method on the function $f(\theta) = \theta^{1/3}$. It can oscillate, for example when applied to find the critical points of $f(\theta) = \frac{x^4}{4} - x^2 + 2x$. It can even behave chaotically [34]. We encourage the reader to apply a few iterations of Algorithm 4 to these functions by hand, using 1 for θ_0.

[h] SYMMLQ, the symmetric LQ method, can also be used as a generic linear solver for compatible systems, but does not satisfy the minimum residual property [63].

[i] Conjugate gradient methods use the Cholesky decomposition, which is only defined for positive semi-definite matrices.

The two most common methods for avoiding these issues are damping (Section 2.6.1) and back-tracking line search (Section 2.6.2)[j]. The former takes the approach of generating multiple Newton-type steps and selecting the best one. The latter uses a standard Newton step to choose a direction and then performs a one-dimensional optimization to select a step size. In both cases, candidate updates are selected according to a scalar-valued *merit function*. For critical point-finding, the appropriate merit function is based on a norm of the gradient, here the squared Euclidean norm. For optimization, the appropriate merit function is the loss.

With these additions, we will have finally defined Newton methods sufficiently robust to handle the case of linear neural networks, as we will see in the first part of Chapter 3.

2.6.1 Damped Newton

The motivation for damped Newton methods comes from the case of convex functions with positive semi-definite but not positive definite Hessians. In that case, the Hessian can be made positive definite by adding an arbitrarily small perturbation γ to the diagional. However, this results in an inverse matrix with arbitarily large maximal eigenvalue γ^{-1}. Depending on how much of the gradient lies in the eigenspace corresponding to this eigenvalue, this could still result in a very large step. Since the Newton step is motivated by a local analysis, this is undesirable. Therefore, we consider a finite number of possible perturbations[k], $\Gamma = \{\gamma_i : \gamma_i \in \mathbb{R}_{\geqslant 0}\}$ and produce a Newton step for each one, with the Hessian matrix H replaced by $H + I\gamma_i$ for each i. Here, I is the identity matrix of the same shape as H.

We provide an example damped Newton method below, in Algorithm 11, based on the exact minimum residual Newton method, Algorithm 10. Damping can be combined with any method for computing the Newton iteration and is compatible with fast Hessian-vector products.

Damping methods are, however, not particularly well-suited to Hessians that are indefinite, with both positive and negative eigenvalues. Indeed, they can cause the matrix to become singular, even if it is initially non-singular. The eigenvalues of the matrix $M + \gamma I$ are equal to the eigenvalues of M plus γ, so any eigenvalue equal to $-\gamma$ becomes 0. More broadly, the goal of damping in the positive semi-definite case was to reduce the maximum step size by controlling the maximum eigenvalue of the inverse matrix. In the indefinite case, this is no longer achieved. This method was, however, used in [22] as a critical point-finding method on neural network loss surfaces, so we will examine its performance in Chapter 3.

[j]There are also trust-region methods. See [60, Chapter 6 and Section 11.2] for more on practical Newton methods.

[k]Denoting the set of all finite subsets of a set S as $\mathcal{P}_\omega(S)$, the type of Γ is written $\Gamma \in \mathcal{P}_\omega(\mathbb{R}_{\geqslant 0})$

Algorithm 11: Minimum Residual Damped Newton Method

Require $T \in \mathbb{N}$, $K \in \mathbb{N}$, $\theta_0 \in \mathbb{R}^N$, $\nabla f \colon \mathbb{R}^N \to \mathbb{R}^N$, $\nabla^2 f \colon \mathbb{R}^N \to \mathbb{R}^{N \times N}$, $\Gamma \in \mathcal{P}_\omega(\mathbb{R}_{\geqslant 0})$

$t \leftarrow 0$

while $t < T$ **do**

  $g \leftarrow \nabla f(\theta_t)$

  $P^\star \leftarrow \{\}$

  for $\gamma \in \Gamma$ **do**

    $H \leftarrow \nabla^2 f(\theta_t) + I\gamma$

    $\mathcal{K} \leftarrow \mathcal{K}_K(H, g)$

    $P \leftarrow \operatorname{argmin}_{p \in \mathcal{K}} \|g + Hp\|^2$

    $P^\star \leftarrow P^\star \cup \operatorname{argmin}_{p \in P} \|p\|^2$

  end

  $p^\star \leftarrow \operatorname{argmin}_{p \in P^\star} \|\nabla f(\theta_t + p)\|^2$

  $\theta_{t+1} \leftarrow \theta_t + p^\star$

  $t \leftarrow t + 1$

end

2.6.2 Guarded Newton

Unlike typical first-order optimization algorithms, the Newton methods presented thus far have had no step size hyperparameter. These Newton methods are sometimes called *pure* Newton methods. Effectively, the step size has been fixed at 1. This is not entirely artificial. First, the Newton step is invariant to affine transformations of the parameterization of the function, and so there is a meaning to unit step size that is absent in first-order methods [12, Section 9.5]. Second, and more critically, the rapid local convergence of Newton methods requires a unit step size and collapses if non-unit step sizes are chosen [60, Theorems 11.2, 11.7, & 11.10]. However, when outside the region of local convergence, the pure Newton step is frequently too long.

One solution is to select the step size adaptively, so that it can be 1 inside the region of rapid convergence and less than 1 outside it. Such methods are sometimes called *guarded Newton methods*: the method is "guarded" against taking steps that are too large. In particular, one common approach to selecting the step size is *back-tracking line search* [5], abbreviated BTLS.

In a line search, the step size $\alpha^\star$ for a direction p at a point θ is cast as the solution to an optimization problem:

$$\alpha^\star \stackrel{\text{def}}{=} \operatorname*{argmin}_{\alpha \in \mathbb{R}_{\geqslant 0}} c(\theta + \alpha p) \tag{2.53}$$

for some *merit function*, c. When selecting a step size for a typical optimization problem, the natural choice is $c(x) = f(x)$, where f is the loss. For critical point-finding, the natural choice is $c(x) = \|\nabla f(x)\|^2$. Note that damped Newton methods also involve a choice of merit function, to select between the potential iterates produced by different

values of γ.

Of course, this problem cannot typically be solved exactly. In back-tracking line search, the step size is selected by starting with a large value $0 < \alpha \leqslant 1$ and then decreasing it multiplicatively by a factor $0 < \beta < 1$ on each failed iteration. A failed iteration k is one where the step size choice $\alpha \times \beta^k$ produces insufficient decrease in the merit function c. The degree of decrease considered sufficient is typically an increasing function of the step size and controlled by another "tolerance" hyperparameter $0 < \rho < 1$. The authors of [71] determined the appropriate criterion for decrease in the case where c is the squared gradient norm by considering the case of a function with constant Hessian. Effectively, a step size is chosen when the squared gradient norm decreases by an amount within a factor of ρ of how much it would decrease in that case.

It is instructive to consider a concrete example. The behavior of pure Newton and guarded Newton on the function $f(\theta) = \|\theta\|^{3/2}$ are contrasted in Figure 2.2. Starting at the point $\theta_0 = -1$, (gold circle) the pure Newton iteration $\theta_1 + p^\star$ (blue star) is 1, which results in oscillation: $\theta_2 = 1, \theta_3 = -1 = \theta_0$. Back-tracking line search avoids this pathology. Only step sizes that result in values of θ that decrease the squared gradient by a sufficient amount (below dashed gray line in Figure 2.2B) are acceptable. These values of θ are indicated in dark red. Back-tracking line search starts at the pure Newton step and decreases its length by a multiplicative factor (black ticks) until it lands inside the red region. The result is the next iterate, θ_1 (gold star).

We present a back-tracking line search appropriate for selecting Newton step sizes in Algorithm 12. We make a few modifications to the basic scheme described above. First, we allow the starting value of α to be non-unit, but enforce that, before the multiplicative BTLS begins, the value of 1 is always checked first. Second, we allow that the criterion for convergence used in this check, ρ', is different from the value used in the BTLS, ρ. Finally, we allow the specification of a minimum step size, ε, below which α is set to 0. We set this to the largest value for which $\alpha = \beta\alpha$.

2.6.2.1 Newton-MR

The combination of MR-QLP for inexact Newton step selection and back-tracking line search for step size selection is known as *Newton-MR* [71]. This will prove to be the most effective algorithm for critical point-finding in Chapter 3, so we provide it in pseudocode here (Algorithm 13). In this pseudocode, MR-QLP is the minimum residual solver based on the QLP decomposition from [19] described in Section 2.5.3 and BTLS is the back-tracking line search method Algorithm 12.

We briefly review the hyperparameters of MR-QLP here. `maxit` sets the maximum number of iterations. In exact arithmetic, this would never need be greater than N. We found that solutions didn't tend to increase in quality for values above N. `maxxnorm` sets the largest acceptable value for the norm of p while `acondlim` sets the largest acceptable value for the estimated condition number of the Hessian restricted to the subspace in which the iterations lie. We found that MR-QLP did not terminate due to violating these constraints for reasonable values, and so we set them to 1e4 and 1e7, respectively.

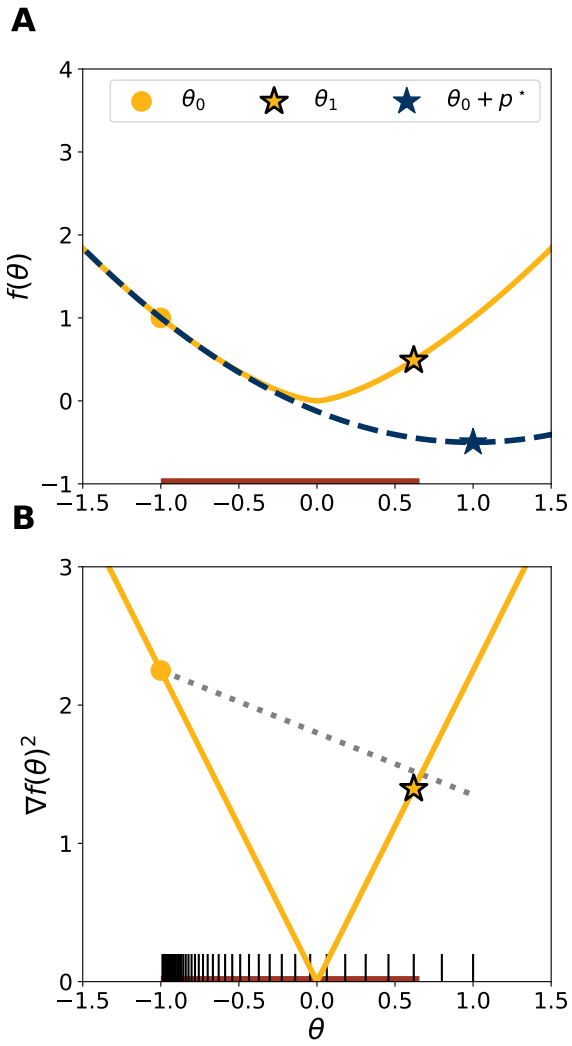


Figure 2.2: **Back-Tracking Line Search for Guarded Newton**
A: $f(\theta) = \|\theta\|^{3/2}$ is plotted as a solid gold line, and its second-order Taylor approximation from the point -1 as a dashed blue line. Values of θ that result in sufficient decrease of the squared gradient with the tolerance parameter $\rho = 0.2$ indicated in dark red. The point selected by BTLS with $\beta = 0.9$ indicated with a gold star. **B**: The squared gradient of f is plotted as a solid gold line. The sufficient decrease criterion that appears in the line search convergence check in Algorithm 12 is plotted as a dashed gray line. Values of θ that result in squared gradients below this line in dark red. Tick marks indicate points possibly visited by the back-tracking line search. Hyperparameters as in A.

Algorithm 12: Backtracking Line Search for Guarded Newton

Require $\nabla f\colon \mathbb{R}^N \to \mathbb{R}^N$, $\nabla^2 f\colon \mathbb{R}^N \to \mathbb{R}^{N\times N}$, $p \in \mathbb{R}^N$, $\theta \in \mathbb{R}^N$, $\alpha \in (0,1]$, $\beta \in (0,1)$, $\rho \in (0,1)$, $\rho' \in (0,1)$, $\varepsilon \in \mathbb{R}_{\geqslant 0}$

$H \leftarrow \nabla^2 f(\theta)$

Function `CheckConvergence`(α, ρ)**:**

  $\theta' \leftarrow \theta + \alpha p$

  $s, s' \leftarrow \|\nabla f(\theta)\|^2, \|\nabla f(\theta')\|^2$

  $\Delta \leftarrow 2\rho\alpha \cdot p^\top H \nabla f(\theta)$

  return $s' \leqslant s + \Delta$

converged $\leftarrow$ `CheckConvergence` $(1, \rho')$

if converged **then**

  $\alpha \leftarrow 1$

  return α

end

while not converged **do**

  converged $\leftarrow$ `CheckConvergence` (α, ρ)

  if not converged **then**

    $\alpha \leftarrow \beta\alpha$

    if $\alpha < \varepsilon$ **then**

      $\alpha \leftarrow 0$

      return α

    end

  end

end

return α

The critical hyperparameter for stopping behavior was `rtol`, short for relative tolerance. When either the residual or the residual projected onto the Hessian co-kernel is less than `rtol`, following suitable normalization, the algorithm terminates. Details on the definitions of these residuals are deferred to Chapter 3. Not listed above is the `trancond` hyperparameter, which determines when the algorithm transitions between using the QR and QLP decompositions, based on an estimate of the Hessian's condition number. This hyperparameter was also not critical for determining stopping behavior of MR-QLP. We set it to 1e4.

