Efficient Second-Order Methods for Non-Convex Optimization and Machine Learning

by

Zhewei Yao

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Applied Mathematics

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Ming Gu, Co-chair
Associate Professor Michael Mahoney, Co-chair
Professor Steven Evans
Professor Satish Rao

Spring 2021

Efficient Second-Order Methods for Non-Convex Optimization and Machine Learning

Abstract

Efficient Second-Order Methods for Non-Convex Optimization and Machine Learning

by

Zhewei Yao

Doctor of Philosophy in Applied Mathematics

University of California, Berkeley

Professor Ming Gu, Co-chair

Associate Professor Michael Mahoney, Co-chair

Hessian-based analysis/computation is widely used in scientific computing. However, due to the (incorrect, but in our experience widespread) belief that Hessian-based computations are infeasible for large machine learning (ML) problems, the majority of work in ML (except for quite small problems) only performs the first-order methods. However, using sub-sampling and randomized numerical linear algebra algorithms, the computation of second-order methods can be efficiently extracted for large-scale machine learning problems. In this thesis, we consider three use cases of second-order methods as follows: (i) For non-convex optimization and/or ML problems, we propose inexact variants of three classic Newton-type methods—Trust Region method, Cubic Regularization method, and Newton-CG method, as well as a novel adaptive second-order method—AdaHessian. For classic methods, under the relaxed gradient and Hessian approximation, we theoretically prove that our methods achieve the same complexity as their full gradient and Hessian counterparts, and empirically show the efficiency and robustness of those Newton-type algorithms. For the novel AdaHessian algorithm, we incorporate moving average for the Hessian information, and illustrate its superb performance on large-scale deep learning problems compared to first-order optimizers; (ii) For machine learning and data science analysis, we develop a popular Hessian-based computation framework. The framework enables fast computations of the top Hessian eigenvalues, the Hessian trace, and the full Hessian eigenvalue/spectral density, and it can be used to analyze neural networks (NNs), including the topology of the loss landscape (i.e., curvature information) to gain insight into the behavior of different models/optimizers; (iii) For application, first, we show that Hessian-based metric can be used as the sensitivity metric to determine the robustness of every single layer of the NNs, and this can help perform mixed-precision quantization. Second, we demonstrate that second-order based adversarial attacks can achieve smaller perturbation as compared to first-order methods, which can be used as a powerful attack method to verify the robustness of a NN to adversarial attacks.

Acknowledgments

As a Ph.D. student, I have had the greatest fortune to have studied at UC Berkeley and to have worked with so many talented and inspired people. Without the help and support from the university and those people, it would not be possible for me to produce this dissertation.

I am very grateful to my advisors Ming Gu and Michael W. Mahoney for their constant support and guidance over the years, for giving me the freedom to pursue my own interests, and for teaching me how to be a better researcher and a better person. I also would like to especially thank Michael, who helped me transition into the field of optimization and machine learning.

I am extremely fortunate to collaborate with Kurt Keutzer. Being working with Kurt has been invaluable for my research quality as well as my personal development. Thank Kurt for his insightful comments and valuable advice.

I am very thankful to Steven N. Evans and Satish Rao for serving on my quals and dissertation committees. I would also particularly thank Steve for his helpful suggestions and advice on course selection and research direction selection when I was fresh to Berkeley.

I am also extremely fortunate to collaborate with many brilliant and talented researchers, including Jianfei Chen, Zhen Dong, Amir Gholami, Zixi Hu, Fred Roosta, Sheng Shen, Peng Xu, Yaoqing Yang, Tianjun Zhang, and many other people that I'm surely forgetting.

Finally, this thesis is dedicated to my parents Meihua Chen and Haijun Yao, as well as my sister Chunye Yao, for all the years of unconditional love and support. Their trust and encouragement have been invaluable. I am super fortunate and grateful to have them in my life.

Contents

Chapter 1

Introduction

In the past decades, first-order methods have taken the stage as the primary workhorse for solving large scale optimization problems, training/analyzing machine learning and data science models, and using the designed algorithms/metric for real applications due to their affordable computational costs per iteration. In particular, a plethora of research has been put into studying and developing variants of stochastic gradient descent (SGD) optimization algorithms [195, 73, 156, 227, 260, 125], first-order based analysis [206, 266], and first-order based applications [144, 222].

However, despite the success of first-order methods in optimization and machine learning, there exists disadvantages since those algorithms solely rely on gradient information. We here separately discuss the shortcomings of first-order methods for optimization algorithms, machine learning and data science analysis, and their applications. First, for optimization problems, those variants of SGD optimization algorithms can be only ensured to converge to *first-order stationary points*, where the gradient is (close to) zero. However, first-order stationary points include saddle points, which are prevalent in high-dimensional non-convex problems. This can significantly hurt the performance of the trained model at test time. Meanwhile, first-order optimization algorithms converge very slowly due to the ill-conditioning property of high-dimensional problems. Another challenging ad-hoc rule is the choice of hyperparameters and hyperparameter tuning methods for first-order methods, including learning rate (aka step size), decay schedule, choice of momentum parameters, etc. This can take a significant amount of both human labor and computing resources. Second, for machine learning and data science analysis purposes, first-order analysis cannot fully capture the topology of the landscape. For instance, gradient-based methods can hardly distinguish two quadratic functions at minimal point (i.e., $\nabla f(x) = 0$), e.g., $f(x) = x^2$ and $f(x) = 10x^2$. Finally, due to the limitations of first-order optimization algorithms and analysis, utilizing first-order methods for applications, such as sensitive measurement for machine learning models and generating adversarial examples, will lead to sub-optimal solutions.

In contrast, second-order based optimization algorithms have shown to be more resilient to ill-conditioning of the problem and have a faster convergence rate, and second-order based analysis can provide finer-scale topology of the loss landscape. As a result, incorporating second-order methods for applications can have better performance, e.g., more accurate sensitivity measurement and smaller error accumulation, as compared to first-order methods. However, second-order methods have typically been much less explored in large-scale optimization and machine learning problems due to their prohibitive computational cost per iteration and/or the prohibitive storage cost of the large Hessian matrix. Those challenges limit the applications of second-order methods for large-scale problems in high dimensions.

In this thesis, we develop novel efficient second-order methods for solving large-scale non-convex optimization and machine learning problems. The rest of the dissertation is consisted of three major parts and is organized as follows.

Part I focuses on non-convex optimization algorithms.

Chapter 2 considers two classical Newton-type optimization algorithms, trust region method (TR) and cubic regularization method (CR), for general non-convex and smooth optimization problems. to increase efficiency, the gradient and Hessian, as well as the solution to the underlying sub-problems are all suitably approximated. We show that under certain conditions on these approximations, to coverage to second-order criticality, our algorithms achieve the same optimal iteration complexity as the exact counterparts. Empirically, we show that by subsampling the gradient and Hessian, we can indeed speed up the algorithms on some real datasets. The material in this chapter has appeared in [252].

Chapter 3 extends variants of Newton-CG algorithm for nonconvex problems proposed in [199] in which only inexact estimates of the gradient and the Hessian information are available. To achieve further speedup, we propose a new pre-defined step size version for each update iteration instead of using a backtracking line search method for the suitable step size. Under the relaxed gradient and Hessian approximation condition, we show that worse-case iteration complexities to converge to an approximate second-order stationary point match the optimal rates obtained for the exact counterparts. We evaluate the performance of our proposed algorithms empirically on several machine learning models. The material in this chapter is in preparation for submission.

Chapter 4 presents a second-order stochastic optimization algorithm which dynamically incorporates the curvature of the loss function via ADAptive estimates of the HESSIAN. The main disadvantage of traditional second-order methods is their heavier per-iteration computation and poor accuracy as compared to first-order methods on state-of-the-art deep learning models. To address these, we incorporate several novel approaches in AdaHessian, including low cost Hessian diagonal approximation, spatial averaging, and temporal averaging. We show that AdaHessian achieves new state-of-the-art results by a large margin as compared to other adaptive optimization methods, including variants of Adam. The material has appeared in [249].

Part II focuses on Hessian-based analysis for machine learning and data science problems.

Chapter 5 uses top-k eigenvalues of the Hessian matrix of the neural network to analyze the difference between the local geometry of the neighborhood that the model converges when large batch size is used as compared to small batch, and the connection between robust optimization and large batch size training. The material in this chapter has appeared in [251].

Chapter 6 presents a new scalable framework that enables fast computation of Hessian (i.e., second-order derivative) information for deep neural networks, including the top eigenvalue, the trace, and the full ESD [111]. We apply this framework to study how residual connections and batch normalization affect training and the resulted generalization performance. Our extensive analysis shows new finer-scale insights, demonstrating that, while conventional wisdom is sometimes validated, in other cases it is simply incorrect. The material in this chapter has appeared in [254].

Part III focuses on the novel applications of second-order method for neural networks.

Chapter 7 introduces a novel second-order quantization method to address the mixed-precision quantization problem, where higher precision is used for certain "sensitive" layers of the neural network, and lower precision for "non-sensitive" layers. We show that the average trace of the Hessian matrix is a proper metric to measure the sensitivity of each layer, and extensively conduct experiments on different deep learning models. The material in this chapter has appeared in [250].

Chapter 8 extends the traditional TR method for the adversarial attack on neural networks to efficiently compute adversarial perturbations. Our proposed attack methods achieve comparable results with the state-of-the-art first-order attacks, but with significant speedup. The material in this chapter has appeared in [255].

Chapter 9 offers some concluding remarks and future directions. For better presentation, we defer most of the proofs, detailed numerical settings, as well as some illustrations to the appendices in Part IV.

Part I

Optimization Algorithm

Chapter 2

Inexact Non-Convex Newton-Type Methods

2.1 Introduction

We consider the following generic optimization problem

$$\min_{\mathbf{x}\in\mathbb{R}^d} F(\mathbf{x}), \tag{2.1}$$

where $F : \mathbb{R}^d \to \mathbb{R}$ is a smooth but possibly non-convex function. Over the last few decades, many optimization algorithms have been developed to solve (2.1). The bulk of these efforts in the machine learning (ML) community has been on developing first-order methods, i.e., those which solely rely on gradient information; see the recent textbooks [15, 132, 143] for excellent and in-depth treatments of such class of methods. Such algorithms, however, can generally be, at best, ensured to converge to *first-order stationary points*, i.e., $\mathbf{x}$ for which $\|\nabla F(\mathbf{x})\| = 0$, which include *saddle-points*. It has been argued that converging to saddle points can be undesirable for obtaining good generalization errors with many non-convex machine learning models, such as deep neural networks [62, 56, 207, 137]. In fact, it has also been shown that in certain settings, existence of "bad" local minima, i.e., sub-optimal local minima with high training error, can significantly hurt the performance of the trained model at test time [224, 80]. Important cases have also been demonstrated where, stochastic gradient descent, which is, nowadays, arguably the optimization method of choice in ML, indeed stagnates at high training error [104]. As a result, scalable algorithms which avoid saddle points and guarantee convergence to a local minimum are desired.

Second-order methods, on the other hand, which effectively employ the curvature information in the form of Hessian, have the potential for convergence to second-order stationary points, i.e., $\mathbf{x}$ for which $\|\nabla F(\mathbf{x})\| = 0$ and $\nabla^2 F(\mathbf{x}) \succeq 0$. However, the main challenge preventing the ubiquitous use of these methods is the computational costs involving the application of the underlying matrices, e.g., Hessian. In an effort to address these computational challenges, for large-scale convex settings, stochastic variants of Newton's methods have been shown, not only, to enjoy desirable theoretical properties, e.g., fast convergence rates and robustness to problem ill-conditioning [196, 247, 23], but also to exhibit superior empirical performance [131, 18].

For non-convex optimization, however, the development of similar efficient methods lags significantly behind. Indeed, designing efficient and Hessian-free variants of classic non-convex Newton-type methods such as trust-region (TR) [57], cubic regularization (CR) [169], and its adaptive variant (ARC) [39, 40], can be an appropriate place to start bridging this gap. This is, in particular, encouraging since Hessian-free methods only involve Hessian-vector products, which in many cases including neural networks [95, 180], are computed as efficiently as evaluating gradients. In this light, coupling *stochastic approximation* with *Hessian-free* techniques indeed holds promise for many of the challenging ML problems of today e.g., [153, 243, 193].

In many applications, however, even accessing the exact gradient information can be very expensive. For example, for finite-sum problems in high dimensions, where

$$F(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x}), \tag{2.2}$$

computing the exact gradient requires a pass over the entire data, which can be costly when $n \gg 1$. Inexact access to *both* the gradient and Hessian information can usually help reduce the underlying computational costs [196, 229]. Here, we aim to advance the developments in the above mentioned directions.

The rest of this paper is organized as follows. We briefly highlight the main contributions of the present paper in Section 2.1.1. A brief survey of the related work is gathered in Section 2.1.2. Notation and assumptions used throughout the paper are introduced in Section 2.1.3. We present the detailed theoretical analysis of our proposed methods in Section 2.2. In particular, analysis of TR and ARC, are respectively, given in Section 2.2.1 and Section 2.2.2. Treatment of the finite-sum problem (2.2) is presented in Section 2.2.3. Section 2.3 contains some numerical examples. Conclusions and further thoughts are gathered in Section 2.4.

2.1.1 Contributions

Here, we further these ideas by analyzing *inexact* variants of TR and ARC algorithms, which, to increase efficiency, incorporate *approximations* of *gradient* and *Hessian information*, as well as solutions of the underlying *sub-problems*. Our algorithms are motivated by the works of [41, 246], which analyzed the variants of TR and ARC where Hessian is approximated but the accurate gradient information is required. We will show that, under mild conditions on approximations of the gradient, Hessian, as well as sub-problem solves, our proposed inexact TR and ARC algorithms can retain the same optimal worst-case convergence guarantees as the exact counterparts [43, 41]. More specifically, to achieve (ϵ_g, ϵ_H)-Optimality (cf. Definition 1), we show the following.

Table 2.1: Comparison of optimal worst-case iteration complexities for convergence to a $(\epsilon, \sqrt{\epsilon})-$ Optimality (cf. Definition 1), among different second-order methods for non-convex optimization. TR and CR refer, respectively, to the class of trust region and cubic regularization methods. "Practically Implementable" refers to an algorithm which not only does not require exhaustive search over hyper-parameter space for tuning, but also failure to precisely "fine-tune" is not likely to result in unwanted behaviors, e.g., divergence or stagnation.

Method Class	Iteration Complexity	Inexact Hessian	Inexact Gradient	Practically Implementable
TR [41]	$\mathcal{O}(\epsilon^{-2.5})$	✓	✗	✓
TR [246]	$\mathcal{O}(\epsilon^{-2.5})$	✓	✗	✓
TR (Algorithm 1)	$\mathcal{O}(\epsilon^{-2.5})$	✓	✓	✓
CR [41]	$\mathcal{O}(\epsilon^{-1.5})$	✓	✗	✓
CR [246]	$\mathcal{O}(\epsilon^{-1.5})$	✓	✗	✓
CR [229]	$\mathcal{O}(\epsilon^{-1.5})$	✓	✓	✗
CR (Algorithm 2)	$\mathcal{O}(\epsilon^{-1.5})$	✓	✓	✓

(i) *Inexact TR (Algorithm 1)*, under Condition 1 on the gradient and Hessian approximation, and Condition 2 on approximate sub-problem solves, requires the optimal iteration complexity of $\mathcal{O}(\max\{\epsilon_g^{-2}\epsilon_H^{-1}, \epsilon_H^{-3}\})$. In particular, we obtain convergence for a practical case where the accuracy tolerances in gradient and Hessian estimations, i.e., (δ_g, δ_H) in Condition 1, are adaptively chosen; see Section 2.2.1 for more details.

(ii) *Inexact ARC (Algorithm 2)*, under Condition 3 on the gradient and Hessian approximation, and Condition 4 on approximate sub-problem solves, requires less than $\mathcal{O}(\max\{\epsilon_g^{-2}, \epsilon_H^{-3}\})$, which is sub-optimal. These conditions are described in Section 2.2.2.1. However, under respectively stronger Conditions 5 and 6, the optimal iteration complexity of $\mathcal{O}(\max\{\epsilon_g^{-3/2}, \epsilon_H^{-3}\})$ is recovered. Unfortunately, we were unable to provide convergence guarantees with adaptive tolerances and as a result (δ_g, δ_H) in Condition 3 are set fixed a priori to a sufficiently small value. The details are given in Section 2.2.2.2.

To prove our results, we follow a similar line of reasoning as that in [246]. However, incorporating gradient approximation introduces several new layers of technical difficulty. These difficulties arise as a result of the discrepancy between the true objective function and its quadratic and cubic approximations within Inexact TR and ARC, respectively, which now involve an additional "bias" term. Properly controlling such an added error term necessitates a much finer-grained analysis. For example, one has to consider various scenarios arising from large and small gradient norms. Among all of these, the case where the true gradient is small enough to be of similar magnitude as the approximation noise level requires a special treatment and analysis.

We finally empirically demonstrate the advantages of our methods on several model problems in Section 2.3. In addition to showing favorable performance, e.g., in terms of efficiency, we also highlight some additional features of our algorithms such as robustness to hyper-parameter tuning. Such properties amount to a *practically implementable* algorithm for which failure to fine-tune of the hyper-parameters is unlikely to result in divergence or stagnation.

A snapshot of comparison among our proposed methods and other similar algorithms is given in Table 2.1.

2.1.2 Related work

Due to the resurgence of deep learning, recently, there has been a rise of interest in efficient non-convex optimization algorithms. For non-convex problems, where saddle points have been shown to give understandable generalization performance, several variants of stochastic gradient descent (SGD) have recently been devised that promise to efficiently escape saddle points and, instead, converge to second-order stationary point [81, 123, 139].

As for second-order methods, there have been a few empirical studies of the application of inexact curvature information for, mostly, deep-learning applications, e.g., see the work of [153] and follow-ups, e.g., [239, 233, 105, 126]. However, the theoretical understanding of these inexact methods remains largely under-studied. Among a few related theoretical prior works, most notably are the ones which study derivative-free and probabilistic models in general, and Hessian approximation in particular for trust-region methods [58, 49, 22, 13, 133, 212, 92].

For cubic regularization, the seminal works of [39, 40] are the first to study Hessian approximation and the resulting algorithm is an adaptive variant of the cubic regularization, referred to as ARC. In [41], similar Hessian inexactness is also extended to trust region methods. However, to guarantee optimal complexity, they require not only exact gradient information but also progressively accurate Hessian information which can be difficulty to satisfy. More general treatment of line-search and cubic regularization methods based on probabilistic models are given in [44]. For minimization of a finite-sum (2.2), a sub-sampled variant of ARC was proposed in [127], which directly relies on the analysis of [39, 40]. A more refined analysis was given in [52], which incorporates sub-sampling strategies to develop both standard and accelerated ARC variants for convex objectives. More recently, [229] proposed a stochastic variant of cubic regularization, henceforth referred to as SCR, in which, in order to guarantee optimal performance, only stochastic gradient and Hessian is required. However for the practical implementation of their algorithm, one must either assume to know, rather unknowable, problem related constants, e.g, Lipschitz continuity of the gradient and Hessian, or perform an exhaustive grid search over the space of hyper-parameters.

In the context of both TR and ARC, under milder Hessian approximation conditions than prior works, [246] recently analyzed optimal complexity of variants in which the Hessian matrix is approximated, but the exact gradient is used. Our approach here builds upon the ideas in [246].

2.1.3 Notation and Assumptions

Notation: Throughout the paper, vectors and matrices are denoted by bold lowercase and blackboard bold uppercase letters, e.g., $\mathbf{v}$ and $\mathbf{V}$, respectively. We use regular lower-case and upper-case letters to denote scalar constants, e.g., c or K. The transpose of a real vector $\mathbf{v}$ is denoted by $\mathbf{v}^T$. The inner-product between two vectors $\mathbf{v}, \mathbf{w}$ is denoted by $\langle \mathbf{v}, \mathbf{w} \rangle$. For a vector $\mathbf{v}$, and a matrix $\mathbf{V}$, $\|\mathbf{v}\|$ and $\|\mathbf{V}\|$ denote the vector ℓ_2 norm and the matrix spectral norm, respectively. The subscript, e.g., $\mathbf{x}_t$, denotes iteration counter. At iteration t, the approximations of the gradient and Hessian are written, respectively, as $\mathbf{g}_t$ and $\mathbf{H}_t$. The smallest eigenvalue of matrix $\mathbf{V}$ is denoted by $\lambda_{\min}(\mathbf{V})$.

Assumptions: Unlike convex problems, where tracking the first-order condition, i.e., norm of the gradient, is sufficient to evaluate (approximate) optimality, in non-convex settings, the situation is much more involved, e.g., see examples of [165, 106]. In this light, one typically sets out to design algorithms that can guarantee convergence to approximate second-order optimality.

Definition 1 ((ϵ_g, ϵ_H)-Optimality)**.** *Given $0 < \epsilon_g, \epsilon_H < 1$, $\boldsymbol{x}$ is an (ϵ_g, ϵ_H)-Optimal solution of (2.1), if*

$$\|\nabla F(\boldsymbol{x})\| \le \epsilon_g, \quad \textit{and} \quad \lambda_{\min}(\nabla^2 F(\boldsymbol{x})) \ge -\epsilon_H. \tag{2.3}$$

For our analysis throughout the paper, we make the following standard assumptions on the smoothness of objective function F.

Assumption 1 (Hessian Regularity)**.** *$F(\boldsymbol{x})$ is twice continuously differentiable. Furthermore, there are some constants $0 < L_F, K_F < \infty$ such that for any $\boldsymbol{x} = \boldsymbol{x}_t + \tau \boldsymbol{s}_t$, $\tau \in [0, 1]$, we have*

$$\left\|\nabla^2 F(\boldsymbol{x}) - \nabla^2 F(\boldsymbol{x}_t)\right\| \le L_F \|\boldsymbol{x} - \boldsymbol{x}_t\|, \tag{2.4a}$$

$$\left\|\nabla^2 F(\boldsymbol{x}_t)\right\| \le K_F, \tag{2.4b}$$

where $\boldsymbol{x}_t$ and $\boldsymbol{s}_t$ are, respectively, the iterate and update direction at the t^{th} iteration.

For our inexact algorithms, we require that the approximate gradient, $\mathbf{g}_t$, and the inexact Hessian, $\mathbf{H}_t$, at each iteration t, satisfy the following conditions.

Assumption 2 (Gradient and Hessian Approximation Error)**.** *For some $0 < \delta_g, \delta_H < 1$, the approximations of the gradient and Hessian t^{th} iteration satisfy,*

$$\|\mathbf{g}_t - \nabla F(\boldsymbol{x}_t)\| \le \delta_g,$$
$$\|\mathbf{H}_t - \nabla^2 F(\boldsymbol{x}_t)\| \le \delta_H.$$

Note that by the triangle inequality, Assumptions 1 and 2 imply that $\|\mathbf{H}_t\| \le K_H$, where $K_H \le K_F + \delta_H$.

2.2 Algorithms and Theoretical Analysis

In this section we will present our main algorithms as well as their respective analysis, i.e., inexact variants of TR (Algorithm 1) and ARC (Algorithm 2) where the gradient, Hessian and the solution to sub-problems are all approximated.

As it can be seen from Algorithms 1 and 2, compared with the standard classical counterparts, the main differences in iterations lie in using the approximations of the gradient, Hessian, and the solution to the corresponding sub-problem (2.6) and (2.9). Another notable difference is when the gradient estimate is small, i.e., $\|\mathbf{g}_t\| \le \epsilon_g$, in which case our algorithm completely ignores the gradient; see Step 5 of Algorithms 1 and 2. This turns out to be crucial in obtaining the optimal iteration complexity for both algorithms. Intuitively, when the gradient is too small, its approximation involves a great degree of noisy information. As a result, solving the subproblems using such noisy gradient information can result in directions of ascent. In practice, however, such unfortunate steps are usually simply corrected by the subsequent steps and hence one can always safely employ the approximate gradient without any such safeguard. In this light, in our experiments, we never needed to enforce this step and opted to retain the gradient term even when it was small.

It can also be seen that Algorithms 1 and 2 are highly similar in their corresponding steps. In particular, after initialization, one computes a local model m_t of F around $\mathbf{x}_t$, and obtains a step that guarantees model reduction $m_t(\mathbf{s}_t) < m_t(\mathbf{0}) = 0$. Subsequently, one checks that the actual reduction in F is in accordance with what is predicted using the local model. More specifically, at every iteration of Algorithms 1 and 2, by computing

$$\rho_t := \frac{F(\mathbf{x}_t + \mathbf{s}_t) - F(\mathbf{x}_t)}{m_t(\mathbf{s}_t)}, \tag{2.5}$$

one checks whether $F(\mathbf{x}_t) - F(\mathbf{x}_T + \mathbf{s}_T)$ is large enough relative to the reduction in the local model $m_t(\mathbf{s}_t) - m_t(\mathbf{0})$. If ρ_t is larger than a preset threshold, the update $\mathbf{s}_t$ will be accepted and we set $\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{s}_t$. In this case, local models are "loosened" to allow for larger trial steps in the next iteration. However, a small value of ρ_t hints at a large discrepancy between the predicted and the actual reduction in F, which implies that the local models mismatch the actual function. In this case, the step is rejected and the local models are "tightened" by adjusting the trust region or cubic regularization parameters.