2.7 Conclusion

In this chapter, we introduced two major classes of critical point-finding algorithms: gradient norm minimization methods, which apply first-order optimization algorithms to the

Algorithm 13: Newton-MR

Require $\theta_0 \in \mathbb{R}^N$, $f\colon \mathbb{R}^N \to \mathbb{R}$, $\nabla f\colon \mathbb{R}^N \to \mathbb{R}^N$, $\nabla^2 f\colon \mathbb{R}^N \to \mathbb{R}^{N\times N}$, $T \in \mathbb{N}$,
`maxit` $\in \mathbb{N}$, `maxxnorm` $\in \mathbb{R}$, `rtol` $\in \mathbb{R}$, `acondlim` $\in \mathbb{R}$, $\alpha \in (0, 1]$, $\beta \in (0, 1)$,
$\rho \in (0, 1)$, $\rho' \in (0, 1)$, $\varepsilon \in \mathbb{R}_{\geqslant 0}$

$t \leftarrow 0$
while $t < T$ **do**
  $g \leftarrow \nabla f(\theta_t)$
  $H \leftarrow \nabla^2 f(\theta_t)$
  $p \leftarrow$ MR-QLP $(H,\ -g,$ `maxit`, `maxxnorm`, `rtol`, `acondlim`$)$
  $\alpha \leftarrow \text{BTLS}(\nabla f, p, \theta_t, \alpha, \beta, \rho, \rho', \varepsilon)$
  $\theta_{t+1} \leftarrow \theta_t + \alpha p$
  if $\theta_{t+1} == \theta_t$ **then**
    break
  end
  $\alpha \leftarrow \max(1, \beta^{-1}\alpha)$
  $t \leftarrow t + 1$
end

squared gradient norm, and Newton methods, which use repeated solution of the Newton system of equations. Both classes of algorithms use the Hessian matrix of second partial derivatives, and so are second-order critical point-finding methods. In part because of the complexity of Newton methods and in part because of the relative lack of experience and expertise with these methods in the neural network community, the Newton methods formed the majority of this chapter. We approached their explanation pedagogically, based on an analogy to the calculation of the square root to high accuracy. We reviewed exact Newton methods, which use algebraic solutions of the Newton system, and inexact Newton methods, which use iterative methods to approximate those solutions. Finally, we developed two inexact Newton methods of sufficient robustness for practical use, damped Newton and Newton-MR.

In the next chapter, we will apply these algorithms to neural network loss functions. After a warm up on a linear neural network on which we can check our answers, we apply these methods to non-linear neural networks. In this case, they will generally fail to find critical points, which will motivate a re-analysis of the behavior second-order critical point-finding algorithms when the Hessian is singular.

Chapter 3

Applying Critical Point-Finding Algorithms to Neural Network Loss Functions Reveals Gradient-Flat Regions

3.1 Chapter Summary

Chapter 2 motivated and defined a number of second-order algorithms for finding the critical points of neural network loss functions. As described in Chapter 1, information about the local curvature at these points is useful for understanding the optimizability of neural networks. For example, the No-Bad-Local-Minima theory (NBLM; see Section 1.4), based on a random function model of neural network losses (Section 1.4.1, Section 1.4.2), predicts that critical points with strictly positive curvature should only occur when the value of the loss is low. This implies that first-order methods like gradient descent can optimize neural networks [42]. Previous work [22, 67] appeared to verify this theory. More recent analytical work, however, has indicated that the NBLM theory is false, and there are in fact bad local minima on neural network loss functions ([23]; see Section 1.4.3). This disagreement with theory motivates a closer look at the numerical evidence, which is the substance of this chapter.

First, in Section 3.2, we present a test problem, the deep linear autoencoder, that is analytically tractable while remaining sufficiently similar to the analytically intractable problem of interest, viz. non-linear networks. The analytically-derived critical points are used to verify that the approximate critical points recovered by the numerical algorithms are accurate. Then, we apply the best-performing of these methods, Newton-MR, to a non-linear network and observe a tremendous change in its behavior: qualitative signatures of convergence disappear and quantitative metrics sharply decline (Section 3.3).

We return, then, to the analysis of second-order critical point-finding methods and demonstrate that a large class of spurious targets of convergence has been over-looked: points where the gradient lies in the kernel of the Hessian. We call these *gradient-flat*

points. They are defined in Section 3.4. We present evidence that these points are encountered by critical point-finding methods applied to neural network losses in Section 3.5.

The gradient-flat points that cause the most trouble are bad local minima of the gradient norm. The core result of this chapter and this dissertation, then, is that the second-order methods used to find the critical points of neural network loss functions in an attempt to prove the no-bad-local-minima theory suffer from a bad-local-minima problem of their own.

3.2 The Deep Linear Autoencoder Provides a Useful Test Problem

This section introduces the deep linear autoencoder as a test problem for critical point-finding methods for neural network loss functions. First, the need for test problems is explained (Section 3.2.1). Then, the deep linear autoencoder is presented and the critical points derived Section 3.2.2. Finally, the performance of Newton-MR (Algorithm 13), damped Newton (Algorithm 11), and gradient norm minimization by gradient descent with back-tracking line search (BTLS-GNM) are compared (Section 3.2.3).

Additional results regarding these critical point-finding methods and the deep linear autoencoder are published in [30].

3.2.1 Test Problems are Necessary

In the case of optimization for the solution of engineering problems, the question of convergence to the neighborhood of a true minimizer is often an academic one. The goal of the optimization procedure is to find a point at which the loss is sufficiently low to e.g. allow the airplane to fly or select the better ad to display to users. The problem of finding critical points is different: our goal is to check analytical properties, which can in principle require arbitrary precision.

The usual solution is to prove an upper bound on the precision required by means of inequalities. In our case, we are interested in two quantities: the loss L and the index I (Definition 1.6) at critical points θ_{cp}. For the loss, we might proceed by demonstrating that the function is K-Lipschitz for some constant K.

Definition 3.1: Lipschitz Continuity

A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is said to be *K-Lipschitz* if

$$\|f(x+\varepsilon) - f(x)\| \leqslant K\|\varepsilon\| \tag{3.1}$$

for some $K \in \mathbb{R}_{\geqslant 0}$ and for all $x, \varepsilon \in \mathbb{R}^n$.

For an almost everywhere differentiable function, like ReLU, K-Lipschitz continuity is equivalent to demonstrating that the gradient norm is bounded by K. Using this fact, a

bound on the difference between the loss at an approximate critical point with gradient norm ε and any nearby true critical points can be derived.

However, the Lipschitz constants of typical neural network losses are not well-controlled [33], resulting in worst-case bounds that are overly-pessimistic. The situation is even worse for calculating the index, which depends both on Lipschitz bounds on the operator norm of the Hessian and on guarantees that the kernel changes only slowly.

In the absence of these guarantees, the results of numerical algorithms must be interpreted with care and hyperparameters must be tuned cautiously. The behavior of algorithms during convergence should be closely monitored and reported along with results. And critically, it is important to test numerical algorithms on problems for which the answers are known before applying them to problems for which the answers are in doubt. In addition to allowing for hyperparameters to be set to reasonable starting values and for the signatures of convergence to be identified, test problems provide an important opportunity to debug implementations that is missing in cases without ground truth. Analytical guarantees are of little value if the algorithms in question are broken.

Furthermore, there are many critical point-finding algorithms that might be chosen. Chapter 2 presented two Newton methods of sufficient robustness for practical use, plus gradient norm minimization. Both the damped Newton method [22] and gradient norm minimization [67] have been used, but sufficient information for comparing performance was not published.

It is critical that the test problem be designed to be as close as possible to the problem of interest. The following section introduces such a test problem for neural network loss functions.

3.2.2 The Deep Linear Autoencoder has Known Critical Points

The key difficulty for deriving analytical expressions for the critical points of non-linear neural networks is that the gradients of the loss are non-linear in an arbitrary fashion. This turns the problem of finding critical points into the problem of finding solutions to generic non-linear equations.

Neural networks without non-linearities, also known as *linear networks*, avoid this problem. The network is constructed by raveling the entries of θ into a D-element sequence of matrices $W_i(\theta)$ which multiply an input x in turn. We call these matrices the *weight matrices* of the network and say that the network has D layers, each with *layer size* equal to the number of rows in the matrix W_i. The layers W_1 through W_{D-1} are called *hidden layers*.

$$\mathrm{NN}_{\mathrm{DLN}}(\theta)(x) = W_D(\theta)W_{D-1}(\theta)\dots W_2(\theta)W_1(\theta)x \tag{3.2}$$

$$\stackrel{\mathrm{def}}{=} W(\theta)x \tag{3.3}$$

As the second line indicates, this is equivalent to applying a single matrix $W(\theta)$. Therefore the function computed by the network as a whole is linear. Any non-linearity in the gradient function of the loss of such a network is introduced only by the cost function and regularizer. For simplicity, we consider only the unregularized case.

The loss of this network can be construed as a function of any of three quantities. As before, it is a function of the vector of parameters θ, which we continute to denote L. It is also a function of the separate weights of each layer, the matrices W_i, which we denote l and refer to as the *layerwise* loss. These functions also exist for non-linear networks. Finally, in the linear case the loss can also be written a function of the "collapsed" matrix W, which we denote $\mathcal{L}$ and refer to as the *collapsed* loss. We further drop the explicit dependence of W and the W_i on θ when writing the loss in this way.

With this setup, we can write the gradients of a linear network of arbitrary depth with arbitrary cost in simple form.

Theorem 3.1: Gradients of a Deep Linear Network

Let l be the layerwise loss and $\mathcal{L}$ the collapsed loss of a deep linear network with D layers. The gradient function of the layerwise loss with respect to layer k, denoted $\nabla_{W_k}(l)$, is

$$\nabla_{W_k} l\left(W_1, \ldots, W_D\right) = W_{k+1:}^\top \nabla \mathcal{L}\left(W\right) W_{:k}^\top \tag{3.4}$$

where the notation $W_{i:j}$, inspired by the slicing syntax of Python and other languages, stands for the products of matrices i to $j-1$, with an empty entry on the left standing for 1 and an empty entry on the right standing for $D+1$.

Proof of Theorem 3.1:

This proof follows closely that in [45]. The gradient is defined in terms of the value of the function at an input perturbed by ε

$$l(W_1, \ldots, W_{k-1}, W_k + \varepsilon, W_{k+1}, \ldots W_D) \tag{3.5}$$

where here ε is a matrix of the same shape as W_k. We proceed by converting to the collasped loss $\mathcal{L}$ and multiplying through the matrix product.

$$\begin{aligned} l(W_1, \ldots, W_{k-1}, W_k + \varepsilon, W_{k+1}, \ldots W_D) &= \mathcal{L}\left(W_D \ldots W_{k+1}(W_k + \varepsilon)W_{k-1} \ldots W_1\right) && (3.6)\\ &= \mathcal{L}\left(W + W_{k+1:}\varepsilon W_{:k}\right) && (3.7)\\ &= \nabla \mathcal{L}\left(W\right) + \left\langle \nabla \mathcal{L}\left(W\right), W_{k+1:}\varepsilon W_{:k} \right\rangle + o(\varepsilon) && (3.8) \end{aligned}$$

where the last line follows by pattern-matching to the definition of the gradient function, Definition 1.2. Note that the inner product here is an inner product of matrices. It is the Frobenius inner product

$$\left\langle A, B \right\rangle = \operatorname{tr}\left(A^\top B\right) \tag{3.9}$$

which is defined by unraveling the matrices into vectors and applying the Euclidean inner product of vectors, i.e. as a pullback of that inner product via a raveling isomorphism. The trace tr is invariant to cyclic permutations, and so we can re-organize the middle term of Equation 3.8

$$\begin{aligned} \left\langle \nabla \mathcal{L}\left(W\right), W_{k+1:}\varepsilon W_{:k} \right\rangle &= \operatorname{tr}\left(\nabla \mathcal{L}\left(W\right)^\top W_{k+1:}\varepsilon W_{:k}\right) && (3.10)\\ &= \operatorname{tr}\left(W_{:k} \nabla \mathcal{L}\left(W\right)^\top W_{k+1:}\varepsilon\right) && (3.11)\\ &= \left\langle W_{k+1:}^\top \nabla \mathcal{L}\left(W\right) W_{:k}^\top, \varepsilon \right\rangle && (3.12) \end{aligned}$$

which implies, by pattern-matching to the definition of the gradient function again,

$$\nabla_{W_k} l\left(W_1, \ldots, W_D\right) = W_{k+1:}^{\top} \nabla \mathcal{L}\left(W\right) W_{:k}^{\top} \tag{3.13}$$

■

This suggests that if we wish to be able to come up with an analytic expression for the critical points, which make this gradient function 0, we choose a collapsed loss function $\mathcal{L}$ that has a simple form that allows us to set Equation 3.4 equal to 0 and solve.

We therefore make several simplifying choices in constructing our test problem. First, we take the inputs and targets to be the same. This type of network is known as an *autoencoder*. Second, we choose the squared error as the loss function. Finally, we choose the number of layers D to be 2. We refer to this combination as the *deep linear autoencoder*.

The critical points of this network can be characterized as follows:

Theorem 3.2: Critical Points of Deep Linear Autoencoder

Let L be the loss function of a linear network such that

$$L(\theta) = \|X - W_2(\theta)W_1(\theta)X\|^2 \tag{3.14}$$

for some matrix $X \in \mathbb{R}^{k \times n}$ such that $XX^\top$ is full rank, with $W_1 \colon \mathbb{R}^{2kp} \to \mathbb{R}^{p \times k}$ $W_2 \colon \mathbb{R}^{2kp} \to \mathbb{R}^{k \times p}$.

Then the critical points of this network correspond to those θ_{cp} such that the matrix $W = W_2(\theta_{\text{cp}})W_1(\theta_{\text{cp}})$ acts as the identity on a subspace spanned by some subset of the eigenvectors of $XX^\top$ and the zero matrix otherwise.

Before diving into the proof, we first consider the interpretation and implications. The matrix $XX^\top$ is the sample covariance matrix of the data X when the data has sample mean 0. The eigenvectors of this matrix are known as the *principal components* of the data. A single critical point can be constructed according to the criterion in the theorem above by choosing $m \leqslant p$ separate principal components for the rows of W_1 (setting the others to 0 when $m < p$) and choosing $W_2 = W_1^\top$. This allows the construction of a number of distinct critical points equal to the number of ways to choose from 0 to p elements from a k element set: $\sum_{i=0}^{p} \binom{k}{i}$. When the eigenvectors selected are the m with largest eigenvalue, this network is performing *principal components analysis*. This value corresponds to the global minimum. It can be demonstrated that there are no non-global local minima [8, 45], and so this loss function has the no-bad-local-minima property. It is a generalization of Example 1.1.