At a high-level, these similar algorithmic steps give rise to similar analytical steps as well. For example, the notion of Cauchy and Eigen Points plays a crucial role in the analysis of both algorithms. Generally, when the gradient is large, both algorithms adopt the Cauchy point, otherwise Eigen point is used as trial step. Furthermore, it is shown that as long as Δ_t is small enough and σ_t is large enough, the trial steps generated, respectively, by Algorithms 1 and 2 are accepted. This in turn implies that, after a fixed number of rejections, both algorithms are guaranteed to eventually accept their trial steps and hence make progress towards optimality. Since the overall number of rejected trial steps is upper bounded, we are guaranteed to obtain convergence for both algorithms. Although the high-level analysis have many common grounds, the analysis of Algorithms 1 and 2 have distinctive technical features

Algorithm 1 Inexact TR

```
1: Input:
      - Starting point: $\mathbf{x}_0$
      - Initial trust-region radius: $\Delta_0 > 0$
      - Other Parameters: $0 \leq \epsilon_g, 0 \leq \epsilon_H, 0 < \eta \leq 1, \gamma > 1$.
2: t=0
3: while $\|\mathbf{g}_t\| \geq \epsilon_g,\ \lambda_{\min}(\mathbf{H}_t) \leq -\epsilon_H$ do
4:    if $\|\mathbf{g}_t\| \leq \epsilon_g$ then
5:       $\mathbf{g}_t = 0$
6:    end if
7:    Find $\mathbf{s}_t$ as in (2.6)
8:    Set $\rho_t$ as in (2.5) with $m_t$ as in (2.6b)
9:    if $\rho_t \geq \eta$ then
10:      $\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{s}_t$ and $\Delta_{t+1} = \gamma\Delta_t$
11:   else
12:      $\mathbf{x}_{t+1} = \mathbf{x}_t$ and $\Delta_{t+1} = \Delta_t/\gamma$
13:   end if
14:   t=t+1
15: end while
16: Output: $\mathbf{x}_t$
```

Algorithm 2 Inexact ARC

```
1: Input:
      - Starting point: $\mathbf{x}_0$
      - Initial regularization parameter: $\sigma_0 > 0$
      - Other Parameters: $0 \leq \epsilon_g, 0 \leq \epsilon_H, 0 < \eta \leq 1, \gamma > 1$.
2: t=0
3: while $\|\mathbf{g}_t\| \geq \epsilon_g,\ \lambda_{\min}(\mathbf{H}_t) \leq -\epsilon_H$ do
4:    if $\|\mathbf{g}_t\| \leq \epsilon_g$ then
5:       $\mathbf{g}_t = 0$
6:    end if
7:    Find $\mathbf{s}_t$ as in (2.9)
8:    Set $\rho_t$ as in (2.5) with $m_t$ as in (2.9b)
9:    if $\rho_t \geq \eta$ then
10:      $\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{s}_t$ and $\sigma_{t+1} = \sigma_t/\gamma$
11:   else
12:      $\mathbf{x}_{t+1} = \mathbf{x}_t$ and $\sigma_{t+1} = \gamma\sigma_t$
13:   end if
14:   t=t+1
15: end while
16: Output: $\mathbf{x}_t$
```

as well. In particular, obtaining optimal complexity of Algorithm 2 requires more restrictive conditions on the solution of the sub-problem than simple Cauchy or Eigen points, and it also necessitates a more refined analysis and careful control over the size of the accepted steps at each successful iterations.

2.2.1 Inexact Trust Region

The inexact TR algorithm is depicted in Algorithm 1. Every iteration of Algorithm 1 involves approximate solution to a sub-problem of the form

$$\mathbf{s}_t \approx \underset{\|\mathbf{s}\| \leq \Delta_t}{argmin}\, m_t(\mathbf{s}), \tag{2.6a}$$

where

$$m_t(\mathbf{s}) \triangleq \begin{cases} \langle \mathbf{g}_t, \mathbf{s}\rangle + \dfrac{1}{2}\langle \mathbf{s}, \mathbf{H}_t\mathbf{s}\rangle, & \|\mathbf{g}_t\| \geq \epsilon_g \\ \\ \langle \mathbf{s}, \mathbf{H}_t\mathbf{s}\rangle, & \text{Otherwise} \end{cases}. \tag{2.6b}$$

Classically, the analysis of TR method involves obtaining a minimum descent along two important directions, namely negative gradient and (approximate) negative curvature. Updating the current point using these directions gives, respectively, what are known as Cauchy Point and Eigen Point [57]. In other words, Cauchy Point and Eigen Point, respectively, correspond to the optimal solution of (2.6) along the negative gradient and the negative curvature direction (if it exists).

Definition 2 (Cauchy Point for Algorithm 1)**.** *When $\|\mathbf{g}_t\| \geq \epsilon_g$, Cauchy Point for Algorithm 1 is obtained from* (2.6) *as*

$$\boldsymbol{s}_t^C = -\frac{\alpha^C}{\|\mathbf{g}_t\|}\mathbf{g}_t, \quad \alpha^C = \underset{0\leq\alpha\leq\Delta_t}{argmin}\, m_t\left(-\frac{\alpha}{\|\mathbf{g}_t\|}\mathbf{g}_t\right). \tag{2.7a}$$

Definition 3 (Eigen Point for Algorithm 1)**.** *When $\lambda_{\min}(\mathbf{H}_t) \leq -\epsilon_H$, Eigen Point for Algorithm 1 is obtained from* (2.6) *as*

$$\boldsymbol{s}_t^E = \alpha^E \boldsymbol{u}_t, \quad \alpha^E = \underset{|\alpha|\leq\Delta_t}{argmin}\, m_t(\alpha \boldsymbol{u}_t), \tag{2.7b}$$

where $\boldsymbol{u}_t$ is an approximation to the corresponding negative curvature direction, i.e., for some $0 < \nu < 1$,

$$\langle \boldsymbol{u}_t, \mathbf{H}_t \boldsymbol{u}_t\rangle \leq \nu\lambda_{\min}(\mathbf{H}_t) \quad \textit{and} \quad \|\boldsymbol{u}_t\| = 1.$$

The properties of Cauchy and Eigen Points have been studied in [39, 40, 246], and are also stated in Lemmas 2 and 3.

We are now ready to give the convergence guarantee of Algorithm 1. For this, we first present sufficient conditions (Condition 1) on the degree of inexactness of the gradient and Hessian. In other words, we now give conditions on δ_g, δ_H in Assumption 2 which ensure convergence.

Condition 1 (Gradient and Hessian Approximation for Algorithm 1)**.** *Given the termination criteria, ϵ_g, ϵ_H, in Algorithm 1, we require the inexact gradient and Hessian to satisfy*

$$\delta_g \leq \left(\frac{1-\eta}{4}\right)\max\left\{\epsilon_g, \|\mathbf{g}_t\|\right\} \quad \textit{and} \quad \delta_H \leq \min\left\{\max\left\{\frac{(1-\eta)\nu\epsilon_H}{2}, \Delta_t\right\}, 1\right\}. \tag{2.8}$$

Note that Condition 1 are *adaptive*, which can have desirable consequences in practice. For example, when Δ_t is large (which is typically the case during the early stages of the algorithm), one can afford cruder approximation of the Hessian by choosing larger δ_H. Similarly, the condition on δ_g, for large $\|\mathbf{g}_t\|$, amounts to a relative error condition. Although this latter condition on δ_g is perhaps not easily enforceable a priori (unless one has a lower-bound estimate of $\|\mathbf{g}_t\|$), it nonetheless qualitatively indicates that when the true gradient is large, one can very well employ loose approximations; see also Remark 3. As the algorithm progresses towards convergence, Condition 1 implies that ultimately we must seek to have

$\delta_g \in \mathcal{O}(\epsilon_g), \delta_H \in \mathcal{O}(\epsilon_H)$. These bounds are indeed the minimum requirements for the gradient and Hessian approximations to achieve (ϵ_g, ϵ_H)-Optimality; see the termination step for Algorithm 1.

In Algorithm 1, sub-problem (2.6) need only be solved approximately. Indeed, in large-scale settings, obtaining the exact solution of the sub-problem (2.6) is computationally prohibitive. For this, as it has been classically done, we require that an approximate solution of the sub-problem satisfies what are known as Cauchy and Eigen Conditions [57, 42, 246]. In other words, we require that an approximate solution to (2.6) is at least as good as Cauchy and Eigen points in Definitions 2 and 3, respectively. Condition 2 makes this explicit.

Condition 2 (Approximate solution of (2.6) for Algorithm 1)**.** *If $\|\mathbf{g}_t\| \geq \epsilon_g$, then we take the Cauchy Point, i.e. $\mathbf{s}_t = \mathbf{s}_t^C$, otherwise we take the Eigen point, i.e., $\mathbf{s}_t = \mathbf{s}_t^E$. Here, $\mathbf{s}_t^C$ and $\mathbf{s}_t^E$ are Cauchy and Eigen points, as in Definitions 2 and 3, respectively.*

Under Assumptions 1 and 2 , as well as assuming Conditions 1 and 2 hold, we are now ready to give the optimal iteration complexity of Algorithm 1.

Theorem 1 (Optimal Complexity of Algorithm 1)**.** *Let Assumption 1 hold and suppose that $\mathbf{g}_t$ and $\mathbf{H}_t$ satisfy Assumption 2 with δ_g and δ_H under Condition 1. If the approximate solution to the sub-problem (2.6) satisfies Condition 2, then Algorithm 1 terminates after at most*

$$T \in \mathcal{O}\left(\max\left\{\epsilon_g^{-2}\epsilon_H^{-1}, \epsilon_H^{-3}\right\}\right),$$

iterations.

The worst iteration complexity of Theorem 1 matches the bound obtained in [57, 41, 246], which is known to be optimal in worst-case sense [41]. Further, it follows immediately that the terminating points of Algorithm 1 satisfies $\|\mathbf{g}_T\| \leq \epsilon_g + \delta_g$ and $\lambda_{\min}(\mathbf{H}_T) \geq -\epsilon_H - \delta_h$, i.e. $\mathbf{x}_T$ is a $(\epsilon_g + \delta_g, \epsilon_H + \delta_h)$-optimal solution of (2.1).

2.2.2 Inexact ARC

The inexact ARC algorithm is given in Algorithm 2. Every iteration of Algorithm 2 involves an approximate solution to the following sub-problem:

$$\mathbf{s}_t \approx \underset{\mathbf{s}\in\mathbb{R}^d}{argmin}\, m_t(\mathbf{s}), \tag{2.9a}$$

where

$$m_t(\mathbf{s}) \triangleq \begin{cases} \langle \mathbf{g}_t, \mathbf{s}\rangle + \dfrac{1}{2}\langle \mathbf{s}, \mathbf{H}_t\mathbf{s}\rangle + \dfrac{\sigma_t}{3}\|\mathbf{s}\|^3, & \|\mathbf{g}_t\| \geq \epsilon_g \\ \dfrac{1}{2}\langle \mathbf{s}, \mathbf{H}_t\mathbf{s}\rangle + \dfrac{\sigma_t}{3}\|\mathbf{s}\|^3, & \text{Otherwise} \end{cases}. \tag{2.9b}$$

Similar to Section 2.2.1, our analysis for inexact ARC also involves Cauchy and Eigen points obtained from (2.9) as follows.

Definition 4 (Cauchy Point for Algorithm 2). *When $\|\mathbf{g}_t\| \geq \epsilon_g$, Cauchy Point for Algorithm 2 is obtained from (2.9) as*

$$\boldsymbol{s}_t^C = -\alpha^C \mathbf{g}_t, \quad \alpha^C = \underset{\alpha \geq 0}{\arg\min}\, m_t(-\alpha \mathbf{g}_t). \tag{2.10a}$$

Definition 5 (Eigen Point for Algorithm 2). *When $\lambda_{\min}(\mathbf{H}_t) \leq -\epsilon_H$, Eigen Point for Algorithm 2 is obtained from (2.9) as*

$$\boldsymbol{s}_t^E = \alpha^E \boldsymbol{u}_t, \quad \alpha^E = \underset{\alpha \in \mathbb{R}}{\arg\min}\, m_t(\alpha \boldsymbol{u}_t), \tag{2.10b}$$

where $\boldsymbol{u}_t$ is an approximation to the corresponding negative curvature direction, i.e., for some $0 < \nu < 1$,

$$\langle \boldsymbol{u}_t, \mathbf{H}_t \boldsymbol{u}_t \rangle \leq \nu \lambda_{\min}(\mathbf{H}_t) \quad \textit{and} \quad \|\boldsymbol{u}_t\| = 1.$$

Note that since both $\mathbf{s}_t^C$ and $\mathbf{s}_t^E$ are line minimizers of $m_t(\mathbf{s})$ along the directions $-\mathbf{g}_t$ and $\mathbf{u}_t$, respectively, they satisfy

$$\langle -\mathbf{g}_t, \nabla m_t(\mathbf{s}_t^C) \rangle = \langle \mathbf{s}_t^C, \nabla m_t(\mathbf{s}_t^C) \rangle = 0,$$
$$\langle \mathbf{u}_t, \nabla m_t(\mathbf{s}_t^E) \rangle = \langle \mathbf{s}_t^E, \nabla m_t(\mathbf{s}_t^E) \rangle = 0.$$

Further properties of Cauchy Point and Eigen Point for the cubic problem can be found in Lemma 10 and Lemma 11.

As we shall show, the worst-case iteration complexity of inexact ARC depends on how accurately we approximate the gradient and Hessian, as well as the problem solves. In Section 2.2.2.1, we show that under *nearly* minimum requirement of the gradient and Hessian approximation (Condition 3), the inexact ARC can achieve *sub-optimal* complexity $\mathcal{O}(\max\{\epsilon_g^{-2}, \epsilon_H^{-3}\})$. In Section 2.2.2.2, we then show that under more restrict approximation condition (Condition 5), the *optimal* worst-case complexity $\mathcal{O}(\max\{\epsilon_g^{-1.5}, \epsilon_H^{-3}\})$ can be recovered.

2.2.2.1 Sub-optimal Complexity for Algorithm 2

In this section, we provide sufficient conditions on approximating the gradient and Hessian, as well as the sub-problem solves for inexact ARC to achieve the sub-optimal complexity $\mathcal{O}(\max\{\epsilon_g^{-2}, \epsilon_H^{-3}\})$.

First, similar to Section 2.2.1, we require that the estimates of the gradient and Hessian satisfy the following condition.

Condition 3 (Gradient and Hessian Approximation for Algorithm 2). *Given the termination criteria, ϵ_g, ϵ_H, in Algorithm 2, we require the inexact gradient and Hessian to satisfy*

$$\begin{aligned} \delta_g &\leq \left(\frac{1-\eta}{12}\right) \max\{\epsilon_g, \|\mathbf{g}_t\|\}, \\ \textit{and} \quad \delta_H &\leq \left(\frac{1-\eta}{6}\right) \min\left\{\nu \max\left\{-\lambda_{\min}(\mathbf{H}_t), \epsilon_H\right\}, \sqrt{2L_F\epsilon_g}\right\}. \end{aligned} \tag{2.11}$$

It is easy to see that $\delta_g \in \mathcal{O}(\epsilon_g),\ \delta_H \in \mathcal{O}\left(\min\left\{\sqrt{\epsilon_g}, \epsilon_H\right\}\right)$. Similar constraints on δ_H have appeared in several previous works, e.g. [229, 246]. These are nearly minimum requirement for the approximation to determine whether the iterate satisfies (ϵ_g, ϵ_H)-Optimality (Definition 1). In the case when $\epsilon_H = \mathcal{O}(\sqrt{\epsilon_g})$, Condition 3 is indeed the minimum requirement. We note that the condition on δ_g and δ_H are adaptive in that, for large $\|\mathbf{g}_t\|$ and $-\lambda_{\min}(\mathbf{H}_t)$, they amount to relative error conditions on the approximate gradient and Hessian, respectively. In fact the condition on δ_g is very similar to that in Condition 1. Of course, in such such cases, these conditions cannot be a priori enforced in a straightforward manner. Nonetheless, they qualitatively indicates that in regions with large gradient and negative curvature, one can rely on loose approximations of the gradient and Hessian, respectively. As the algorithm progresses towards convergence, Condition 3 implies that ultimately we must seek to have $\delta_g \in \mathcal{O}(\epsilon_g)$ and $\delta_H \in \mathcal{O}\left(\min\left\{\sqrt{\epsilon_g}, \epsilon_H\right\}\right)$.

As for solving the sub-problem, we require the following.

Condition 4 (Approximate solution of (2.9) for Algorithm 2)**.** *We use the same trial steps as in Condition 2 but with $\boldsymbol{s}_t^C$ and $\boldsymbol{s}_t^E$ as in Definitions 4 and 5, respectively.*

Condition 4 implies that when the gradient is large-enough, we take the Cauchy step. Otherwise, we update along the Eigen Point direction.

Under Assumptions 1 and 2, as well as Conditions 3 and 4, we now present the proof of sub-optimal complexity of Algorithm 2.

Based on the above lemmas, it follows,

Theorem 2 (Complexity of Algorithm 2)**.** *Let Assumption 1 hold and consider any $0 < \epsilon_g, \epsilon_H < 1$. Further, suppose that $\mathbf{g}_t$ and $\mathbf{H}_t$ satisfy Assumption 2 with δ_g and δ_H under Condition 3. If the approximate solution to the sub-problem (2.9) satisfies Condition 4, then Algorithm 2 terminates after at most*

$$T \in \mathcal{O}\left(\max\left\{\epsilon_g^{-2}, \epsilon_H^{-3}\right\}\right),$$

iterations.

Remark 1. *To obtain similar sub-optimal iteration complexity, the sufficient condition on approximating Hessian in [246] requires that $\delta_H \in \mathcal{O}\left(\min\left\{\epsilon_g, \epsilon_H\right\}\right)$, which is stronger than Condition 3.*

2.2.2.2 Optimal Complexity for Algorithm 2

In this section, we show that by better approximation of the gradient, Hessian as well as the sub-problem (2.9), Algorithm 2 indeed enjoys the optimal iteration complexity.

First we require the following condition on approximating the gradient and Hessian.

Condition 5 (Gradient and Hessian Approximation for Algorithm 2)**.** *Given the termination criteria, ϵ_g, ϵ_H, in Algorithm 2, we require the inexact gradient and Hessian to satisfy*

$$\delta_g \leq \frac{(1-\eta)}{192 L_F}\left(\sqrt{K_H^2 + 8L_F \max\{\min\{\|\mathbf{g}_t\|, \|\mathbf{g}_{t+1}\|\}, \epsilon_g\}} - K_H\right)^2, \tag{2.12a}$$

$$\delta_H \leq \frac{(1-\eta)}{6}\min\left\{\frac{1}{4}\left(\sqrt{K_H^2 + 8L_F\|\mathbf{g}_t\|} - K_H\right), \nu\epsilon_H\right\}, \tag{2.12b}$$

$$\delta_g \leq \delta_H \leq \zeta\epsilon_g, \tag{2.12c}$$

where $0 < \zeta < 1 - \sqrt{2}/2$*.*

Condition 5 implies $\delta_g = \mathcal{O}(\epsilon_g^2)$ and $\delta_H = \mathcal{O}(\min\{\epsilon_g, \epsilon_H\})$, which is strictly stronger than Condition 3 in Section 2.2.2.1. Admittedly, although Condition 5 allows one to obtain optimal iteration complexity of Algorithm 2, it also implies more computations, e.g., for finite-sum problems of Section 2.2.3, this translates to larger sampling complexities. We suspect that, instead of being an inherent property of Algorithm 2, this is merely a by-product of our analysis. In this light, we conjecture that the same requirement as (2.11) should also be sufficient for Algorithm 2; investigating this conjecture is left for future work.

Now we provide a sufficient condition on approximating the solution of the sub-problem (2.9). Here, we require that the sub-problem (2.9) is solved more accurately than in Condition 4. To obtain optimal complexity, similar conditions have been considered in several previous works [42, 246]. Specifically we require the solution is, not only, as good as Cauchy and Eigen points, but also it satisfies an extra requirement, (2.13), which accelerates the convergence to first-order critical points.

Condition 6 (Approximate solution of (2.9) for Algorithm 2)**.** *If $\|\mathbf{g}_t\| \geq \epsilon_g$, find $\boldsymbol{s}_t$, such that $m_t(\boldsymbol{s}_t) \leq m_t(\boldsymbol{s}_t^C)$ and*

$$\|\nabla m(\boldsymbol{s}_t)\| \leq \theta_t\|\mathbf{g}_t\|, \qquad \theta_t \leq \min\{\zeta, 1/5, \|\boldsymbol{s}_t\|/5\}. \tag{2.13}$$

Otherwise, we take the Eigen Point, i.e. $\boldsymbol{s}_t = \boldsymbol{s}_t^E$. Here, $\boldsymbol{s}_t^C$ and $\boldsymbol{s}_t^E$ are Cauchy and Eigen points, as in Definitions 4 and 5, respectively.

It is not hard to see that, compared with Condition 4, when the gradient is large enough, Condition 6 involves a more accurate solution of (2.9) than a simple Cauchy Point. For a given $p \ll d$, let $\mathbf{U}_t \in \mathbb{R}^{d\times p}$ be any orthonormal basis for some p-dimensional sub-space $\mathcal{S}$ such that $\text{Span}\{\mathbf{s}_t^C\} \subseteq \mathcal{S} \subset \mathbb{R}^d$. Such a sub-space can be easily constructed from $\mathbf{s}_t^C$ and $\mathbf{H}_t$ using standard methods such as Lanczos process [8, Section 7.5]. Now, a practical way to ensure Condition 6 for the case where $\|\mathbf{g}_t\| \geq \epsilon_g$, is by approximating the unconstrained high-dimensional sub-problem (2.9) with the following lower-dimensional problem

$$\min_{\mathbf{v}\in\mathbb{R}^p} \langle \mathbf{U}_t\mathbf{v}, \mathbf{g}_t\rangle + \frac{1}{2}\langle \mathbf{U}_t\mathbf{v}, \mathbf{H}_t\mathbf{U}_t\mathbf{v}\rangle + \frac{\sigma_t}{3}\|\mathbf{U}_t\mathbf{v}\|^3,$$

followed by setting $\mathbf{s}_t = \mathbf{U}_t\mathbf{v}$. Obviously, when $p \ll d$, solving such lower-dimensional problem, which involves smaller matrix and vectors, can be significantly easier than the original high-dimensional one. One can consider a sequence of such reduced sub-problems using progressively larger sub-spaces and stop when (2.13) holds. Since ultimately $\|\nabla m(\mathbf{s}_t)\| = 0$ for when $\mathcal{S} = \mathbb{R}^d$, we are guaranteed to also satisfy (2.13) for large enough $\mathcal{S} \subset \mathbb{R}^d$.

The optimal iteration complexity of Algorithm 2 is stated in Theorem 3.

Theorem 3 (Optimal Complexity of Algorithm 2)**.** *Let Assumption 1 hold and consider any* $0 < \epsilon_g, \epsilon_H < 1$. *Further, suppose that* $\mathbf{g}_t$ *and* $\mathbf{H}_t$ *satisfy Assumption 2 with* δ_g *and* δ_H *under Condition 5. If the approximate solution to the sub-problem* (2.9) *satisfies Condition 6, then Algorithm 2 terminates after at most*

$$T \in \mathcal{O}\left(\max\{\epsilon_g^{-1.5}, \epsilon_H^{-3}\}\right),$$

iterations.