An uncountable collection of additional critical points can be constructed from the points corresponding directly to the principal components. Indeed, the key property in Theorem 3.2 is defined in terms of the collapsed matrix W. Applying any invertible $p \times p$ matrix C after W_1 and its inverse C^{-1} before W_2 leaves W unchanged[a], and so

[a]Note that invertible matrices form a Lie group, and therefore so do these critical points. This situation is shared, to an extent, in ReLU networks [28].

the resulting new point is also a critical point. The loss and index are unchanged, and so we can consider these collections to be equivalence classes of critical points, with representatives given by the choice $C = C^{-1} = I$.

Theorem 3.2, which was first proven in [8], is proven in the following section for completeness. Readers uninterested in the technical details are invited to skip to Section 3.2.3.

3.2.2.1 Proof of Theorem 3.2

The collapsed loss function of the linear network is

$$\mathcal{L}(W) = \|X - WX\|^2 \tag{3.15}$$

Expanding the right-hand side, we have that

$$\mathcal{L}(W+\varepsilon) = \operatorname{tr}\left(X^\top X\right) - \operatorname{tr}\left(2X^\top (W+\varepsilon)X\right) + \operatorname{tr}\left(X^\top (W+\varepsilon)^\top (W+\varepsilon)X\right) \tag{3.16}$$

$$= \mathcal{L}(W) + \operatorname{tr}\left(2XX^\top W\varepsilon - 2XX^\top \varepsilon\right) + o(\varepsilon) \tag{3.17}$$

which, after re-arrangement, gives the gradient function for $\mathcal{L}$ by pattern-matching to Definition 1.2:

$$\nabla\mathcal{L}(W) = 2XX^\top (W - I) \tag{3.18}$$

To find the analytical critical points, we plug this definition into the expressions for the gradients with respect to the two weight matrices W_1 and W_2 given by Theorem 3.1 and solve for 0.

We start with W_2:

$$\nabla l_{W_2}(W_1, W_2) = \nabla\mathcal{L}(W)\, W_1^\top \tag{3.19}$$

$$0 = \nabla\mathcal{L}(W)\, W_1^\top \tag{3.20}$$

$$0 = 2XX^\top (W_2W_1 - I)\, W_1^\top \tag{3.21}$$

$$W_1^\top = W_2W_1W_1^\top \tag{3.22}$$

where the transition to Equation 3.22 used the invertibility of $XX^\top$. This equation is satisfied whenever W_2W_1 is equivalent to the identity matrix when restricted to the co-kernel of W_1 (i.e. when W_2W_1 is a projection matrix onto that subspace), which is the range of $W_1^\top$.

This isn't sufficient to completely determine W_1 and W_2, so we proceed to the equations for W_1:

$$0 = W_2^\top \nabla\mathcal{L}(W) \tag{3.23}$$

$$0 = W_2^\top 2XX^\top (W_2W_1 - I) \tag{3.24}$$

$$W_2^\top = W_2^\top XX^\top W_2W_1\left(XX^\top\right)^{-1} \tag{3.25}$$

When the matrix $XX^\top$ is simultaneously diagonalizable with the matrix W_2W_1, the matrices commute, which gives the equation

$$W_2^\top = W_2^\top W_2W_1 \tag{3.26}$$

This equation holds when W_2W_1 is equivalent to the identity when applied to the co-kernel of $W_2^\top$.

For W_2W_1 to be simultaneously diagonalizable with $XX^\top$, the non-zero eigenvectors of W_2W_1 need to be the same as some subset of the eigenvectors of $XX^\top$. Together, these conditions imply that at a critical point of the loss, W_2W_1 acts as the identity on a subspace spanned by some subset of the eigenvectors of $XX^\top$.

3.2.3 Newton-MR Outperforms Previous Methods on this Problem

We are now ready to set up the deep linear autoencoder test problem and compare the performance of our critical point-finding algorithms.

3.2.3.1 Data and Network

Theorem 3.2 gives a characterization of the set of all critical points, Θ_{cp}, for a given choice of dataset $X \in \mathbb{R}^{k\times n}$ and two-layer network. The key parameters for the dataset are its covariance matrix, dimension k, and size n. The key architectural hyperparameter of the network is the size of the hidden layer (p, as in Theorem 3.2).

For simplicity, we choose a multivariate Gaussian distribution for the columns of X, as this distribution is entirely specified by its mean and covariance. The connection to PCA required that $XX^\top$ be the sample covariance matrix of X, and so we enforce that X has row means exactly 0. Note that this must be done after sampling, as setting the distributional means to 0 does not result in sample row means of 0. We choose a diagonal covariance matrix. To obtain maximally-spaced eigenvalues while still controlling the condition number of $XX^\top$, we space the diagonal values linearly between 1 and k, inclusive. We choose n, the number of samples, to be 1e4.

The parameters k, for the dimension of the data, and p, for the dimension of the hidden layer of the neural network, directly determine the number of critical points, up to equivalence. Since this count is effectively given by the number of ways to choose subsets from a set of size k, it grows nearly as fast as 2^k. For this reason, we choose k to be small, relative to the typical inputs to neural networks: 16, as opposed to order 100 or 1000. Selecting p also to be small, specifically 4, gives a total number of equivalence classes of critical points equal to $2517 = \sum_{i=0}^{4} \binom{16}{i}$. Below, we will call the set of all representatives of the equivalence classes the *analytical critical points* of the deep linear autoencoder.

With all of these choices in place, we can now calculate the loss and index values of the analytical critical points for a neural network loss function. These are plotted in Figure 3.1. The squared gradient norms of these points are generally not 0, but on the order of 1e-32. Already, this provides a loose lower bound on the precision we need in order to recover critical points. There is one notable exception: the 0 vector, which is a critical point for this loss. Here, the gradient is exactly 0. In general, the density of floating point numbers is greatest near 0, and so critical points near 0 can be recovered with greater accuracy. Note that, in a numerical setting, the index is defined as the number of eigenvalues below some small negative value, rather than 0, to account for

error. We choose -1e-5, based on inspection of the spectra of these analytical critical points.

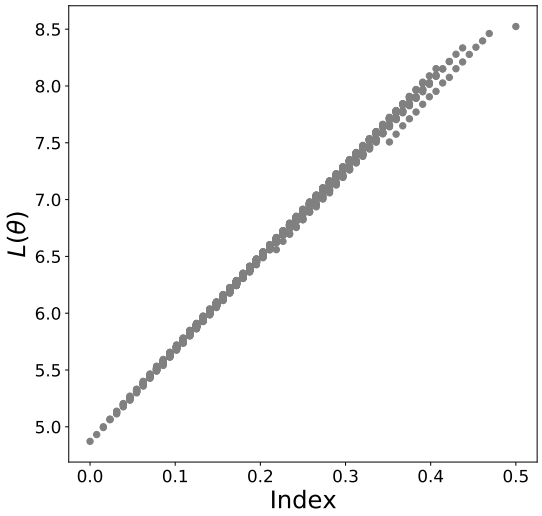


Figure 3.1: **The Analytical Critical Points of a Deep Linear Autoencoder**.

3.2.3.2 Two-Step Process for Sampling Critical Points

The full set of critical points, Θ_{cp}, is uncountably infinite in size, as is the full set of ε-critical points, $\Theta_{\varepsilon\text{-cp}}$. Even the set of critical points up to equivalence is exponentially large. For a problem as small as this one, it might be feasible to find all of them, but for larger networks this won't be the case. Instead, our methods must aim to sample the set of critical points. If the sample is unbiased, qualitative and quantitative features of the loss-index relationship (Figure 3.1) should be preserved.

To obtain initial points for our critical point-finding methods, we first apply an optimization algorithm to the loss. This generates a trajectory of values of θ with varying loss, from near the value of the highest critical point to near the value of the global minimum. These values are then chosen as initial points for critical point-finding algorithms. We follow [67] in selecting these points uniformly at random based on their loss value. Due to the exponential rate of first-order optimization algorithms, selecting points uniformly at random from the trajectory would over-represent lower loss values. Furthermore, decreasing gradient norms during training mean that later parameter values are closer together, and so this scheme would heavily over-sample a small region. Both factors should reduce the variety in recovered critical points[b]. To half of the initial points attained in this manner, we add Gaussian noise of with variance 1e-2 (an SNR of approximately 2.8 dB). In

[b]These approaches were directly compared in experiments not presented here and this intuition is borne out. See [30].

separate experiments, noise of this magnitude this was found to increase the diversity in critical points recovered [30].

3.2.3.3 Critical Point-Finding Methods Have Differential Performance

When applied to the problem of finding the critical points of the deep linear autoencoder problem outlined above, the three critical point finding methods, Newton-MR, damped Newton, and gradient norm minimization by gradient descent with backtracking line search (BTLS-GNM), have wildly different performance, even though all three are capable of finding numerical critical points with the same loss and index as the analytical critical points.

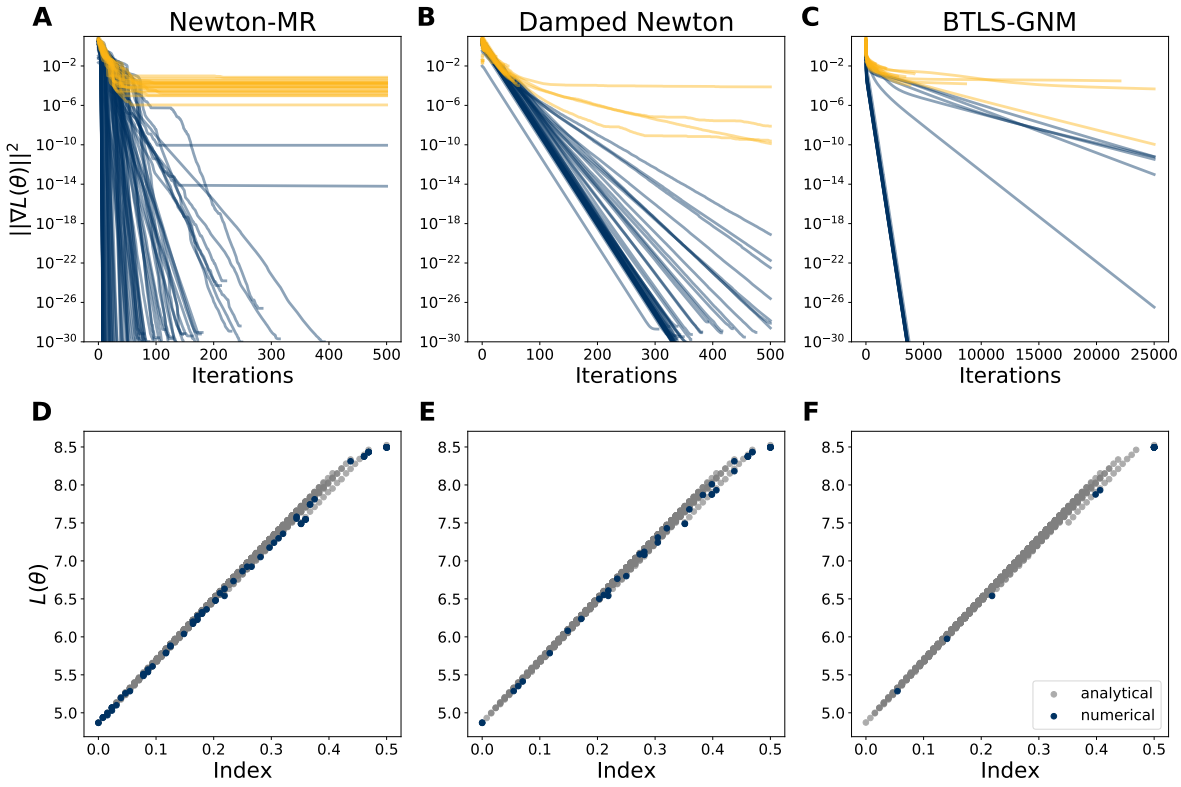


Figure 3.2: **Newton-MR, Damped Newton, and BTLS-Gradient Norm Minimization can Recover Critical Points of a Deep Linear Autoencoder**.
A-C. Squared gradient norms at each iteration of the three critical point-finding algorithms. Runs that terminate with squared gradient norm below 1e-10 in blue, otherwise in gold. The y-axis is truncated at 1e-30, since only runs that terminate at or near $\theta = 0$ reach substantially below this value. Horizontal and vertical axes are shared across the top row and the bottom row of panels separately.

Results for all three algorithms appear in Figure 3.2. The first row of panels, A-C, presents squared gradient norms for each algorithm across iterations. Results for 100 runs

of Newton-MR appear in panel A, for 100 runs of damped Newton in panel B, and for 60 runs of BTLS-GNM in panel C. Runs are terminated either due to hitting the maximum number of iterations or due to hitting the minimum step size of the backtracking line search without finding an acceptable update. Note that BTLS-GNM is allowed 25000 iterations before termination, while the Newton methods are only allowed 500. This algorithm only requires a single Hessian-vector multiplication outside of the backtracking line search phase, which is much less than the maximum of $O(n)$ multiplications required by the inexact Newton methods. Traces are plotted transparently to allow a rough estimation of density. Runs that terminate with squared gradient norm above 1e-10 are in gold, others in blue. Note that a large number of damped Newton and BTLS-GNM runs terminate in a small number of iterations and with squared gradient norm above 1e-10.

Notably, a number of runs of all three algorithms, but especially BTLS-GNM, obtain much lower values of the squared gradient norm. These are runs that terminate at or near the zero vector, which is a critical point for this problem. Floating point numbers are much denser near the origin, and so more accurate computations, and so lower gradient norms, are possible. The loss of this critical point is approximately 8.5 and the index 0.5.