Remark 2. *If we assume* L_F *is known (set* $\sigma_t \equiv L_F$*) and* $\boldsymbol{s}_t$ *is close enough to the best solution* $\boldsymbol{s}_t^*$ *of* $m_t(\boldsymbol{s})$*, by using Taylor expansion, it is not hard to show that*

$$F(\boldsymbol{x}_t + \boldsymbol{s}_t) - F(\boldsymbol{x}_t) \geq -c_1 m_t(\boldsymbol{s}_t) \geq -c_2 m_t(\boldsymbol{s}_t^*).$$

Given $\|\mathbf{g}_t\|$ *or* $-\lambda_{\min}(\mathbf{H}_t)$ *is large,* $-m(\boldsymbol{s}_t^*)$ *would then be large. Therefore, there could be enough descent along* $\boldsymbol{s}_t$*. Roughly speaking, we could drop Lemmas 10 to 16, and get the same iteration complexity results, i.e.* $T \in \mathcal{O}(\max\{\epsilon_g^{-1.5}, \epsilon_H^{-3}\})$*. For example, we do not need Lemma 10 to show Cauchy Point is one of the directions for* $-m_t(\boldsymbol{s}_t)$*. Also, in this case, either Lemma 18 or Lemma 19 becomes redundant.*

2.2.3 Finite-Sum Problems

As a special class of (2.1), we now consider non-convex finite-sum minimization of (2.2), where each $f_i : \mathbb{R}^d \to \mathbb{R}$ is smooth and non-convex. In big-data regimes where $n \gg 1$, one can consider sub-sampling schemes to speed up various aspects of many Newton-type methods, e.g., see [196, 247, 23] for such techniques in the context of convex optimization. More specifically, we consider the sub-sampled gradient and Hessian as

$$\mathbf{g} \triangleq \frac{1}{|\mathcal{S}_g|}\sum_{i\in\mathcal{S}_g} \nabla f_i(\mathbf{x}), \quad \text{and} \quad \mathbf{H} \triangleq \frac{1}{|\mathcal{S}_H|}\sum_{i\in\mathcal{S}_H} \nabla^2 f_i(\mathbf{x}), \tag{2.14}$$

where $\mathcal{S}_g, \mathcal{S}_H \subset \{1, \cdots, n\}$ are the sub-sample batches for the estimates of the gradient and Hessian, respectively. In this setting, a relevant question is that of "how large sample sizes $\mathcal{S}_g$ and $\mathcal{S}_H$ should be to guarantee, at least with high probability, that $\mathbf{g}$ and $\mathbf{H}$ in (2.14) satisfy Assumption 2". As long as $|\mathcal{S}_g| \ll n$ and $|\mathcal{S}_H| \ll n$, such sub-sampling strategies can result in significant reduction in overall computational costs.

If sampling is done uniformly at random, we have the following sampling complexity bounds, whose proofs can be found in [196, 246]. For more sophisticated sampling/sketching schemes, see [184, 247, 246].

Lemma 1 (Sampling Complexity [196, 246])**.** *For any $0 < \delta_g, \delta_H, \delta < 1$, let $\mathbf{g}$ and $\mathbf{H}$ be as in (2.14) with*

$$|\mathcal{S}_g| \geq \frac{16K_g^2}{\delta_g^2} \log \frac{1}{\delta} \quad \textit{and} \quad |\mathcal{S}_H| \geq \frac{16K_H^2}{\delta_H^2} \log \frac{2d}{\delta},$$

where $0 < K_g, K_H < \infty$ are such that $\|\nabla f_i(\boldsymbol{x})\| \leq K_g$ and $\|\nabla^2 f_i(\boldsymbol{x})\| \leq K_H$. Then, with probability at least $1 - \delta$, Assumption 2 holds with the corresponding δ_g and δ_H.

Combining Lemma 1 with the sufficient conditions presented earlier, i.e., Conditions 1 and 2 for Algorithm 1 and Conditions 3 and 4 or Conditions 5 and 6 for Algorithm 2, we can immediately obtain, similar, but probabilistic, iteration complexities as in Sections 2.2.1 and 2.2.2. For completeness, we bring such a result for Algorithm 1 and omit those related to Algorithm 2.

Since Conditions 1, 3 and 5 are only guaranteed probabilistically, in order to guarantee success, a small failure probability across all iterations is required. In particular, in order to get an accumulative success probability of $1 - \delta$ for the entire T iterations, the per-iteration failure probability is set as $(1 - \sqrt[T]{(1 - \delta)}) \in \mathcal{O}(\delta/T)$. Fortunately, this failure probability appears only in the "log factor" in Lemma 1 and so it is not the dominating cost. For example, for $T \in \mathcal{O}(\max\{\epsilon_g^{-2}\epsilon_H^{-1}, \epsilon_H^{-3}\})$, as in Theorem 1, we can set the per-iteration failure probability to $\delta \min\{\epsilon_g^2\epsilon_H, \epsilon_H^3\}$.

Corollary 1 (Optimal Complexity of Algorithm 1 For Finite-Sum Problem (2.2))**.** *Consider any $0 < \epsilon_g, \epsilon_H, \delta < 1$. Let δ_g and δ_H be as in Condition 1 and set $\delta_0 = \delta \min\{\epsilon_g^2\epsilon_H, \epsilon_H^3\}$. Furthermore, for such δ_g, δ_H and δ_0, let the sample-size $|\mathcal{S}_g|$ and $|\mathcal{S}_H|$ be as in Lemma 1 and form the sub-sampled gradient and Hessian as in $\mathbf{H}$ as in (2.14). For Problem (2.2), under Assumptions 1 and 2 and Conditions 1 and 2, Algorithm 1 terminates in at most $T \in \mathcal{O}(\max\{\epsilon_g^{-2}\epsilon_H^{-1}, \epsilon_H^{-3}\})$ iterations, upon which, with probability $1 - \delta$, we have that $\|\nabla F(\boldsymbol{x})\| \leq \epsilon_g + \delta_g$, and $\lambda_{\min}(\nabla^2 F(\boldsymbol{x})) \geq -(\delta_H + \epsilon_H)$.*

2.3 Experiments

In this section, we provide empirical results evaluating the performance of Algorithms 1 and 2. We aim to demonstrate that approximate gradient, approximate Hessian and approximate sub-problem solves indeed help improve the computational efficiency. For our experiments, we consider the following methods:

- *F*ull ARC: Standard ARC algorithms with exact gradient and Hessian.
- *S*ubH TR/ARC [246]: TR and ARC with exact gradient and sub-sampled Hessian.
- *S*CR (GD) [229]: CR with sub-sampled gradient and Hessian. The sub-problems are solved by gradient descent (GD) [37].

• *S*CR (Lanczos): CR which is similar to SCR (GD) [229] but the sub-problems are solved by generalized Lanczos method [39].

• *S*GD: Stochastic gradient descent with momentum [223]. The momentum parameter is set to the typical value of 0.9. The gradient size is set to be 1000.

• *A*dagrad: An adaptive first-order method developed in [73]. The gradient size is set to be 1000.

• *A*dam: A modification of Adagrad which has become the method of choice within the machine learning community [125]. The two momentum terms in ADAM are set to be 0.9 and 0.999, which are typically chosen in practice. The gradient size is set to be 1000.

• *I*nexact TR/ARC (**this work**): TR and ARC with sub-sampled gradient and Hessian as described in Algorithms 1 and 2. The sub-problems of Algorithms 1 and 2 are solved, respectively, by CG-Steihaug [221], and by generalized Lanczos method [39]. For both algorithms, the gradient sample size is adaptively chosen as follows: if $\|g_t\| \geq 1.2\|g_{t-1}\|$ or $\|g_t\| \leq \|g_{t-1}\|/1.2$, we respectively decrease or increase the sample size for gradient estimation by a factor of 1.2. Otherwise, the sample size stays the same as the previous iteration.

For our experiments, except SCR (GD), we use CG-Steihaug method [171] and generalized Lanczos method [39] to solve the sub-problems of TR and ARC, respectively. Also following [243], we set the maximum iterations for the sub-problem solvers to 250. Further specific hyper-parameters as well as samples sizes used for second-order algorithms in our experiments are gathered in Table 2.2.

Method	Full ARC	SubH TR	SubH ARC	SCR	Algorithm 1	Algorithm 2
Hyper-parameter	$\sigma_0 = 10$	$\Delta_0 = 10$	$\sigma_0 = 10$	$\sigma = 10$ (Figure 2.2)	$\Delta_0 = 10$	$\sigma_0 = 10$
$\|\mathcal{S}_g\|$ (*Section* 2.3.1)	n	n	n	N/A	5,000	5,000
$\|\mathcal{S}_H\|$ (*Section* 2.3.1)	n	1,000	1,000	N/A	1,000	1,000
$\|\mathcal{S}_g\|$ (*Section* 2.3.2)	n	n	n	$0.1n$	$0.1n$	$0.1n$
$\|\mathcal{S}_H\|$ (*Section* 2.3.2)	n	$0.01n$	$0.01n$	$0.01n$	$0.01n$	$0.01n$

Table 2.2: The hyper-parameters and the samples sizes used for Newton-type methods used in the experiments. n is as in Tables 2.3 and 2.4. Recall that $|\mathcal{S}_g|$ is adjusted adaptively for Algorithms 1 and 2, and hence the values given here refer to the gradient sample size at initialization. For Hessian estimation, however, We use fixed sample size for both Algorithms 1 and 2.

Similar to [243], the performance of all the algorithms in our experiments is measured by tallying total *number of propagations*, i.e., number of oracle calls of function, gradient and Hessian-vector products. More specifically, for each i in (2.2), after computing $f_i(\mathbf{x})$, computing $\nabla f_i(\mathbf{x})$ is equivalent to one additional function evaluation. In our implementations, we merely require Hessian-vector products $\nabla^2 f_i(\mathbf{x})\mathbf{v}$, instead of forming the explicit Hessian, which amounts to two additional function evaluations, as compared with gradient evaluation. We would like note that we opted to choose propagations as the complexity since "wall-clock" time can be highly affected by particular implementation details as well as system specifications. In contrast, counting the number of propagations (or oracle calls) is implementation and system independent and is hence more appropriate and fair. For experiments of Section 2.3.1, we use GTX Titan X GPU with 12GB RAM memory. The code is based on Python with framework PyTorch 1.2.0. In Section 2.3.2, the experiments are performed on a Macbook Pro, 2017c(2.9GHz Intel Core i7-7820HQ, 16 GB RAM) with Matlab. Our code is publicly available at `https://github.com/yaozhewei/Inexact_Newton_Method`.

2.3.1 Multi-layer Perceptron

We first evaluate the performance of Algorithm 1 in terms of running time, as measured by the training loss versus total number of propagations. We do this using a simple multi-layer perceptron (MLP) model on MNIST dataset, which is available from `LIBSVM` library [45].

Hidden Layer Size	n	d
16	60,000	12,704
128	60,000	101,632
1,024	60,000	813,056

Table 2.3: The dimension of the parameter space for various hidden layer sizes in MLP experiment.

Here, we consider an MLP involving one hidden layer and one output layer to determine the assigned the class of the input image. All intermediate neurons involve SoftPlus activation function [85], which amounts to a smooth optimization problem. We consider three instances of such an MLP with hidden layer sizes of 16, 128, and 1,024. Table 2.3 gathers the dimensions of the resulting optimization problems.

Similar to the observations in [243], despite the best of our efforts, we were unable to obtain the expected performance of any variant of ARC and CR on this model problem using a variety of implementations. As a result, we did not include them in this experiment.

For all first-order methods, fixed step sizes in the range $\alpha = [0.0001, 0.001, 0.01, 0.1]$ are tested. As is clearly observed here, and also is reported in the similar literature [131, 243, 18], the performance of first-order methods strongly depends on the particular choice of their main hyper-parameter, i.e., the step-size. For example, in Figure 2.1 (g)-(i), a finely-tuned ADAM can have superior performance. However, if the step-size is not chosen appropriately, the performance of ADAM could be unpleasantly erratic. As it can also be seen, even the best performing step-size for ADAM ceases to be appropriate at later stages of the algorithm.

As a result, to obtain a solution with higher accuracy, one needs to pick a new step-size at later stages of the algorithm, as otherwise ADAM ultimately diverges or exhibits violent zigzagging behavior.

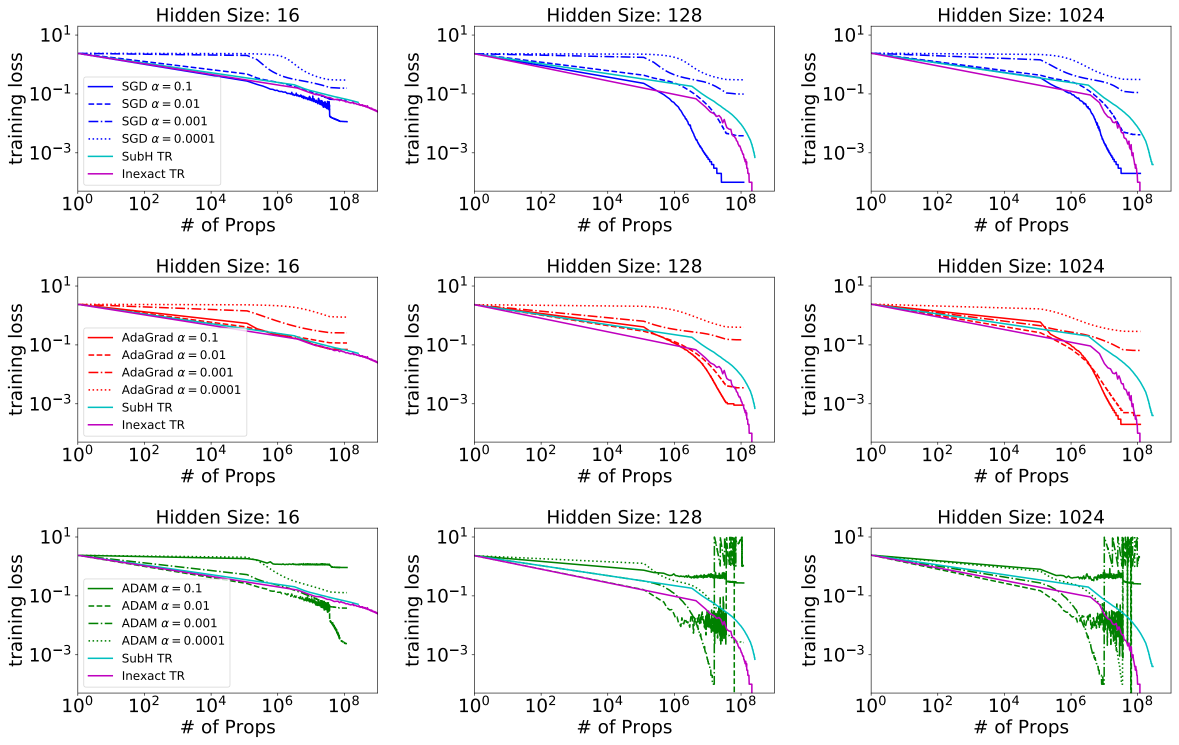

Figure 2.1: Results of variants of TR (SubH TR and Inexact TR) and first-order methods (SGD, AdaGrad, and ADAM) on MLP with different hidden size (16, 128, and 1024). Both x-axis and y-axis are drawn using the logarithmic scaling.

2.3.2 Non-linear Least Squares

Since, we did not manage to obtain a reasonable performance using any variants of ARC and CR, we opted to exclude them from the previous experiments in Section 2.3.1. Nonetheless, on a simpler non-linear least squares problem, we were able to compare and contrast various properties of these methods, which we include in this section.

Computational Efficiency (Figure 2.2): We now consider the running time of Algorithm 2 in the context of simple, yet illustrative, nonlinear least squares arising from the task of binary classification with squared loss[1]. Specifically, given training data $\{\mathbf{a}_i, b_i\}_{i=1}^n$, where $\mathbf{a}_i \in \mathbb{R}^d, b_i \in \{0,1\}$, consider the following empirical risk minimization problem

$$\min_{\mathbf{x}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n}\Big(b_i - \phi\big(\langle \mathbf{a}_i, \mathbf{x}\rangle\big)\Big)^2,$$

where $\phi(z)$ is the sigmoid function, i.e. $\phi(z) = 1/(1+e^{-z})$. Datasets are taken from `LIBSVM` library [45]; see Table 2.4. We use the same setup in [243].

Table 2.4: Datasets used for experiments with non-linear least squares.

Data	n	d
`covertype`	464,810	54
`ijcnn1`	49,990	22

Figure 2.2 depicts the results. For all variants of SCR, we hand-tuned the algorithm by performing an exhaustive grid-search over the involving hyper-parameters, and we show the best results. For all variants ARC, we choose the same initial parameters, σ_0. We can observe that all methods achieve similar training errors, while Algorithm 2 does so with much fewer number of propagation calls, as compared with other methods. Furthermore, all variants of ARC perform similarly, or better, than all variants of CR. This is an empirical evidence that the "optimal" worst-case analysis of CR, while theoretically interesting, might not translate to many practical applications of interest.

Robustness to Hyper-parameters (Figure 2.3): Next, we highlight the practical challenges arising with algorithms that heavily rely on the knowledge of hard-to-estimate parameters. In particular, we aim here to demonstrate that an algorithm whose performance is greatly affected by specific settings of parameters that cannot be easily estimated, lacks the versatility needed in many practical applications. To do so, we perform one such demonstration by focusing on sensitivity/robustness of Algorithm 2 and SCR to the cubic regularization parameter σ. The results are gathered in Figure 2.3. One can see that the performance of SCR is highly dependent on the choice of its main hyper-parameter, i.e., σ. Indeed, if σ is not chosen appropriately, SCR either converges very slowly or does not converge at all. Determining an appropriate value of σ requires an expensive (in human time or CPU time) hyper-parameter search. This is in sharp contrast with Algorithm 2 which shows great robustness to the choice of σ_0 and works more-or-less "out of the box."

[1]Since logistic loss, which is the "standard" loss used in this task, leads to a convex problem, we use square loss to obtain a non-convex objective.

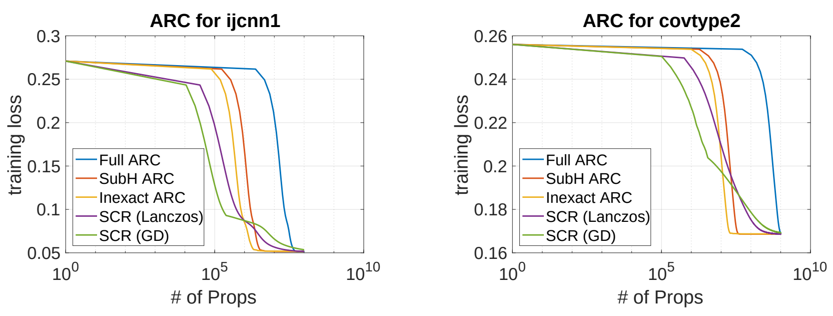

Figure 2.2: Performance of variants of ARC and CR methods on `ijcnn1` and `covertype` for binary linear classification. The x-axis is drawn on the logarithmic scale.

2.3.3 Summary of Numerical Experiments

From the above numerical examples, i.e., multi-layer perceptron in Section 2.3.1 and non-linear least squares in Section 2.3.2, we can make the following general observations regarding the overall performance of Algorithms 1 and 2.

i. Within the context of both inexact TR and ARC, we can clearly see the added efficiency obtained from sub-sampling both the gradient and the Hessian. This is illustrated by competitive performance compared with several first-order methods as well as superior performance relative to more expensive variants, i.e., exact algorithms and those where only the Hessian is approximated as in [246].

ii. In terms of tuning the respective underlying hyper-parameters, our inexact ARC variant is significantly more robust compared to SCR [229]. Similarly, in contrast to first-order algorithms whose performance is greatly affected by the choice of their main hyper-parameter, i.e., step-size, the performance of the proposed Newton-type methods exhibits significant resilience to particular choices of their hyper-parameters.

2.4 Conclusions and Further Thoughts

In this paper, we considered inexact variants of trust region and adaptive cubic regularization in which, to increase efficiency, the gradient and Hessian, as well as the solution to the

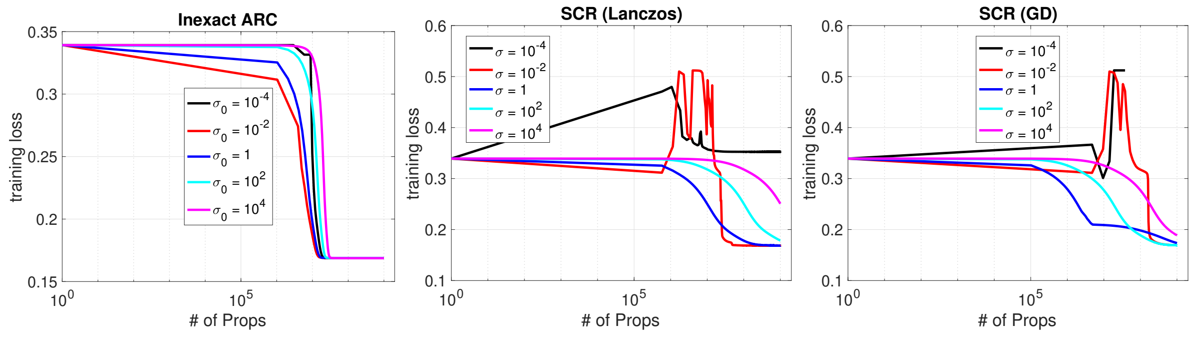

Figure 2.3: Robustness of Algorithm 2 and sensitivity of SCR w.r.t. the cubic regularization parameter on `covertype` dataset. For Algorithm 2, this parameter, initially set to σ_0, adaptively changes across iterations; while for SCR, it is kept fixed at a certain σ for all iterations. (a) Robustness of Algorithm 2 to the choice of σ_0, where σ_0 varies over several orders of magnitude. (b)–(c) Sensitivity of SCR with two different sub-problem solvers (Lanczos and GD) and several choices of the fixed cubic regularization σ. For SCR (GD), the step size of GD for solving the sub-problem is hand-tuned to obtain the best performance (which can be extremely expensive).

underlying sub-problems are all suitably approximated. We showed that under certain conditions on these approximation, to coverage to second-order criticality, the inexact variants achieve the same optimal iteration complexity as the exact counterparts. The advantages, and perhaps shortcomings, of our algorithms were also numerically demonstrated.

We note that unlike Conditions 2, 4 and 6, ensuring Conditions 1, 3 and 5 is not generally as straightforward and remains the main practical challenge in our work and, to our knowledge, all of related literature. Although, deterministic approaches such as finite-difference schemes can theoretically guarantee these conditions, obtaining an appropriate discretization scheme relies on the knowledge of problems-dependent constants that are typically hard to estimate. Similarly, for the case of finite-sum minimization in Section 2.2.3, which is an important driving application for our results here, randomized sub-sampling techniques can give sufficient sample sizes to guarantee such conditions. However, this also requires estimates of the constants L_F, K_H, and K_g. Fortunately, for several problems in machine learning obtaining such estimates is in fact straightforward, e.g., linear predictor models in [246, Table 1] and [196, Table 2] as well as deep learning in [78]. Furthermore, in our experience as well as that of many others, the performance of such sub-sampled algorithms is most resilient to under-sampling. This is in sharp contrast, however, to the quality of the sub-problem solutions, which significantly affect the overall performance of the algorithms.

As a by-product of our analysis, the bound on gradient approximations for obtaining the optimal iteration complexity of inexact ARC remains very pessimistic, and tightening such a bound is left for future work. An important missing piece from our work here is incorporating function approximations as a way to further reduce the computational costs, which we are currently pursuing. Finally, our results here only consider iteration complexities of the proposed algorithms. A much finer grained analysis is required to obtain overall running time, which is an important avenue for future work.

Chapter 3

Inexact Newton-CG Algorithms with Complexity Guarantees

3.1 Introduction

We consider the following unconstrained optimization problem

$$\min_{\mathbf{x}\in\mathbf{R}^d} f(\mathbf{x}), \tag{3.1}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is a smooth but nonconvex function. At the heart of many machine learning and scientific computing applications lies the problem of finding an (approximate) minimizer of Problem (3.1). Faced with modern "big data" problems, many classical optimization algorithms [171, 19] are inefficient in terms of memory and/or computational overhead. Much recent research has focused on approximating various aspects of these algorithms. For example, efficient variants of first-order algorithms, such as the stochastic gradient method, make use of inexact approximations of the gradient. The defining element of second-order algorithms is the use of the curvature information from the Hessian matrix. In these methods, the main computational bottleneck lies with evaluating the Hessian, or at least being able to perform matrix-vector products involving the Hessian. Evaluation of the gradient may continue to be an unacceptably expensive operation in second-order algorithms too. Hence, in adapting second-order algorithms to machine learning and scientific computing applications, we seek to approximate the computations involving the Hessian and the gradient, while preserving much of the convergence behavior of the exact underlying second-order algorithm.

Second-order methods use curvature information to nonuniformly rescale the gradient in a way that often makes if a more "useful" search direction, in the sense of providing a greater decrease in function value. Superior convergence properties can be obtained by comparison with first-order methods; for example, the classical Newton's method has a locally quadratic convergence rate. Second-order information also opens the possibility of convergence to points that satisfy second-order necessary conditions for optimality, that is, $\mathbf{x}$ for which $\|\nabla f(\mathbf{x})\| = 0$ and $\nabla^2 f(\mathbf{x}) \succeq \mathbf{0}$. (By contrast, first-order methods typically guarantee convergence only to first-order stationary points, for which $\|\nabla f(\mathbf{x})\| = 0$, although first-order methods that make use of random perturbations can guarantee approximate second-order optimality; see for example [123].) For nonconvex machine learning problems, first-order stationary points include saddle points, which are undesirable for obtaining good generalization performance [62, 56, 207, 137].

The canonical example of second-order methods is the classical Newton's method, which in its pure form is often written as

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k, \quad \text{where } \mathbf{d}_k = -\mathbf{H}_k^{-1}\mathbf{g}_k,$$

where $\mathbf{H}_k = \nabla^2 f(\mathbf{x}_k)$ is the Hessian, $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$ is the gradient, and α_k is some appropriate step-size, often chosen using an Armijo-type line-search [171, Chapter 3]. A more practical variant for large-scale problems is Newton-Conjugate-Gradient (Newton-CG), in which the linear system $\mathbf{H}_k\mathbf{d}_k = -\mathbf{g}_k$ is solved inexactly using the conjugate gradient (CG) algorithm [221]. Such an approach requires access to the Hessian matrix only via matrix-vector products; it does not require $\mathbf{H}_k$ to be evaluated explicitly.