Runs that terminated with squared gradient norm above 1e-10 were considered failures. This filter value was determined by comparing loss and index values at candidate critical points to the loss and index at analytical critical points, as in Figure 3.2D-F (only successful runs shown). A value of 1e-10 was sufficient to obtain the close match there displayed, while a value of 1e-8 or higher was not. Note the greater number and diversity of critical points recovered by Newton-MR (Figure 3.2D) compared to the damped Newton method (Figure 3.2E) and especially to BTLS-GNM (Figure 3.2F; but note that results from BTLS-GNM are based on 60 runs, versus 100 for the Newton methods).

The relative performance of the three algorithms is summarized in Table 3.1. Newton-MR was clearly superior in terms of the fraction of runs that ended in successes (80% versus 45% for damped Newton and 35% for BTLS-GNM) and in terms of the elapsed walltime per success (20 min versus 1hr 45 minutes for damped Newton and 35% for BTLS-GNM).

Algorithm	**Pseudocode**	**Fraction Successful**	**Time per Success**
Newton-MR	Algorithm 13	80% ± 4	20 min
Damped Newton	Algorithm 11	45% ± 5	1hr 45 min
BTLS-GNM	Algorithm 2[c]	35% ± 6	1hr 15 min

Table 3.1: **Newton-MR Outperforms Damped Newton and BTLS-GNM**. The fraction of successful runs is given as a percentage with a standard error based on a Gaussian approximation to the sampling distribution of the percentage. Newton-MR and Newton-TR results based on 100 runs; BTLS-GNM results based on 60 runs. Runtimes are based on walltime on commodity hardware and are to be regarded as highly approximate.

[c]The indicated pseudocode algorithm is for gradient norm minimization by gradient descent. These

3.2.3.4 Cutoffs Must be Set Strictly

As noted in Section 3.2.1, one of the key purposes of applying our algorithms to a test problem is to obtain estimates for the value of ε necessary to guarantee that the loss and index values for members of $\Theta^L_{\varepsilon\text{-cp}}$ are similar to those of Θ^L_{cp}. With such a value in hand, we can terminate our critical point-finding algorithms early, as soon as they produce a candidate critical point with squared gradient norm below ε.

We simulate the results of such a procedure by applying a "cutoff" to the traces from Figure 3.2A. We truncate each run at the first T such that $\|\nabla L(\theta_T)\|^2 < \varepsilon$. As our cutoffs, we choose the value used to filter the traces in Figure 3.2, 1e-10, the value used as a filter in [67], 1e-6, a looser criterion of 1e-4, and ∞. The latter corresponds to no cutoff, taking the initial points, the iterates of the optimization algorithm, as candidate critical points. The results are in Figure 3.3.

Notably, using the value of 1e-10 that, as a filter, resulted in accurate recovery of loss and index values (Figure 3.2), introduces a small amount of error, in particular at low and high values of the index (Figure 3.3A). This suggests that a strategy of running critical point-finding methods until termination and then filtering gives better results than a strategy of early termination. Looser cutoff values (Figure 3.3B,C) result in quantitatively worse recovery of loss-index values but qualitatively similar loss-index relationships. Interestingly, applying no cutoff at all, as in Figure 3.3D, results in a convex upwards relationship between index and loss, much like that reported in [22] and [67] for non-linear networks, despite the fact that the true relationship is linear.

3.3 Methods that Work on a Linear Network Fail on a Non-Linear Network

Unfortunately, success on this test problem does not guarantee success on the original problem of interest. When applied to a nonlinear network, even with the same data, these methods fail.

Figure 3.4 shows results for Newton-MR applied to a network with two hidden layers of sizes 8 and 16 and Swish non-linearity [70], but the same data as in Section 3.2.3. Results for other algorithms are qualitatively similar. The squared gradient norms do not drop nearly as low (greater than 1e-18, versus 1e-30 in the linear case, Figure 3.2A), even though the iteration budget is doubled. All runs exhibit a "plateau", unlike the successful runs in Figure 3.2A, which exhibited linear or faster convergence to a point at which no further progess could be made. A handful of experiments with much larger iteration counts indicated that this plateau persists even as the computational budget increases (data not shown). It is unclear whether the loss and index values (Figure 3.4B) recovered in this case are accurate. The results here are just a single example; in Section 3.5, we will see that these phenomena recur on many neural network losses.

results are for gradient norm minimization by gradient descent with backtracking line search, which adds a line search step akin to Algorithm 12, but with a convergence criterion based on the Wolfe conditions [78] applied to the squared gradient norm.

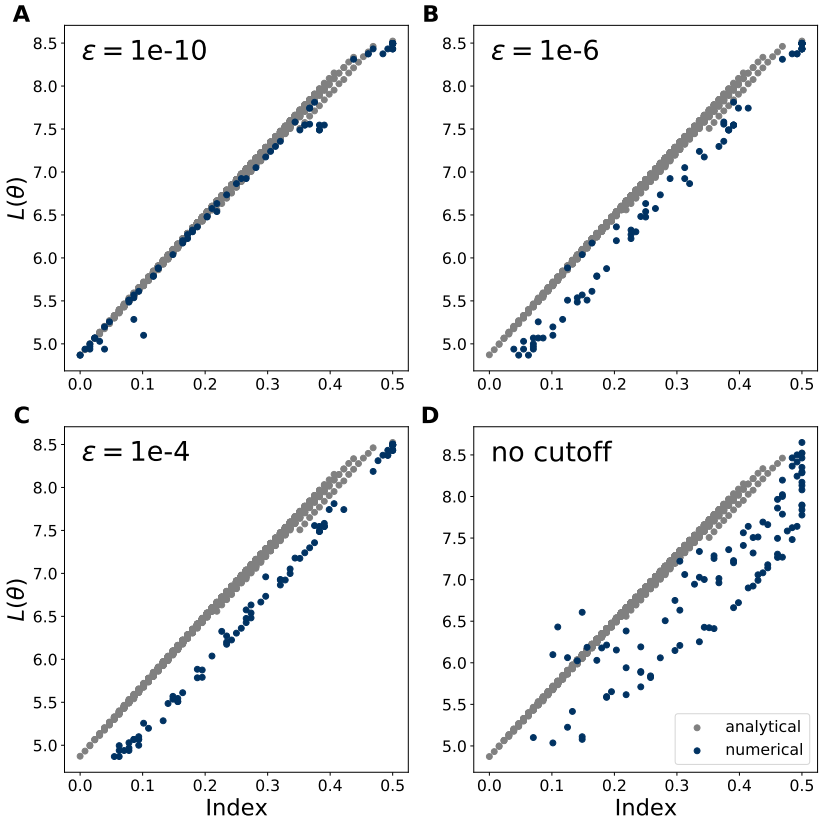


Figure 3.3: **Cutoffs Above 1e-10 are Insufficient to Guarantee Accurate Loss and Index Recovery**.
As in Figure 3.2, ε-CPs are plotted in blue over analytical CPs in gray. For each panel, ε-CPs are selected by taking the 100 runs of Newton-MR in Figure 3.2A and taking the first point whose squared gradient norm is below the cutoff value, ε, in the top-left corner. Horizontal and vertical axes are shared between panels.

This motivates a re-analysis of the critical point-finding methods, in particular in the singular case. We will see, in the following section, that the plateau behavior in Figure 3.4 can be recreated on a very simple test problem, so long as that problem has a point where the gradient lies in the Hessian's kernel: a *gradient-flat* point. This analysis, and the experiments in Section 3.5 that it motivates, are also presented in [29].

3.4 Gradient-Flat Regions can Cause Critical Point-Finding Methods to Fail

In this section, we introduce and define gradient-flat points and explain why they are problematic for second-order critical point-finding methods (Section 3.4.1), with the help of a low-dimensional example to build intuition (Section 3.4.2). In numerical settings and

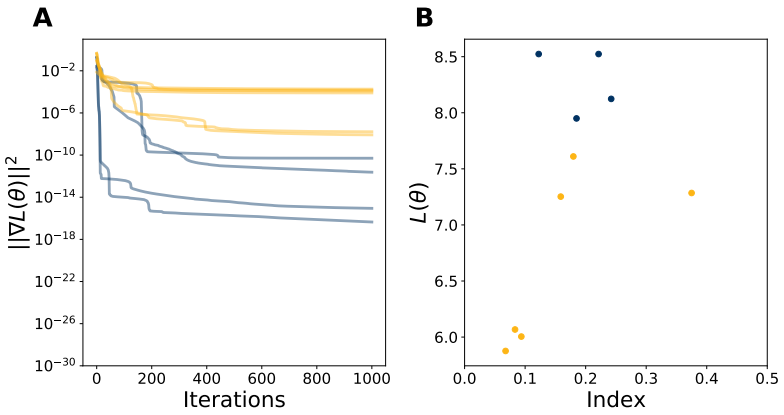


Figure 3.4: **Newton-MR Fails to Find Critical Points on a Non-Linear Network**. **A**. Squared gradient norms across iteration for ten runs of Newton-MR applied to a non-linear network. **B**. Loss and index values at 1000 iterations for the runs of Newton-MR in *A*. Reproduced, with permission, from [29].

in high dimensions, approximately gradient-flat points are also important, and so we define a quantitative index of gradient-flatness based on the residual norm of the Newton update (Section 3.4.3). Connected sets of these numerically gradient-flat points are gradient-flat regions, which cause trouble for second-order critical point-finding methods.

3.4.1 At Gradient-Flat Points, the Gradient Lies in the Hessian's Kernel

Critical points are of interest because they are points where the first-order approximation of a function f at a point $x + \delta$ based on the local information at x

$$f(x + \delta) \approx f(x) + \nabla f\left(x\right)^{\top} \delta \tag{3.27}$$

is constant, indicating that they are the stationary points of first-order optimization algorithms like gradient descent and its variants.

We can similarly understand the behavior of our second-order critical point-finding methods by considering their stationary points. In our construction of these methods in Chapter 2, we used a linear approximation of the behavior of the gradient function at a point $x + p$ given the local information at a point x

$$\nabla f\left(x + p\right) \approx \nabla f\left(x\right) + \nabla^2 f\left(x\right) p \tag{3.28}$$

Solving for the value of p that makes the left-hand side 0 gave us the Newton update

$$\nabla^2 f\left(x\right)^{+} \nabla f\left(x\right) \tag{3.29}$$

which is only 0, for a non-singular Hessian, if $\nabla f\left(x\right)$ is 0. For a singular Hessian, the update p is zero iff $\nabla f\left(x\right)$ is in the kernel of the pseudoinverse.

In gradient norm minimization, we search for critical points by applying first-order optimization methods to the squared gradient norm function, $g(\theta) = \|\nabla L(\theta)\|^2$. These methods also make a linear approximation of the gradient function of L. The gradient function of g is

$$\nabla g(x) = \nabla^2 f(x) \nabla f(x) \tag{3.30}$$

As with Newton methods, in the invertible case the updates are zero iff $\nabla f(x)$ is 0. In the singular case, the updates are zero if the gradient is in the Hessian's kernel.

In the case of invertible Hessians, then, these methods can guarantee convergence to critical points. However, neural network Hessians are generally singular, especially in the overparameterized case [72, 32]. In this case, neither class of methods can guarantee convergence to critical points (for GNM, see [24]; for Newton methods, see [69, 34]).

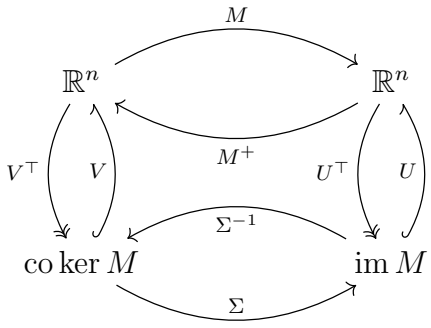
What are the stationary points, besides critical points, for these two method classes in the case of singular Hessians? It would seem at first that they are different: for gradient norm minimization, when the gradient is in the Hessian's kernel; for Newton-type methods, when the gradient is in the Hessian's pseudoinverse's kernel. In fact, however, these conditions are identical, due to the Hessian's symmetry, and so both algorithms share a broad class of stationary points. A proof, based on the construction of the pseudo-inverse in Section 2.4.3.1, follows.

Theorem 3.3: Kernel Equals Pseudo-Inverse Kernel for Symmetric M

Let $M \in \mathbb{R}^{n \times n}$ be a symmetric matrix. Then

$$\ker M = \ker M^+ \tag{3.31}$$

Proof of Theorem 3.3:
We first repeat the commutative diagram relating the SVDs of a matrix and its pseudo-inverse, specialized to a square matrix.

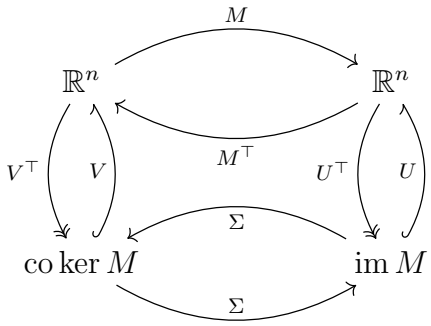


Note that $\operatorname{im} M$ is also $\operatorname{co}\ker M^+$, by reading the diagram counter-clockwise. The SVD of the transpose of M can be attained by transposing each of the elements of the SVD:

$$M^\top = \left(U \Sigma V^\top\right)^\top \tag{3.32}$$

$$= V \Sigma U^\top \tag{3.33}$$

This can be summarized in the commutative diagram below.



This implies that $\operatorname{im} M = \operatorname{co}\ker M^\top$. But for a symmetric matrix, $M = M^\top$, and so $\operatorname{co}\ker M = \operatorname{co}\ker M^\top = \operatorname{im} M$. But above, we saw that $\operatorname{im} M = \operatorname{co}\ker M^+$, and therefore $\operatorname{co}\ker M = \operatorname{co}\ker M^+$, which further implies that $\ker M = \ker M^+$.

■

These stationary points have been identified previously, but nomenclature is not standard: Doye and Wales, studying gradient norm minimization, call them *non-stationary points* [24], since they are non-stationary with respect to the function f, while Byrd et al., studying Newton methods, call them *stationary points* [17], since they are stationary with respect to the merit function g. To avoid confusion between these incommensurate conventions or with the stationary points of the function f, we introduce our own terminology.