Recently, a new variant of the Newton-CG algorithm was proposed in [199] that can be applied to large-scale non-convex problems. This algorithm comes equipped with certain safeguards and enhancements that allow derivation of worst-case complexity results in terms of number of iterations and total running time. However, this approach relies on exact evaluation of the gradient and on matrix-vector multiplication involving the exact Hessian at each iteration. Such operations can be prohibitively expensive in machine learning problems. For example, when the underlying optimization problem has the finite-sum form

$$\min_{\mathbf{x}\in\mathbf{R}^d} f(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x}), \tag{3.2}$$

exact computation of the Hessian/gradient can be extremely costly when $n \gg 1$, requiring a complete pass through the training data set. Our work here builds upon that of [199] to allow for the inexact computation of the gradient and Hessian, while preserving the complexity result of [199].

3.1.1 Related Work

Since deep learning became ubiquitous, first-order methods such as gradient descent and its adaptive, stochastic variants [125, 73], have become the most popular class of optimization algorithms in machine learning; see the recent textbooks [15, 132, 143, 240] for in-depth treatments. These methods are easy to implement, and their per-iteration cost is low compared to second-order alternatives. Although classical theory for first-order methods guarantees convergence only to first-order optimal (stationary) points, [81, 123, 139] argued that stochastic variants of certain first-order methods such as SGD have the potential of escaping saddle points and converging to the second-order stationary point. Effectiveness of such methods usually requires painstaking fine-tuning of their (often many) hyperparameters, and the number of iterations they require to escape saddle regions can be large. By contrast, second-order methods can make use of curvature information (via the Hessian) to escape saddle points efficiently and ultimately converge to the second-order stationary points. This behavior is seen in trust-region region methods [57, 60, 59], cubic regularization [169] and its adaptive variants (ARC) [39, 40], as well as line-search based second-order methods [200, 199].

Subsequent to [39, 40, 41], which were among the first works to study Hessian approximations to ARC and trust region algorithms, respectively, [246] analyzed the optimal complexity of both trust region and cubic regularization, in which the Hessian matrix is approximated under milder conditions. Extension to gradient approximations was then studied in [229, 252]. The analysis in [93, 44, 21] relies on probabilistic models whose quality are ensured with a certain probability, but which allow for approximate evaluation of the objective function as well.

A notable difficulty of these methods concerns solution of their respective subproblems, which can themselves be nontrivial nonconvex optimization problems. An exception is [199], whose fundamental operations are linear algebra computations, which are much better understood. This paper enhances the classical Newton-CG approach with safeguards to detect negative curvature in the Hessian, during solution of the Newton equations to obtain the step $\mathbf{d}_k$. Negative curvature directions can subsequently be exploited by the algorithm to make significant progress in reducing the objective. Moreover, [199] gives complexity guarantees that have been shown to be optimal in certain settings. (Henceforth, we use the term "Newton-CG" to refer specifically to the algorithm in [199].)

3.1.2 Contribution

We describe variants of the Newton-CG algorithm in [199] in which both the gradient and the Hessian are approximated, while maintaining the convergence and complexity properties of the original algorithm. More specifically, to achieve $(\epsilon_g, \sqrt{\epsilon_g})$-Optimality (see Definition 6 below) under Condition 7 on gradient and Hessian approximations (see below, in Section 3.2.2), we show the following.

- Inexact Newton-CG with backtracking line-search (Algorithm 3), achieves the optimal iteration complexity of $\mathcal{O}(\epsilon_g^{-3/2})$; see Section 3.2.2.

- Inexact Newton-CG in which a pre-defined step size replaces the backtracking line searches (Algorithm 4) achieves the same optimal iteration complexity of $\mathcal{O}(\epsilon_g^{-3/2})$; see Section 3.2.3.

- The accuracy required in our gradient approximation changes adaptively with the current gradient size. One consequence of this feature is to allow cruder gradient approximations in the regions with larger gradients, translating to a more efficient algorithm overall.

• We empirically illustrate the advantages of our methods on several real datasets; see Section 3.3.

3.2 Algorithms and Theoretical Analysis

We describe our algorithms and present our main theoretical results in this section. We start with background (Section 3.2.1), then proceed to our two main algorithms (Section 3.2.2 and Section 3.2.3).

3.2.1 Notation and Assumption

Throughout this paper, scalar constants are denoted by regular lower-case and upper-case letters, e.g., c and K. We use bold lowercase and blackboard bold uppercase letters to denote vectors and matrices, e.g., $\mathbf{a}$ and $\boldsymbol{A}$, respectively. The transpose of a real vector $\mathbf{a}$ is denoted by $\mathbf{a}^T$. For a vector $\mathbf{a}$, and a matrix $\boldsymbol{A}$, $\|\mathbf{a}\|$ and $\|\boldsymbol{A}\|$ denote the vector ℓ_2 norm and the matrix spectral norm, respectively. Subscripts (as in $\mathbf{a}_t$) denote iteration counters. The smallest eigenvalue of a symmetric matrix $\boldsymbol{A}$ is denoted by $\lambda_{\min}(\boldsymbol{A})$.

For nonconvex problems, determination of near-optimality can be much more complicated than for convex problems; see the examples of [165, 106]. In this paper, as in earlier works (see for example [199]), we make use of approximate second-order optimality, defined as follows.

Definition 6 ((ϵ_g, ϵ_H)-optimality)**.** *Given $0 < \epsilon_g, \epsilon_H < 1$, $\boldsymbol{x}$ is an (ϵ_g, ϵ_H)-optimal solution of (3.1), if*

$$\|\nabla f(\boldsymbol{x})\| \leq \epsilon_g \quad \text{and} \quad \lambda_{\min}(\nabla^2 f(\boldsymbol{x})) \geq -\epsilon_H. \tag{3.3}$$

Assumption 3. *The smooth nonconvex function f is bounded below by the finite value f_{low}. Moreover, on an open set in $\boldsymbol{R}^n$ containing all line segments $[\boldsymbol{x}_k, \boldsymbol{x}_k + \mathbf{d}_k]$ for iterates $\boldsymbol{x}_k$ and search directions $\mathbf{d}_k$ generated by our algorithms, the objective function has Lipschitz continuous gradient and Hessian, that is, there are positive constants $0 < L_g < \infty$ and $0 < L_H < \infty$ such that*

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\| \leq L_g \|\boldsymbol{x} - \boldsymbol{y}\| \quad \text{and} \quad \left\|\nabla^2 f(\boldsymbol{x}) - \nabla^2 f(\boldsymbol{y})\right\| \leq L_H \|\boldsymbol{x} - \boldsymbol{y}\|.$$

For our inexact Newton-CG algorithms, we also require that the approximate gradient and Hessian satisfy the following conditions, for prescribed positive values $\delta_{g,t}$ and δ_H.

Definition 7. *For given $\delta_{g,t}$ and δ_H, we say that the approximate gradient $\mathbf{g}_t$ and Hessian $\mathbf{H}_t$ at iteration t are $\delta_{g,t}$-accurate and δ_H-accurate if*

$$\|\mathbf{g}_t - \nabla f(\boldsymbol{x}_t)\| \leq \delta_{g,t} \quad \text{and} \quad \|\mathbf{H}_t - \nabla^2 f(\boldsymbol{x}_t)\| \leq \delta_H, \tag{3.4}$$

respectively.

Under these assumptions and conditions, it is easy to show that there exist constants U_g and U_H such that the following are satisfied for all iterates $\mathbf{x}_t$ in the set defined in Assumption 3:

$$\|\mathbf{g}_t\| \leq U_g \quad \text{and} \quad \|\mathbf{H}_t\| \leq U_H. \tag{3.5}$$

3.2.2 Results with Line Search

Here, we present our inexact damped Newton-CG algorithm (Algorithm 3); and, using Procedures 8 and 9, we establish worst case iteration complexity results to achieve (ϵ_g, ϵ_H)-optimality (Definition 6). More specifically, we will show that under mild conditions on the approximate gradient and Hessian, the worst case complexity of our inexact algorithm is the same as the original exact counterpart, given in [199].

Our inexact damped Newton-CG method is depicted in Algorithm 3, where the major difference between our method and the exact counterpart in [199] is using the approximations of gradient and Hessian. Another notable difference (highlighted in blue) is that our algorithm directly calls Procedure 9 to check if there is a sufficiently large negative curvature direction when the direction $\mathbf{d}_k$ obtained from Procedure 8 is small, specifically $\|\mathbf{d}_k\| \leq \sqrt{\epsilon_g/L_{\mathbf{H}}}$. If there is no large negative curvature, we terminate and return the point $\mathbf{x}_k + \mathbf{d}_k$, which already satisfies the second-order optimality condition. Otherwise, a backtracking line search will be performed along the negative curvature direction detected by Procedure 9. In theory, this modification is critical to obtaining the optimal worst case complexity. In practice, however, we have never observed the need to enforce this step, even when the norm of update direction, $\mathbf{d}_k$, from Procedure 8 is very small.

In order to establish the iteration complexity of Algorithm 3, we first present a sufficient condition on the degree of the inexactness of the gradient and Hessian.

Condition 7. *To make the output of Algorithm 3 satisfy Definition 6, we require the inexact gradient $\mathbf{g}_k$ and Hessian $\mathbf{H}_k$ to satisfy Definition 7 with*

$$\delta_{g,k} \leq \frac{1-\zeta}{8} \max\left(\epsilon_g, \min(\epsilon_H \|\mathbf{d}_k\|, \|\mathbf{g}_k\|, \|\mathbf{g}_{k+1}\|)\right), \quad \textit{and} \quad \delta_H \leq \frac{1-\zeta}{2}\epsilon_H.$$

Obviously, one can simplify Condition 7 to have iteration-independent δ_g as

$$\delta_g \leq \frac{(1-\zeta)\epsilon_g}{8}.$$

However, the adaptively of the iteration-dependent version of Condition 7 through $\mathbf{g}_k$ and $\mathbf{g}_{k+1}$ offers practical advantages. Indeed, in many iterations, one can expect $\|\mathbf{g}_k\|$ and $\|\mathbf{g}_{k+1}\|$ to be of similar magnitudes. Also, as shown in Lemma 27, we have $\|\mathbf{d}_k\| \leq \epsilon_H^{-1}\sqrt{1+\zeta^2/4}\|\mathbf{g}_k\|$. Thus, the three terms in $\min(\epsilon_H\|\mathbf{d}_k\|, \|\mathbf{g}_k\|, \|\mathbf{g}_{k+1}\|)$ are often roughly of the same order, and usually larger than ϵ_g. These observations suggest that when the true gradient is large, we can employ loose approximations.

Now, combining Lemmas 29–32, we have the final iteration complexity in Theorem 4.