Definition 3.2: Gradient-Flat Points

A point $\theta \in \mathbb{R}^n$ is a *gradient-flat* point of a twice continuously-differentiable function $f\colon \mathbb{R}^n \to \mathbb{R}$ if

$$\nabla f(\theta) \in \ker \nabla^2 f(\theta) \tag{3.34}$$

This name was chosen because a function is *flat* when its Hessian is 0, meaning every direction is in the kernel, and so it is locally flat around a point in a given direction whenever that direction is in the kernel of the Hessian at that point. Note that, because $0 \in \ker$ for all matrices, every critical point is also a gradient-flat point, but the reverse is not true. When we wish to explicitly refer to gradient-flat points which are not critical points, we will call them *strict* gradient-flat points. At a strict gradient-flat point, the function is, along the direction of the gradient, locally linear up to second order.

There is an alternative view of gradient-flat points based on the squared gradient norm function g. All gradient-flat points are stationary points of the gradient norm: they are zeroes for ∇g. They may in principle be local minima, maxima, or saddles, while the global minima of the gradient norm are critical points. When they are local minima of the gradient norm, they can be targets of convergence for methods that use first-order

approximations of the gradient map, as in gradient norm minimization and in Newton-type methods. Strict gradient-flat points, then, can be bad local minima of the gradient norm, and therefore prevent the convergence of second-order root-finding methods to critical points, just as bad local minima of the loss function can prevent convergence of first-order optimization methods to global optima.

Note that Newton methods cannot be demonstrated to converge only to gradient-flat points [69, 17]. Furthermore, Newton convergence can be substantially slowed when even a small fraction of the gradient is in the kernel [34]. Below we will see that, while Newton-MR applied to a neural network loss sometimes converges to and almost always encounters strict gradient-flat points, the final iterate is not always either a strict gradient-flat point or a critical point.

3.4.2 Convergence to Gradient-Flat Points in a Quartic Example

The difficulties that gradient-flat points pose for Newton methods can be demonstrated with a polynomial example in two dimensions, plotted in Figure 3.5A. Below, we will characterize the strict gradient-flat (gold) and critical (blue) points of this function (Figure 3.5A). Then, we will observe the behavior of Newton-MR when applied to it (Figure 3.5B) and note similarities to the results in Figure 3.4. We will use this simple, low-dimensional example to demonstrate principles useful for understanding the results of applying second-order critical point-finding methods to more complex, higher-dimensional neural network losses.

Example 3.1: A Quartic Polynomial with a Gradient-Flat Point

$$f(x, y) = 1/4x^4 - 3x^2 + 9x + 0.9y^4 + 5y^2 + 40 \tag{3.35}$$

Equation 3.35 is plotted in Figure 3.5A, central panel. This quartic function has two affine subspaces of points with non-trivial Hessian kernel, defined by $[\pm\sqrt{2}, y]$. The kernel points along the x direction and so is orthogonal to this affine subspace at every point. As a function of y, f is convex, with one-dimensional minimizers at $y = 0$. The strict gradient-flat points occur at the intersections of these two sets: one strict gradient-flat point at $[\sqrt{2}, 0]$, which is a local minimum of the gradient norm, and one at $[-\sqrt{2}, 0]$, which is a saddle of the same (Figure 3.5A, gold points, all panels). In the vicinity of these points, the gradient is, to first order, constant along the x-axis, and so the function is locally linear or flat. These points are gradient-flat but neither is a critical point of f. The only critical point is located at the minimum of the polynomial, at $[-3, 0]$ (Figure 3.5A, blue point, all panels), which is also a global minimum of the gradient norm. The affine subspace that passes through $[-\sqrt{2}, 0]$ divides the space into two basins of attraction, loosely defined, for second-order methods: one, with initial x-coordinate $x_0 < -\sqrt{2}$, for the critical point of f and the other for the strict gradient-flat point. Note that the vector field in the central panel shows update directions for the pure Newton method, which

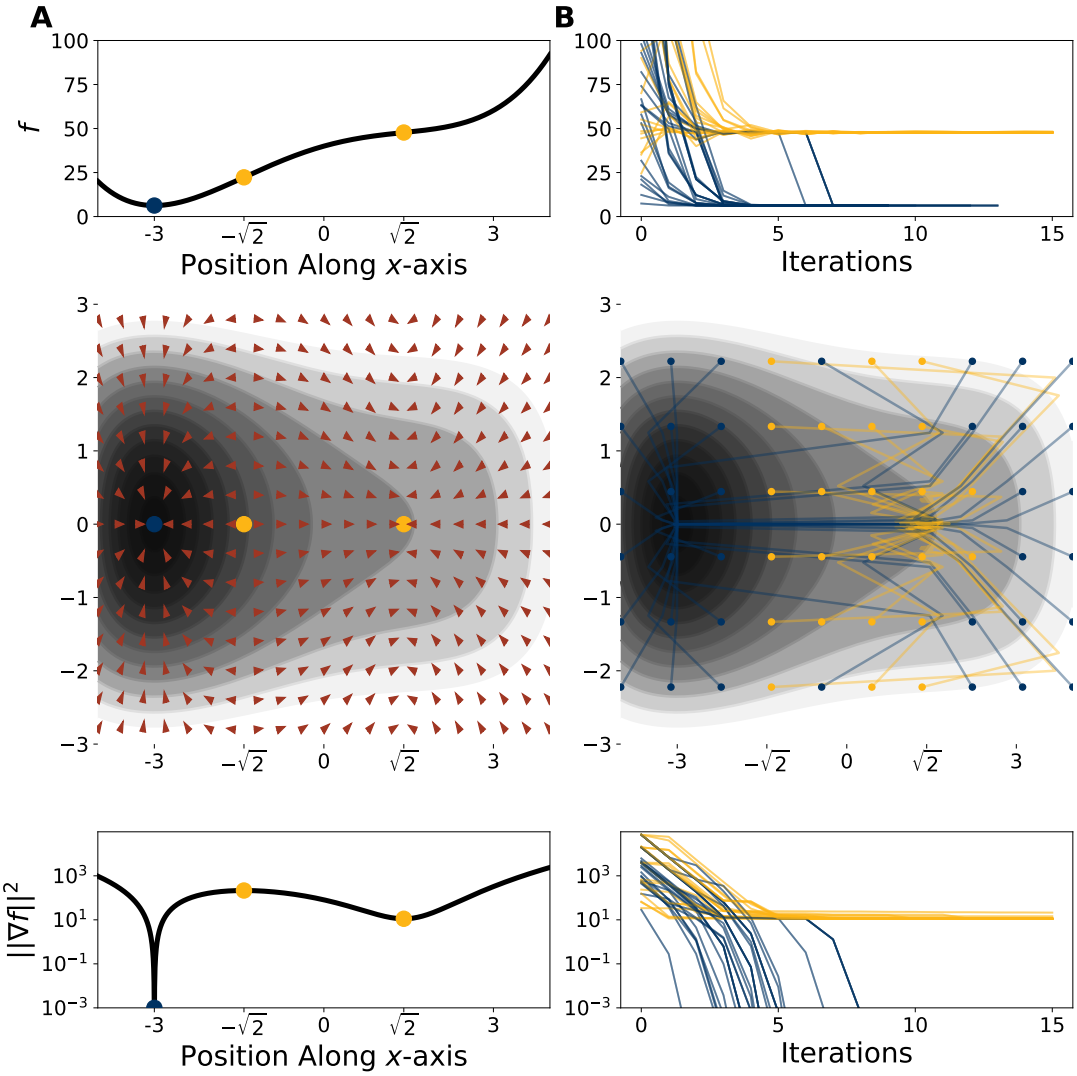


Figure 3.5: **Stationarity of and Convergence to a Strict Gradient-Flat Point on a Quartic Function**.
A: Critical and strict gradient-flat points of quartic $f(x, y)$ (defined in Equation 3.35). Central panel: $f(x, y)$ plotted in black and white (black, low values; white, high values), along with the direction of the Newton update p as a (notably non-smooth) vector field (red). Stationary points of the squared gradient norm merit function g are indicated: strict gradient-flat points in gold, the critical point in blue. Top and bottom panels: The value (top) and squared gradient norm (bottom) of f as a function of x value with y fixed at 0. The x axis is shared between panels. **B**: Performance and trajectories of Newton-MR on Equation 3.35. Runs that terminate near a strict gradient-flat point are in gold, while those that terminate near a critical point are in blue. Central panel: Trajectories of Newton-MR laid over $f(x, y)$. x and y axes are shared with the central panel of A. Initial values indicated with scatter points. Top and bottom panels: Function values (top) and squared gradient norms (bottom) of Newton-MR trajectories as a function of iteration. The x axis is shared between panels. Reproduced, with permission, from [29].

can behave extremely poorly in the vicinity of singularities [69, 34], often oscillating and converging very slowly or diverging.

Practical Newton methods, like those introduced in Chapter 2 and used in Section 3.2.3 above, use techniques like damping and line search to improve behavior. To demonstrate how a practical Newton method behaves on this function, we focus on the case of Newton-MR. Results are qualitatively similar for damped Newton.

The results of applying Newton-MR to Equation 3.35 are shown in Figure 3.5B. The gradient-flat point is attracting for some trajectories (gold), while the critical point is attracting for others (blue). For trajectories that approach the strict gradient-flat point, the gradient norm does not converge to 0, but converges to a non-zero value near 10 (gold trajectories; Figure 3.5B, bottom panel). This value is typically several orders of magnitude lower than the initial point, and so would appear to be close to 0 on a linear scale that includes the gradient norm of the initial point. Since log-scaling of loss functions is uncommon in machine learning, as losses do not always have minima at 0, second-order methods apporaching gradient-flat points can appear to converge to critical points if typical methods for visually assessing convergence are used.

There are two interesting and atypical behaviors worth noting. First, the trajectories tend to oscillate in the vicinity of the gradient-flat point and converge more slowly (Figure 3.5B, central panel, gold lines). Updates from points close to the affine subspace where the Hessian has a kernel, and so which have an approximate kernel themselves, sometimes jump to points where the Hessian doesn't have an approximate kernel. This suggests that, when converging towards a gradient-flat point, the degree of flatness will change iteration by iteration. Second, some trajectories begin in the nominal basin of attraction of the gradient-flat point but converge to the critical point (Figure 3.5B, central panel, blue points with x-coordinate $> -\sqrt{2}$). This is because the combination of back-tracking line search and large proposed step sizes means that occasionally, very large steps can be taken, based on non-local features of the function. Indeed, back-tracking line search is a limited form of global optimization and the ability of line searches to change convergence behaviors predicted from local properties on nonconvex problems is known [60]. Since the back-tracking line search is based on the gradient norm, the basin of attraction for the true critical point, which has a lower gradient norm than the gradient-flat point, is much enlarged relative to that for the gradient-flat point. This suggests that Newton methods using the gradient norm merit function will be biased towards finding gradient-flat points that also have low gradient norm.

Finally, we note that the behavior of the failed runs in the bottom panel of Figure 3.5B matches that of the runs in Figure 3.4. In particular, after a brief period of rapid improvement, the squared gradient norm plateaus at relatively large value given by the value of the gradient norm at the gradient-flat point (in this case, 1e1).

3.4.3 Approximate Gradient-Flat Points Form Gradient-Flat Regions

Analytical arguments focus on exactly gradient-flat points, where the Hessian has an exact kernel and the gradient is entirely within it. In numerical settings, it is almost certain no matrix will have an exact kernel, due to rounding error. For the same reason, the computed gradient vector will generically not lie entirely within the exact or approximate kernel. However, numerical implementations of second-order methods will struggle even when there is no exact kernel or when the gradient is only partly in it, and so a numerical index of flatness is required. This is analogous to the requirement to specify a tolerance for the norm of the gradient when deciding whether to consider a point an approximate critical point or not.

We quantify the degree of gradient-flatness of a point by means of the *relative residual norm* (r) and the *relative co-kernel residual norm* (r_H) for the Newton update direction p. The vector p is an inexact solution to the Newton system $Hp + g = 0$, where H and g are the current iterate's Hessian and gradient. The residual is equal to $Hp + g$, and the smaller its norm, the better p is as a solution. The co-kernel residual is equal to the Hessian times the residual, and so ignores any component in the kernel of the Hessian. Its norm quantifies the quality of an inexact Newton solution in the case that the gradient lies partly in the Hessian kernel, the unsatisfiable case, where $Hp \neq -g$ for any p. When the residual is large but the co-kernel residual is small (norms near 1 and 0, respectively, following suitable normalization), then we are at a point where the gradient is almost entirely in the kernel of the Hessian: an approximate gradient-flat point. In the results below, we consider a point approximately gradient-flat when the value of r_H is below 5e-4 while the value of r is above 0.9. We emphasize that numerical issues for second-order methods can arise even when the degree of gradient-flatness is small.

The relative residual norm, r, measures the size of the error of an approximate solution to the Newton system:

$$r(p) = \frac{\|Hp + g\|}{\|H\|_F\|p\| + \|g\|} \tag{3.36}$$

where $\|M\|_F$ of a matrix M is its Frobenius norm. Since all quantities are non-negative, r is non-negative; because the denominator bounds the numerator, by the triangle inequality and the compatibility of the Frobenius and Euclidean norms, r is at most 1. For an exact solution of the Newton system $p^\star$, $r(p^\star)$ is 0, the minimum value, while $r(0)$ is 1, the maximum value. Note that small values of $\|p\|$ do not imply large values of this quantity, since $\|p\|$ goes to 0 when a Newton method converges towards a critical point, while r goes to 0.

When g is partially in the kernel of H, the Newton system is unsatisfiable, as g will also be partly in the co-image of H, the linear subspace into which H cannot map any vector. In this case, the minimal value for r will no longer be 0. The optimal solution for $\|Hp + g\|$ instead has the property that its residual is 0 once restricted to the co-kernel of H. This co-kernel residual can be measured by applying the matrix H to the residual

vector $Hp + g$. After normalization, it becomes

$$r_H(p) = \frac{\|H(Hp + g)\|}{\|H\|_F \|Hp + g\|} \tag{3.37}$$

Note that this value is also small when the gradient lies primarily along the eigenvalues of smallest magnitude. On each internal iteration, MR-QLP checks whether either of these values is below a tolerance level (the hyperparameter `rtol`; in our experiments, 5e-4) and if either is, it ceases iteration. With exact arithmetic, either one or the other of these values should go to 0 within a finite number of iterations; with inexact arithmetic, they should just become small. See [19] for details.

Under this relaxed definition of gradient-flatness, there will be a neighborhood of approximate gradient-flat points around a strict, exact gradient-flat point for functions with Lipschitz gradients and Hessians. Furthermore, there might be connected sets of non-null Lebesgue measure which all satisfy the approximate gradient-flatness condition but none of which satisfy the exact gradient-flatness condition. We call both of these *gradient-flat regions*.

There are multiple reasonable numerical indices of flatness besides the definition above. For example, the Hessian-gradient regularity condition in [71], which is used to prove convergence of Newton-MR, would suggest creating a basis for the approximate kernel of the Hessian and projecting the gradient onto it. Alternatively, one could compute the Rayleigh quotient of the gradient with respect to the Hessian. Our method has the advantage of being computed as part of the Newton-MR algorithm. It furthermore avoids diagonalizing the Hessian or the specification of an arbitrary eigenvalue cutoff. The Rayleigh quotient can be computed with only one Hessian-vector product, plus several vector-vector products, so it might be a superior choice for larger problems where computing a high-quality inexact Newton step is computationally infeasible.

3.5 Gradient-Flat Regions Abound on Several Neural Network Losses

To determine whether gradient-flat regions are responsible for the poor behavior of Newton methods on deep neural network (DNN) losses demonstrated in Figure 3.4, we applied Newton-MR to the losses of several neural networks. We focused on Newton-MR because we found that a damped Newton method like that in [22] performed poorly, as reported for the XOR problem in [20].

3.5.1 Gradient-Flat Regions on an Autoencoder Loss

We first present results for two networks trained on 10k MNIST [48] images downsized to 4×4, similar to the downsized datasets in [22, 67]. Images were cropped to 20×20 and rescaled to 4×4 using PyTorch [64], then z-scored. This was done to improve the condition of the data covariance matrix, which is very poor for MNIST due to the low variance in the

border pixels. It also reduced the size of the network. Non-linear classification networks trained on this down-sampled data could still obtain accuracies above 90%, better than the performance of logistic regression ($\approx 87\%$).

First, we consider a nonlinear autoencoder. This network had two hidden layers of 8 and 16 units, used Swish non-linearities. Unlike previous networks, this network had biases, meaning each layer performed an affine rather than a linear transformation. Gradient norms for the first 100 iterations of Newton-MR applied to this loss appear in Figure 3.6A. As in the non-linear autoencoder applied to the multivariate Gaussian data (Figure 3.4), we found that, after 500 iterations, all of the runs had squared gradient norms over 10 orders of magnitude greater than the typical values observed after convergence in the linear case (<1e-30, Figure 3.2A). 14% of runs terminated with squared gradient norm below the filtering value of 1e-10 and so found likely critical points (blue). Twice as many runs terminated above that cutoff but terminated in a gradient-flat region (28%, gold), while the remainder were above the cutoff but were not in a gradient-flat region at the final iteration (black).

The relative residual norm for the Newton solution, r, is an index of gradient-flatness; see Section 3.4.3 for details. The values of r for every iteration of Newton-MR are shown for three representative traces in Figure 3.6B. In the top trace, r is close to 0, indicating that the iterates are not in a gradient-flat region ($r \ll 0.9$, black). Newton methods can be substantially slowed when even a small fraction of the gradient is in the kernel [34] and can converge to points that are not gradient-flat [17]. By contrast, in the middle trace (gold), the value of r approaches 1, indicating that almost the entirety of the gradient is in the kernel. This run terminated in a gradient-flat region, at effectively an exactly gradient-flat point. Further, the squared gradient norm at 500 iterations, 2e-5, is five orders of magnitude higher than the cutoff, 1e-10. This is smaller than the minimum observed during optimization of this loss (squared gradient norms between 1e-4 and 5e1), indicating the presence of non-critical gradient-flat regions with very low gradient norm. Critical point-finding methods that disqualify points on the basis of their norm will both converge to and accept these points, even though they need not be near true critical points. In the bottom trace (blue), the behavior of r is the same, while the gradient norm drops much lower, to 3e-13, suggesting convergence to a gradient-flat region around a critical point that has an approximately singular Hessian.

We found that 99 of 100 traces included a point where at least half of the gradient was in the kernel, according to our residual measure, while 89% of traces included a point that had a residual greater than 0.9, and 50% included a point with $r > 0.99$ (Figure 3.6C, bottom). This demonstrates that there are many regions of substantive gradient-flatness, in which second-order critical point-finding methods could be substantively slowed.

The original purpose of applying these critical point-finding methods was to determine whether the no-bad-local-minima property held for this loss function, and more broadly to characterize the relationship at the critical points between the loss and the local curvature, summarized via the Morse index. If we look at either the iterates with the highest gradient-flatness (Figure 3.6D), we find that the qualitative features of the loss-index relationship reported in [22] and [67] are recreated: convex shape, small spread at low index that increases for higher index, no minima or near-minima at high values of the loss.

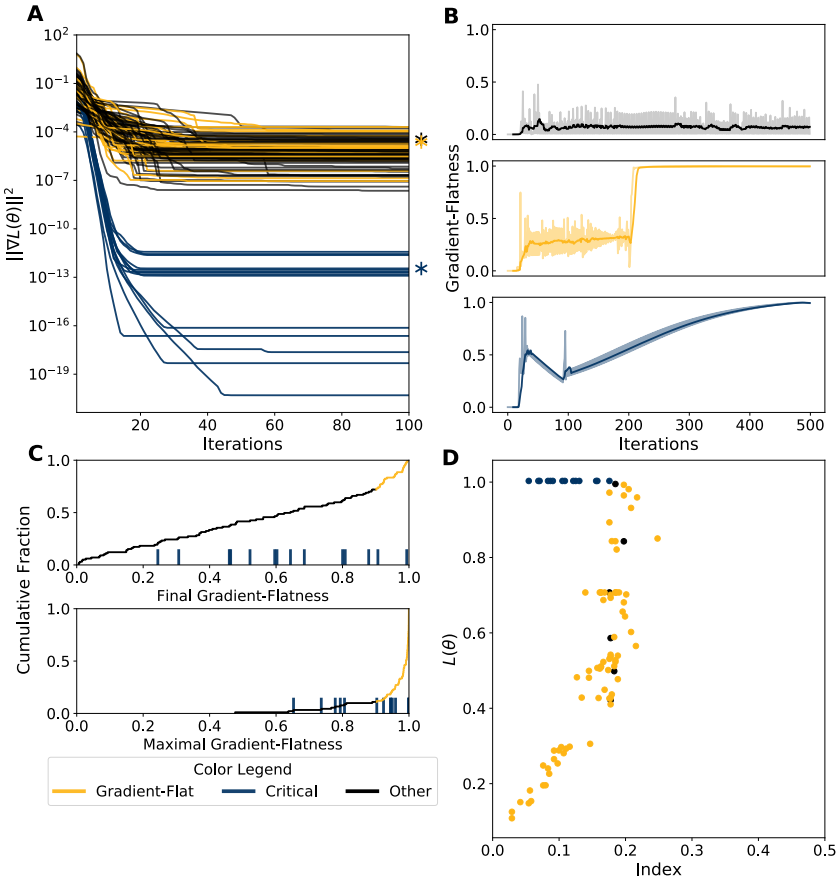


Figure 3.6: **Critical Point-Finding Methods More Often Find Gradient-Flat Regions on a Neural Network Loss**.
A: Squared gradient norms across the first 100 iterations of Newton-MR for 100 separate runs on an auto-encoder loss. Gradient norms were flat after 100 iterations. See app:networks for details. Runs that terminate with squared gradient norm below 1e-10, i.e. at a candidate critical point, in blue. Runs that terminate above that cutoff and with r above 0.9, i.e. in a gradient-flat region, in gold. All other runs in black. Asterisks indicate trajectories in B. **B**: The relative residual norm r, an index of gradient-flatness, for the approximate Newton update computed by MR-QLP at each iteration (solid lines) for three representative traces. Values are local averages with a window size of 10 iterations. Raw values are plotted with reduced opacity underneath. Top: non-flat, non-critical point (black). Middle: flat, non-critical point (gold). Bottom: flat, critical point (blue). **C**: Empirical cumulative distribution functions for the final (top) and maximal (bottom) relative residual norm r observed during each run of Newton-MR. Values above the cutoff for approximate gradient-flatness, $r > 0.9$, in gold. Observations from runs that terminated below the cutoff for critical points, $\|\nabla L(\theta)\|^2 <$ 1e-10, indicated with blue ticks. **D**: Loss and index for the maximally gradient-flat points obtained during application of Newton-MR. Points with squared gradient norm below 1e-10 in blue. Other points colored by their gradient-flatness: points above 0.9 in gold, points below in black. Only points with squared gradient norm below 1e-4 shown. Reproduced, with permission, from [29].

However, our analysis suggests that the majority of these points are not critical points but either strict gradient-flat points (gold) or simply points of spurious or incomplete Newton convergence (black). The approximately critical points we do see (blue) have a very different loss-index relationship: their loss is equal to the loss of a network that has constant output equal to the mean of the data, and their index is low, but not 0. This suggests that the results presented in [22] and [67] are not evidence of the reported loss-index relationship at critical points of neural network losses.

3.5.2 Gradient-Flat Regions on a Classifier Loss

We repeated these experiments on a fully-connected classifier (aka a *multilayer perceptron* or MLP) trained on the same MNIST images via the cross-entropy cost function. This network also had two hidden layers of 12 and 8 units, used Swish activations, and included biases. Unlike all networks considered up to this point, the regularizer r was non-zero. Losses based on the cross entropy cost function can have critical points of infinite norm in the absence of regularization, and so we applied ℓ_2 norm regularization: $r(\theta) = \|\theta\|^2$.

We again found that the performance of the Newton-MR critical point-finding algorithm was poor (Figure 3.7A) and that around 90% of runs encountered a point with gradient-flatness above 0.9 (Figure 3.7C, bottom row). However, we observed that fewer runs terminated at a gradient-flat point (Figure 3.7C, top row).

In many traces, the value of r oscillates from values close to 1 to middling values, indicating that the algorithm is bouncing in and out of one or more gradient-flat regions, rather than because of another type of spurious Newton convergence. This can occur when the final target of convergence given infinite iterations is a gradient-flat point, as in the example in Section 3.4.2. This behavior is evident in the traces presented in the top and bottom rows of Figure 3.7B. Even for the autoencoding network presented in Figure 3.6, not all traces exhibit the monotonic behavior for the value of r apparent in Figure 3.6B.

If we measure the loss-index relationship at the (mostly non-gradient-flat) final points, we see the same pattern as in Figure 3.6: convex shape, separation of critical points from gradient-flat points (Figure 3.7D). This also holds if we look at the maximally flat points, as in Figure 3.6D, or if we look at the final iterates of the traces in Figure 3.6 (neither results shown). This underlines a particular problem with detecting convergence issues caused by a gradient-flat region. On any given iterate, the algorithm may be inside or outside the gradient-flat region, so it is insufficient just to examine the degree of flatness on one iteration, e.g. the final iteration, as determined by computational budget.

3.5.3 Gradient-Flat Regions on an Over-Parameterized Loss

Many contemporary neural networks have extremely large parameter counts: in the millions and tens of millions. By varying definitions of the relationship between parameter count and size or complexity of the dataset, these networks are *over-parameterized*. As described in Section 1.4.4, recent theoretical results have suggested that, even in the possible presence of bad local minima, neural networks might be easily trainable if they are

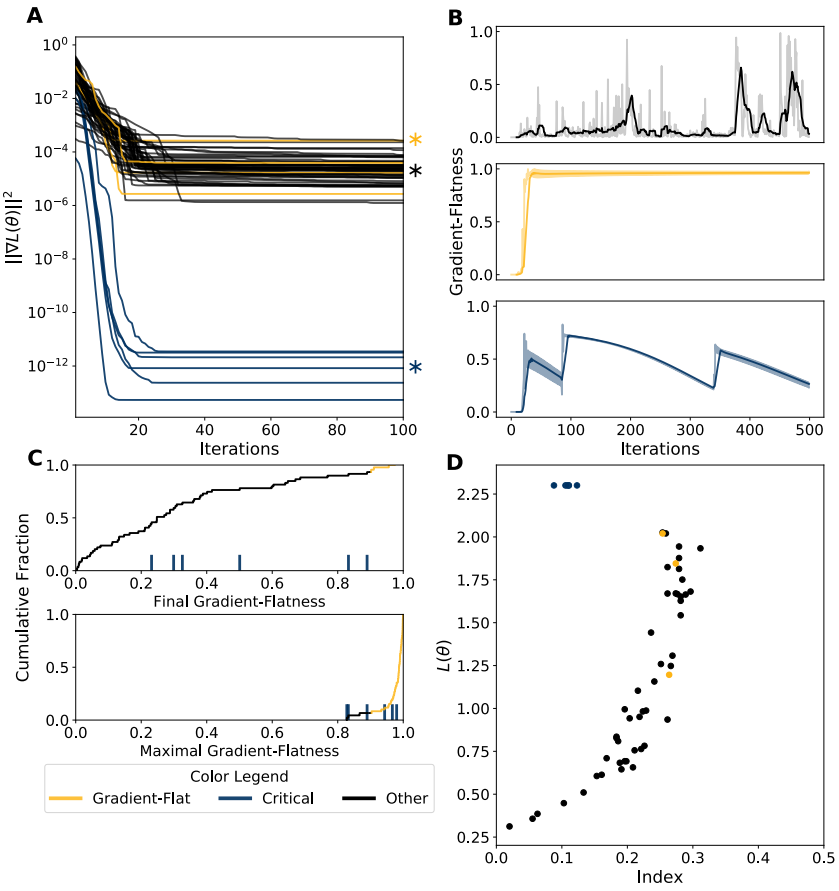


Figure 3.7: **Gradient-Flat Regions Also Appear on an MLP Loss**.
A: Squared gradient norms across the first 100 iterations of Newton-MR for 60 separate runs on an MLP loss. Runs that terminate with squared gradient norm below 1e-10 in blue. Runs that terminate above that cutoff and with r above 0.9, in gold. All other runs in black. Asterisks indicate trajectories in *B*. **B**: The relative residual norm r, for the approximate Newton update computed by MR-QLP at each iteration for three representative traces. Values are local averages with a window size of 10 iterations. Raw values are plotted with reduced opacity underneath. Top: non-flat, non-critical point (black). Middle: flat, non-critical point (gold). Bottom: non-flat, critical point (blue). **C**: Empirical cumulative distribution functions for the final (top) and maximal (bottom) relative residual norm r. Values above the cutoff for approximate gradient-flatness, $r > 0.9$, in gold. Observations from runs that terminated below the cutoff for critical points, $\|\nabla L(\theta)\|^2 <$ 1e-10, indicated with blue ticks. **D**: Loss and index for the points found after 500 iterations of Newton-MR. Colors as in top-left; only points with squared gradient norm below 1e-4 shown. Note that color is determined by the value of r on the last iteration, rather than on the iteration with maximal r, as in Figure 3.6. Reproduced, with permission, from [29].

over-parameterized. It is furthermore known that increasing the number of parameters while holding the dataset constant increases the size of the Hessian kernel [72]. The loss function of such networks has not been considered from the critical point perspective.

We repeated our critical point-finding experiments on the loss function of a fully-connected classifier on a small subset of 50 0s and 1s from the MNIST dataset. The images were PCA-downsampled to 32 dimensions using sklearn [66] the labels permuted, turning the classification task into a memorization task, as in [81]. This classifier had two hidden layers of sizes 32 and 4, no biases, also used Swish activations, and was trained with ℓ_2 regularization. In this setting, the network is over-parameterized in several senses: it has a hidden layer almost as wide as the number of points in the dataset (32 vs 50), it has more parameters than there are examples in the dataset (1160 vs 50), and it is also capable of achieving 100% accuracy on the task of random label memorization.

We again observe that the majority of runs of Newton-MR terminate with high squared gradient norm (33 out of 50 runs above 1e-8) and a similar fraction (31 out of 50 runs) encounter gradient-flat points (Figure 3.8A and C, bottom panel). The loss-index relationship looks qualitatively different, as might be expected for a task with random labels. Notice the appearance of a bad local minimum: the blue point at index 0 and loss ln(2).

3.6 Conclusion

We began by seeking to understand why neural networks are so easily trained: despite the substantial non-convexity of their loss functions, first-order optimization methods produce near-global-optima from random initial points. In Chapter 1, we developed the "no-bad-local-minima" (NBLM) theory, which posits that, like certain classes of random functions, the loss functions of neural networks have no local minima that are much worse than the global minima. While numerical experiments in [22] and [67] agreed with this hypothesis, more recent analytical results [23] suggest that it is false. These experiments relied on the ability to find the critical points of the loss function, and so we developed an understanding of critical point-finding algorithms in Chapter 2 and Section 3.4.1. This led us to identify gradient-flat regions, where the gradient is nearly in the approximate kernel of the Hessian, as a source of trouble for these algorithms: effectively, bad local minima for the problem of critical point-finding. We ended this chapter by observing that gradient-flat regions are a prevalent feature of three prototypical neural network loss functions: those of an autoencoder (Figure 3.6), a classifier (Figure 3.7), and an overparameterized network (Figure 3.8) applied to versions of the MNIST dataset. We now conclude the thesis by considering the implications of our observations for critical point-finding experiments, for the nature of neural network loss functions, and for the optimization of neural networks.

The strategy of using gradient norm cutoffs to determine whether a point is near enough to a critical point for the loss and index to match the true value is natural. However, in the absence of guarantees on the smoothness of the behavior of the Hessian (and its spectrum) around the critical point, the numerical value sufficient to guarantee correctness is unclear. The observation of gradient-flat regions at extremely low gradient

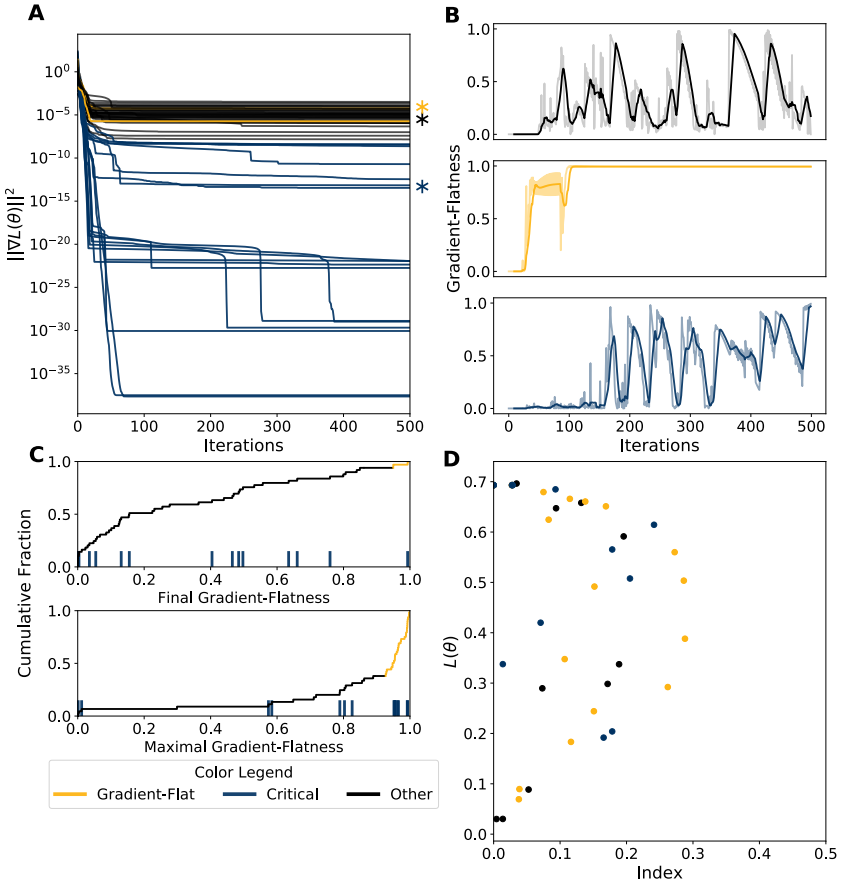


Figure 3.8: **Gradient-Flat Regions Also Appear on an Over-Parameterized Loss**. **A**: Squared gradient norms across 500 iterations of Newton-MR for 50 separate runs on the loss of an over-parameterized network. Runs that terminate with squared gradient norm below 1e-8 in blue. Runs that terminate above that cutoff and with r above 0.9, in gold. All other runs in black. Asterisks indicate trajectories in B. **B**: The relative residual norm r, for the approximate Newton update computed by MR-QLP at each iteration for three representative traces. Values are local averages with a window size of 10 iterations. Raw values are plotted with reduced opacity underneath. Top: non-flat, non-critical point (black). Middle: flat, non-critical point (gold). Bottom: flat, critical point (blue). **C**: Empirical cumulative distribution functions for the final (top) and maximal (bottom) relative residual norm r. Values above the cutoff for approximate gradient-flatness, $r > 0.9$, in gold. Observations from runs that terminated below the cutoff for critical points, $\|\nabla L(\theta)\|^2 <$ 1e-10, indicated with blue ticks. **D**: Loss and index for the points found after 500 iterations of Newton-MR. Colors as in top-left; only points with squared gradient norm below 1e-4 shown. Reproduced, with permission, from [29].

norm and the separation of these values, in terms of loss-index relationship, from the bulk of the observations suggest that there may be spurious targets of convergence for critical point-finding methods even at such low gradient norm. Alternatively, they may in fact be near real critical points, and so indicate that the simple, convex picture of loss-index relationship painted by the numerical results in [22] and [67] is incomplete. Furthermore, our observation of singular Hessians at low gradient norm suggests that some approximate saddle points of neural network losses may be degenerate (as defined in [42]) and non-strict (as defined in [50]), which indicates that gradient descent may be attracted to these points, according to the analyses in [42] and [50]. These points need not be local minima. However, in two cases we observe the lowest-index saddles at low values of the loss (see Figure 3.6, Figure 3.7) and so these analyses still predict that gradient descent will successfully reduce the loss, even if it doesn't find a local minimum. In the third case, an over-parameterized network Figure 3.8, we do observe a bad local minimum, as predicted in [23] for networks capable of achieving 0 training error.

Our results motivate a revisiting of the numerical results in [22] and [67]. Looking back at Figure 4 of [22], we see that their non-convex Newton method, a second-order optimization algorithm designed to avoid saddle points by reversing the Newton update along directions of negative curvature, appears to terminate at a gradient norm of order 1. This is only a single order of magnitude lower than what was observed during training. It is likely that this point was either in a gradient-flat region or otherwise had sufficient gradient norm in the Hessian kernel to slow the progress of their algorithm. This suggests that second-order methods designed for optimization, which use the loss as a merit function, rather than norms of the gradient, can terminate in gradient-flat regions. In this case, the merit function encourages convergence to points where the loss, rather than the gradient norm, is small, but it still cannot guarantee convergence to a critical point. The authors of [22] do not report a gradient norm cutoff, among other details needed to recreate their critical point-finding experiments, so it is unclear to which kind of points they converged. If, however, the norms are as large as those of the targets of their non-convex Newton method, in accordance with our experience with damped Newton methods and that of [20], then the loss-index relationships reported in their Figure 1 are likely to be for gradient-flat points, rather than critical points.

The authors of [67] do report a squared gradient norm cutoff of 1e-6. This cutoff is right in the middle of the bulk of values we observed, and which we labeled gradient-flat regions and points of spurious convergence, based on the experiments in Section 3.2.3, which separates a small fraction of runs from this bulk. This suggests that some of their putative critical points were gradient-flat points. Their Figure 6 shows a disagreement between their predictions for the index, based on a loss-weighted mixture of Wishart and Wigner random matrices, and their observations. We speculate that some of this gap is due to their method recovering approximate gradient-flat points rather than critical points.

Even in the face of results indicating the existence of bad local minima [23], it remains possible that bad local minima of the loss are avoided by initialization and optimization strategies. For example ReLU networks suffer from bad local minima when one layer's activations are all 0, or when the biases are initialized at too small of a value [37], but

careful initialization and training can avoid the issue. Our results do not directly invalidate this hypothesis, but they do call the supporting numerical evidence into question. Our observation of gradient-flat regions on almost every single run suggests that, while critical points are hard to find and may even be rare, regions where gradient norm is extremely small are neither. For non-smooth losses, e.g. those of ReLU networks or networks with max-pooling, whose loss gradients can have discontinuities, critical points need not exist, but gradient-flat regions may. Indeed, in some cases, the only differentiable minima in ReLU networks are also flat [46].

Other types of critical point-finding methods are not necessarily attracted to gradient-flat regions, in particular Newton homotopy methods (first used on neural networks in the 90s [20], then revived in the 2010s [9, 55]), which are popular in algebraic geometry [10]. However, singular Hessians still cause issues: for a singular Hessian H, the curve to be continued by the homotopy becomes a manifold with dimension $1 + \operatorname{co\,rk}(H)$, and orientation becomes more difficult. This can be avoided by removing the singularity of the Hessian, e.g. by the randomly-weighted regularization method in [56]. However, while these techniques may make it possible to find critical points, they fundamentally alter the loss surface, limiting their utility in drawing conclusions about other features of the loss.

The authors of [72] emphasize that when the Hessian is singular everywhere, the notion of a basin of attraction is misleading, since targets of convergence form connected manifolds and some assumptions in theorems guaranteeing first-order convergence become invalid [42], though with sufficient, if unrealistic, over-parameterization convergence can be proven [25]. They speculate that a better approach to understanding the behavior of optimizers focuses on their exploration of the sub-level sets of the loss. Our results corroborate that speculation and further indicate that this flatness means using second-order methods to try to accelerate exploration of these regions in search of minimizers is likely to fail: the alignment of the gradient with the Hessian's approximate kernel will tend to produce extremely large steps, for some methods, or no acceleration and even convergence to non-minimizers, for others.

Our observation of ubiquitous gradient-flatness further provides an alternative explanation for the success and popularity of approximate second-order optimizers for neural networks, like K-FAC [53], which uses a layerwise approximation to the Hessian. These methods are typically motivated by appeals to the computational cost of even Hessian-free exact second-order methods and their brittleness in the stochastic (non-batch) setting. However, exact second-order methods are only justified when the second-order model is good, and at an exact gradient-flat point, the second-order model can be infinitely bad, in a sense, along the direction of the gradient. Approximations need not share this property. Even more extreme approximations, like the diagonal approximations in the adaptive gradient family (e.g. AdaGrad [26], Adam [43]), behave very reasonably in gradient-flat regions: they smoothly scale up the gradient in the directions in which it is small and changing slowly, without making a quadratic model that is optimal in a local sense but poor in a global sense.

Overall, our results underscore the difficulty of searching for critical points of singular non-convex functions, including deep network loss functions, and shed new light on other numerical results in this field. In this setting, second-order methods for finding critical

points can fail badly, by converging to gradient-flat points. This failure can be hard to detect unless it is specifically measured. Furthermore, gradient-flat points are generally places where quadratic approximations become untrustworthy, and so our observations are of relevance for the design of exact and approximate second-order optimization methods as well.

Bibliography

[1] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. *A Convergence Theory for Deep Learning via Over-Parameterization*. 2018. arXiv: 1811.03962 [cs.LG].

[2] Paolo Aluffi. *Algebra: Chapter 0*. Providence, R.I: American Mathematical Society, 2009. ISBN: 978-0821847817.

[3] Anima Anandkumar et al. "Tensor decompositions for learning latent variable models". In: (2012). eprint: arXiv:1210.7559.

[4] L. Angelani et al. "Saddles in the Energy Landscape Probed by Supercooled Liquids". In: *Physical Review Letters* 85.25 (2000), pp. 5356–5359.

[5] Larry Armijo. "Minimization of functions having Lipschitz continuous first partial derivatives". In: *Pacific Journal of Mathematics* 16.1 (Jan. 1966), pp. 1–3. DOI: 10.2140/pjm.1966.16.1.

[6] Sanjeev Arora et al. *On Exact Computation with an Infinitely Wide Neural Net*. 2019. arXiv: 1904.11955 [cs.LG].

[7] Peter Auer, Mark Herbster, and Manfred K Warmuth. "Exponentially many local minima for single neurons". In: *Advances in Neural Information Processing Systems 8*. Ed. by D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo. MIT Press, 1996, pp. 316–322.

[8] Pierre Baldi and Kurt Hornik. "Neural networks and principal component analysis: Learning from examples without local minima". In: *Neural Networks* 2.1 (1989), pp. 53 –58.

[9] Andrew J. Ballard et al. "Energy landscapes for machine learning". In: *Phys. Chem. Chem. Phys.* 19 (20 2017), pp. 12585–12603.

[10] Daniel J. Bates et al. *Numerically Solving Polynomial Systems with Bertini (Software, Environments and Tools)*. Society for Industrial and Applied Mathematics, 2013. ISBN: 1611972698.

[11] Anthony J. Bell and Terrence J. Sejnowski. "The "independent components" of natural scenes are edge filters". In: *Vision Research* 37.23 (Dec. 1997), pp. 3327–3338. DOI: 10.1016/s0042-6989(97)00121-1.

[12] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004. ISBN: 0521833787.

[13] Alan J. Bray and David S. Dean. "Statistics of Critical Points of Gaussian Fields on Large-Dimensional Spaces". In: *Phys. Rev. Lett.* 98 (15 2007), p. 150201.

[14] Kurt Broderix et al. "Energy Landscape of a Lennard-Jones Liquid: statistics of Stationary Points". In: *Physical Review Letters* 85.25 (2000), pp. 5360–5363.

[15] Ezra Brown. "Square roots from 1; 24, 51, 10 to Dan Shanks". In: *The College Mathematics Journal* 30.2 (1999), pp. 82–95.

[16] Sébastien Bubeck. "Convex Optimization: Algorithms and Complexity". In: *Foundations and Trends® in Machine Learning* 8.3-4 (2015), pp. 231–357. DOI: 10.1561/2200000050.

[17] Richard H. Byrd, Marcelo Marazzi, and Jorge Nocedal. "On the convergence of Newton iterations to non-stationary points". In: *Mathematical Programming* 99.1 (Jan. 2004), pp. 127–148. DOI: 10.1007/s10107-003-0376-8.

[18] Charles J. Cerjan and William H. Miller. "On finding transition states". In: *The Journal of Chemical Physics* 75.6 (Sept. 1981), pp. 2800–2806. DOI: 10.1063/1.442352.

[19] Sou-Cheng T. Choi, Christopher C. Paige, and Michael A. Saunders. "MINRES-QLP: A Krylov Subspace Method for Indefinite or Singular Symmetric Systems". In: *SIAM Journal on Scientific Computing* 33.4 (2011), pp. 1810–1836.

[20] Frans Coetzee and Virginia L. Stonick. "488 Solutions to the XOR Problem". In: *Advances in Neural Information Processing Systems 9*. Ed. by M. C. Mozer, M. I. Jordan, and T. Petsche. MIT Press, 1997, pp. 410–416.

[21] Pierre Comon. "Tensor Decompositions, State of the Art and Applications". In: (2009). arXiv: 0905.0454.

[22] Yann Dauphin et al. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". In: *CoRR* abs/1406.2572 (2014).

[23] Tian Ding, Dawei Li, and Ruoyu Sun. *Sub-Optimal Local Minima Exist for Almost All Over-parameterized Neural Networks*. 2019. eprint: arXiv:1911.01413.

[24] Jonathan P. K. Doye and David J. Wales. "Saddle points and dynamics of Lennard-Jones clusters, solids, and supercooled liquids". In: *The Journal of Chemical Physics* 116.9 (2002), pp. 3777–3788.

[25] Simon S. Du et al. "Gradient Descent Provably Optimizes Over-parameterized Neural Networks". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=S1eK3i09YQ.

[26] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *J. Mach. Learn. Res.* 12.null (July 2011), 21212159. ISSN: 1532-4435.

[27] A.R. Feier. *Methods of Proof in Random Matrix Theory*. Harvard University, 2012.

[28] C Daniel Freeman and Joan Bruna. "Topology and geometry of half-rectified network optimization". In: *arXiv preprint arXiv:1611.01540* (2016).

[29] Charles G. Frye et al. *Critical Point-Finding Methods Reveal Gradient-Flat Regions of Deep Network Losses*. 2020. arXiv: 2003.10397 [cs.LG].

[30] Charles G. Frye et al. "Numerically Recovering the Critical Points of a Deep Linear Autoencoder". In: (2019). eprint: arXiv:1901.10603.

[31] Rong Ge et al. *Escaping From Saddle Points — Online Stochastic Gradient for Tensor Decomposition*. 2015. arXiv: 1503.02101.

[32] Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. *An Investigation into Neural Net Optimization via Hessian Eigenvalue Density*. 2019. arXiv: 1901.10159 [cs.LG].

[33] Henry Gouk et al. "Regularisation of Neural Networks by Enforcing Lipschitz Continuity". In: *arXiv preprint arXiv:1804.04368* (2018).

[34] A. Griewank and M. R. Osborne. "Analysis of Newton's Method at Irregular Singularities". In: *SIAM Journal on Numerical Analysis* 20.4 (1983), pp. 747–773. ISSN: 00361429.

[35] David Harvey and Joris Van Der Hoeven. "Integer multiplication in time O(n log n)". Mar. 2019. URL: https://hal.archives-ouvertes.fr/hal-02070778.

[36] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics)*. Springer, 2016. ISBN: 0387848576.

[37] David Holzmüller and Ingo Steinwart. *Training Two-Layer ReLU Networks with Gradient Descent is Inconsistent*. 2020. arXiv: 2002.04861 [stat.ML].

[38] D. A. Huckaby and T. F. Chan. "Stewart's pivoted QLP decomposition for low-rank matrices". In: *Numerical Linear Algebra with Applications* 12.2-3 (2005), pp. 153–159. DOI: 10.1002/nla.404.

[39] "Invex Functions (The Smooth Case)". In: *Nonconvex Optimization and Its Applications*. Springer Berlin Heidelberg, 2008, pp. 11–38. DOI: 10.1007/978-3-540-78562-0_2.

[40] Arthur Jacot, Franck Gabriel, and Clément Hongler. "Neural Tangent Kernel: Convergence and Generalization in Neural Networks". In: (2018). eprint: arXiv:1806.07572.

[41] Chi Jin, Praneeth Netrapalli, and Michael I. Jordan. "Accelerated Gradient Descent Escapes Saddle Points Faster than Gradient Descent". In: *Proceedings of the 31st Conference On Learning Theory*. Ed. by Sébastien Bubeck, Vianney Perchet, and Philippe Rigollet. Vol. 75. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1042–1085.

[42] Chi Jin et al. "How to Escape Saddle Points Efficiently". In: *CoRR* abs/1703.00887 (2017). arXiv: 1703.00887.

[43] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. eprint: arXiv:1412.6980.

[44] Nick Kollerstrom. "Thomas Simpson and 'Newton's method of approximation': an enduring myth". In: *The British Journal for the History of Science* 25.3 (Sept. 1992), pp. 347–354. DOI: 10.1017/s0007087400029150.

[45] Thomas Laurent and James von Brecht. "Deep linear neural networks with arbitrary loss: all local minima are global". In: *CoRR* abs/1712.01473 (2017). arXiv: 1712.01473.

[46] Thomas Laurent and James von Brecht. *The Multilinear Structure of ReLU Networks*. 2017. arXiv: 1712.10132 [cs.LG].

[47] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521 (2015), pp. 436 –444.

[48] Yann LeCun, Corinna Cortes, and CJ Burges. "MNIST handwritten digit database". In: *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist* 2 (2010).

[49] Jason D. Lee et al. "First-Order Methods Almost Always Avoid Strict Saddle Points". In: *Math. Program.* 176.1-2 (July 2019), pp. 311–337. ISSN: 0025-5610. DOI: 10.1007/s10107-019-01374-3.

[50] Jason D. Lee et al. "Gradient Descent Only Converges to Minimizers". In: *29th Annual Conference on Learning Theory*. Ed. by Vitaly Feldman, Alexander Rakhlin, and Ohad Shamir. Vol. 49. Proceedings of Machine Learning Research. Columbia University, New York, New York, USA: PMLR, 2016, pp. 1246–1257.

[51] Dawei Li, Tian Ding, and Ruoyu Sun. *On the Benefit of Width for Neural Networks: Disappearance of Bad Basins*. 2018. eprint: arXiv:1812.11039.

[52] Zhiyuan Li et al. *Enhanced Convolutional Neural Tangent Kernels*. 2019. arXiv: 1911.00809 [cs.LG].

[53] James Martens and Roger Grosse. *Optimizing Neural Networks with Kronecker-factored Approximate Curvature*. 2015. eprint: arXiv:1503.05671.

[54] James W. McIver and Andrew Komornicki. "Structure of transition states in organic reactions. General theory and an application to the cyclobutene-butadiene isomerization using a semiempirical molecular orbital method". In: *Journal of the American Chemical Society* 94.8 (1972), pp. 2625–2633.

[55] Dhagash Mehta et al. "Loss surface of XOR artificial neural networks". In: *Physical Review E* 97.5 (2018).

[56] Dhagash Mehta et al. *The loss surface of deep linear networks viewed through the algebraic geometry lens*. 2018. eprint: arXiv:1810.07716.

[57] Bartosz Milewski. *Understanding Products and Universal Constructions*. 2014. URL: https://bartoszmilewski.com/2014/05/05/understanding-products-and-universal-constructions/.

[58] Katta G. Murty and Santosh N. Kabadi. "Some NP-complete problems in quadratic and nonlinear programming". In: *Mathematical Programming* 39.2 (June 1987), pp. 117–129. DOI: 10.1007/bf02592948.

[59] Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer New York, 1996. DOI: 10.1007/978-1-4612-0745-0.

[60] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. 2. ed. Springer series in operations research and financial engineering. New York, NY: Springer, 2006. XXII, 664. ISBN: 978-0-387-30303-1.

[61] Roman Novak et al. *Neural Tangents: Fast and Easy Infinite Neural Networks in Python*. 2019. arXiv: 1912.02803 [stat.ML].

[62] Bruno A. Olshausen and David J. Field. "Emergence of simple-cell receptive field properties by learning a sparse code for natural images". In: *Nature* 381.6583 (June 1996), pp. 607–609. DOI: 10.1038/381607a0.

[63] C. C. Paige and M. A. Saunders. "Solution of Sparse Indefinite Systems of Linear Equations". In: *SIAM Journal on Numerical Analysis* 12.4 (Sept. 1975), pp. 617–629. DOI: 10.1137/0712047.

[64] Adam Paszke et al. "PyTorch: An imperative style, high-performance deep learning library". In: *Advances in Neural Information Processing Systems*. 2019, pp. 8024–8035.

[65] Barak A. Pearlmutter. "Fast Exact Multiplication by the Hessian". In: *Neural Computation* 6 (1994), pp. 147–160.

[66] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[67] Jeffrey Pennington and Yasaman Bahri. "Geometry of Neural Network Loss Surfaces via Random Matrix Theory". In: *International Conference on Learning Representations (ICLR)*. 2017.

[68] R. Penrose. "A generalized inverse for matrices". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 51.3 (1955), pp. 406–13. DOI: 10.1017/S0305004100030401.

[69] Michael JD Powell. "A hybrid method for nonlinear equations". In: *Numerical methods for nonlinear algebraic equations* (1970).

[70] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. eprint: arXiv:1710.05941.

[71] Fred Roosta et al. "Newton-MR: Newton's Method Without Smoothness or Convexity". In: *arXiv preprint arXiv:1810.00303* (2018).

[72] Levent Sagun et al. "Empirical Analysis of the Hessian of Over-Parametrized Neural Networks". In: *CoRR* abs/1706.04454 (2017). arXiv: 1706.04454.

[73] Stephen M. Stigler. "Stigler's Law of Epynomy". In: *Transactions of the New York Academy of Sciences* 39.1 Series II (Apr. 1980), pp. 147–157. DOI: 10.1111/j.2164-0947.1980.tb02775.x.

[74] Gilbert Strang. "The Fundamental Theorem of Linear Algebra". In: *The American Mathematical Monthly* 100.9 (Nov. 1993), p. 848. DOI: 10.2307/2324660.

[75] Ruoyu Sun. *Optimization for deep learning: theory and algorithms*. 2019. eprint: arXiv:1912.08957.

[76] Terence Tao. *Topics in Random Matrix Theory*. Vol. 132. American Mathematical Soc., 2012.

[77] Martin J. Wainwright and Michael I. Jordan. "Graphical Models, Exponential Families, and Variational Inference". In: *Foundations and Trends in Machine Learning* 1.1–2 (2007), pp. 1–305. DOI: 10.1561/2200000001.

[78] Philip Wolfe. "Convergence Conditions for Ascent Methods. II: Some Corrections". In: *SIAM Review* 13.2 (1971), pp. 185–188.

[79] Greg Yang. *Scaling Limits of Wide Neural Networks with Weight Sharing: Gaussian Process Behavior, Gradient Independence, and Neural Tangent Kernel Derivation*. 2019. arXiv: 1902.04760 [cs.NE].

[80] Xiao-Hu Yu and Guo-An Chen. "On the local minima free condition of backpropagation learning". In: *IEEE Transactions on Neural Networks* 6.5 (1995), pp. 1300–1303. DOI: 10.1109/72.410380.

[81] Chiyuan Zhang et al. "Understanding deep learning requires rethinking generalization". In: *CoRR* abs/1611.03530 (2016). arXiv: 1611.03530.