Algorithm 3 Inexact Damped Newton-CG with Line Search

```
1: Inputs: $\epsilon_g, \epsilon_H > 0$; Backtracking parameter $\theta \in (0,1)$; Starting point $\mathbf{x}_0$; upper bound
   on Hessian norm $U_H > 0$; accuracy parameter $\zeta \in (0, \min\{1, U_H\})$;
2: for $k = 0, 1, 2, \cdots$ do
3:   if $\|\mathbf{g}_k\| \geq \epsilon_g$ then
4:     Call Procedure 8 with $\mathbf{H} = \mathbf{H}_k, M = U_H, \epsilon = \epsilon_H, \mathbf{g} = \mathbf{g}_k$ and accuracy parameter $\zeta$
       to obtain $\mathbf{d}$ and $d_{\text{type}}$;
5:     if $d_{\text{type}} == NC$ then
6:       $\mathbf{d}_k \leftarrow -\text{sgn}(\mathbf{d}^T \nabla f(\mathbf{x}_k)) \frac{|\mathbf{d}^T \mathbf{H}_k \mathbf{d}|}{\|\mathbf{d}\|^2} \frac{\mathbf{d}}{\|\mathbf{d}\|}$;
7:     else
8:       $\mathbf{d}_k \leftarrow \mathbf{d}$;
9:     end if
10:    if $\|\mathbf{d}_k\| \leq \sqrt{\frac{\epsilon_g}{L_H}}$ then
11:      Call Procedure 9 with $\mathbf{H} = \mathbf{H}_k, M = U_H, \epsilon = \epsilon_H$ to obtain $\mathbf{v}$ and $\lambda_{min}(\mathbf{H}_k)$;
12:      if Procedure 9 certificates $\lambda_{\min}(\mathbf{H}_k) \geq -\epsilon_H$ then
13:        $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \mathbf{d}_k$ and terminate;
14:      else
15:        $\mathbf{d}_k \leftarrow -\text{sgn}(\mathbf{v}^T \nabla f(\mathbf{x}_k)) \frac{|\mathbf{v}^T \mathbf{H}_k \mathbf{v}|}{\|\mathbf{v}\|^2} \mathbf{v}$ and go to Line-Search;
16:      end if
17:    else
18:      Go to Line-Search;
19:    end if
20:  else
21:    Call Procedure 9 with $\mathbf{H} = \mathbf{H}_k, M = U_H, \epsilon = \epsilon_H$;
22:    if Procedure 9 certificates $\lambda_{\min}(\mathbf{H}_k) \geq -\epsilon$ then
23:      Terminate;
24:    else
25:      $\mathbf{d}_k \leftarrow -\text{sgn}(\mathbf{v}^T \nabla f(\mathbf{x}_k)) \frac{|\mathbf{v}^T \mathbf{H}_k \mathbf{v}|}{\|\mathbf{v}\|^2} \mathbf{v}$ and go to Line-Search;
26:    end if
27:  end if
28:  Line-Search: Compute a step length $\alpha_k = \theta^{j_k}$, where $j_k$ is the smallest nonnegative
     integer such that
         $$f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) < f(\mathbf{x}_k) - \frac{\eta}{6} \alpha_k^3 \|\mathbf{d}_k\|^3; \quad (3.6)$$
29:  $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{d}_k$;
30: end for
```

Theorem 4. *Let Assumption 3 and Condition 7 hold. Let $\epsilon_H = \sqrt{L_H \epsilon}, \epsilon_g = \epsilon$. Define*

$$\bar{K} := \left\lceil \frac{3(f(\boldsymbol{x}_0) - f_{low})}{\min\left(c_{\text{sol}}/(8L_H^{3/2}), c_{\text{sol}}L_H^{3/2}, c_{\text{sol}}, c_{\text{nc}}L_H^{3/2}/8\right)} \epsilon^{-3/2} \right\rceil + 2, \tag{3.7}$$

where c_{sol} and c_{nc} are defined in Lemmas 29 and 30, respectively. Then with probability at least $(1-\delta)^{\bar{K}}$, Algorithm 3 terminates at a point satisfying

$$\|\nabla f(\boldsymbol{x}_k)\| \lesssim \epsilon \quad \textit{and} \quad \lambda_{\min}(\nabla^2 f(\boldsymbol{x}_k)) \gtrsim -\sqrt{L_H \epsilon}, \tag{3.8}$$

in at most K_1 iterations.

Finally, combining the complexity of Procedures 8 and 9, we provide the total computational complexity of our Algorithm 3 in terms of the function, gradient, and Hessian-vector evaluations.

Corollary 2. *Suppose that the assumptions of Theorem 4 hold. Let $\epsilon_g = \epsilon, \epsilon_H = \sqrt{L_H \epsilon}$, and $\bar{K}$ be defined as in (3.7). Then with probability at least $(1-\delta)^{\bar{K}}$, Algorithm 3 terminates after at most $\mathcal{O}(\epsilon^{-7/4})$ numbers of function, gradient, and Hessian-vector evaluations.*

Table 3.1: The upper bound of L_H for some non-convex problems in the form finite sum minimization (3.2). Here, we consider $\{\mathbf{a}_i, b_i\}_{i=1}^n$ as training data where $\mathbf{a}_i \in \mathbb{R}^d$ and $b_i \in \mathbb{R}$. When we calculate the upper bound of L_H for single data point, and for simplicity, we omit the sub-script of $\mathbf{a}_i$ and b_i. For Welsch's exponential, α is some positive constant.

Problem Formulation	Predictor Function	Upper bound of L_H for single data point	Upper bound of L_H for entire problem
$\sum_{i=1}^n (b_i - \phi(\langle \mathbf{a}_i, \mathbf{x} \rangle))^2$	$\phi(z) = 1/(1+e^{-z})$	$2\|\mathbf{a}\|^3(\|b\phi'''(z)\| + 3\|\phi'(z)\phi''(z)\| + \|\phi(z)\phi'''()\|) \leq 2(\|b\|+4)\|\mathbf{a}\|^3$	$\max_i 2(\|b\|+4)\|\mathbf{a}_i\|^3$
$\sum_{i=1}^n (b_i - \phi(\langle \mathbf{a}_i, \mathbf{x} \rangle))^2$	$\phi(z) = (e^z - e^{-z})/(e^z + e^{-z})$	$2\|\mathbf{a}\|^3(\|b\phi'''(z)\| + 3\|\phi'(z)\phi''(z)\| + \|\phi(z)\phi'''(z)\|) \leq 2(\|b\|+4)\|\mathbf{a}\|^3$	$\max_i 2(\|b\|+4)\|\mathbf{a}_i\|^3$
$\sum_{i=1}^n \phi(b_i - \langle \mathbf{a}_i, \mathbf{x} \rangle)$	$\phi(z) = (1 - e^{-\alpha z^2})/\alpha$	$\|\mathbf{a}\|^3\|\phi'''(z)\|$	$9\alpha^{3/2} \max_i \|\mathbf{a}_i\|^3$

3.2.3 Results without Line Search

Although Algorithm 3 employs approximated gradients and Hessian, the back-tracking line search for the step-size α_k is still chosen using the full function evaluation of f.[1] Since the cost of gradient evaluation is typically has the same order as that of the corresponding

[1]The same setting is considered in [252, 196].

Algorithm 4 Inexact Newton-CG without Line Search

```
1: Inputs: $\epsilon_g, \epsilon_H > 0$; Parameter $\theta \in (0, 1)$; Starting point $\mathbf{x}_0$; upper bound on Hessian
   norm $U_H > 0$; accuracy parameter $\zeta \in (0, \min\{1, U_H\})$;
2: for $k = 0, 1, 2, \cdots$ do
3:   if $\|\mathbf{g}_k\| \geq \epsilon_g$ then
4:     Call Procedure 8 with $\mathbf{H} = \mathbf{H}_k, M = U_H, \epsilon = \epsilon_H, \mathbf{g} = \mathbf{g}_k$ and accuracy parameter $\zeta$
       to obtain $\mathbf{d}$ and $d_{\text{type}}$;
5:     if $d_{\text{type}} == NC$ then
6:       $\mathbf{d}_k \leftarrow -\text{sgn}(\mathbf{d}^T\mathbf{g}_k)\frac{|\mathbf{d}^T\mathbf{H}_k\mathbf{d}|}{\|\mathbf{d}\|^2}\frac{\mathbf{d}}{\|\mathbf{d}\|}$;
7:     else
8:       $\mathbf{d}_k \leftarrow \mathbf{d}$;
9:     end if
10:    if $\|\mathbf{d}_k\| \leq \sqrt{\frac{\epsilon_g}{L_H}}$ then
11:      Call Procedure 9 with $\mathbf{H} = \mathbf{H}_k, M = U_H, \epsilon = \epsilon_H$ to obtain $\mathbf{v}$ and $\lambda_{min}(\mathbf{H}_k)$;
12:      if Procedure 9 certificates $\lambda_{\min}(\mathbf{H}_k) \geq -\epsilon_H$ then
13:        $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \mathbf{d}_k$ and terminate;
14:      else
15:        $\mathbf{d}_k \leftarrow -\text{sgn}(\mathbf{v}^T\mathbf{g}_k)\frac{|\mathbf{v}^T\mathbf{H}_k\mathbf{v}|}{\|\mathbf{v}\|^2}\mathbf{v}$ and $d_{\text{type}} = NC$;
16:      end if
17:    end if
18:  else
19:    Call Procedure 9 with $\mathbf{H} = \mathbf{H}_k, M = U_H, \epsilon = \epsilon_H$;
20:    if Procedure 9 certificates $\lambda_{\min}(\mathbf{H}_k) \geq -\epsilon$ then
21:      Terminate;
22:    else
23:      $\mathbf{d}_k \leftarrow -\text{sgn}(\mathbf{v}^T\mathbf{g}_k)\frac{|\mathbf{v}^T\mathbf{H}_k\mathbf{v}|}{\|\mathbf{v}\|^2}\mathbf{v}$ and $d_{\text{type}} = NC$;
24:    end if
25:  end if
26:  if $d_{\text{type}} = NC$ then
27:    $\alpha_k = -\frac{3\tilde{\theta}_k}{2(L_H+\eta)}$
28:  else
29:    $\alpha_k = -\left[\frac{3\theta_k^2(1-\zeta)}{4(L_H+\eta)}\right]^{1/2}\frac{\epsilon_H^{1/2}}{\|\mathbf{d}_k\|^{1/2}}$
30:  end if
31:  $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k\mathbf{d}_k$;
32: end for
```

function, the computation reduction from gradient approximation might be negligible. However, in this section, we show that a "fixed" (pre-defined) step size can be carefully chosen to obviate the need for function evaluations. Despite this desirable advantage, one can also identify two main disadvantages for this approach. First, the guaranteed descent in the objective will usually be smaller than what we can obtain from Lemmas 29 to 31. Second, our approach will require an approximate upperbound on the Lipchitz constant of Hessian (L_H), which might not be readily available. Fortunately, there are many important instances, in particular in machine learning, in which an estimate of (L_H) can easily be obtained, e.g., empirical risk minimization problems involving the usual squared loss [243] as well as and Welsch's exponential variant [264].

Although the gradient and Hessian in Algorithm 3 are already approximated, the backtracking line-search step is still costly when $n \gg 1$. Here, we present Algorithm 4, as a modification to Algorithm 3, which uses a fixed (pre-defined) step size rather than line search, and as a consequence, function evaluations are no longer needed. Clearly, eliminating exact functional evaluations would increase the efficiency of the overall algorithm. The main difference between Algorithm 3 and 4 is highlighted in blue.

Condition 8. *To make the output of Algorithm 3 satisfy Definition 6, we require the inexact gradient $\mathbf{g}_k$ and Hessian $\mathbf{H}_k$ to satisfy Definition 7 with*

$$\delta_{g,k} \leq \frac{1-\zeta}{8} \min \left(\max\left(\epsilon_g, \min(\epsilon_H \|\mathbf{d}_k\|, \|\mathbf{g}_k\|, \|\mathbf{g}_{k+1}\|)\right), \min\{\frac{3\epsilon_H}{8(L_H+\eta)}, \frac{3\epsilon_H^2}{65(L_H+\eta)}\} \right)$$

and $\quad \delta_H \leq \dfrac{1-\zeta}{4}\epsilon_H.$

Note that throughout this section, we assume $\epsilon_H = \sqrt{L_H \epsilon_g}$ to remove the function evaluation of Algorithm 4.

We are now ready to give the iteration complexity of Algorithm 4. The proof is similar to Theorem 4 and is therefore omitted.

Theorem 5. *Let Assumption 3 and Condition 8 hold. Define*

$$\hat{K}_2 := \left\lceil \frac{24(f(\boldsymbol{x}_0) - f_{\text{low}})}{\min\{\bar{c}_{\text{sol}}, \bar{c}_{\text{nc}}\}} \epsilon^{-3} \right\rceil + 4, \tag{3.9}$$

where $\bar{c}_{\text{sol}}$ and $\bar{c}_{\text{nc}}$ are defined in Lemmas 33 and 35, respectively. Then with probability at least $1 - \hat{K}_2\delta$, Algorithm 4 terminates at a point satisfying

$$\|\nabla f(\boldsymbol{x}_k)\| \lesssim \epsilon^2, \quad \lambda_{\min}(\nabla^2 f(\boldsymbol{x}_k)) \gtrsim -\epsilon, \tag{3.10}$$

in at most $\hat{K}_2$ iterations.

Note that the worst case iteration complexity of Algorithm 4 is the same as Algorithm 3 but the function evaluation of Algorithm 4 is removed.

3.2.4 Evaluation Complexity of Algorithm 4 for Finite-Sum Problems

Combining Lemma 1 with the sufficient conditions presented earlier, i.e., Condition 8, we can immediately obtain, similar, but probabilistic, iteration complexities as in Section 3.2.3. Since both Lemma 28 and Condition 8 are only guaranteed probabilistically, in order to get an accumulative success probability of $1-\delta$ for the entire $\hat{K}_2$ iterations, the per-iteration failure probability is set as $(1-\sqrt[\hat{K}_2]{(1-\delta)}) \in \mathcal{O}(\delta/T)$. Fortunately, this failure probability appears only in the "log factor" in both Lemmas 1 and 28 and so it is not the dominating cost. For $\hat{K}_2 \in \mathcal{O}(\epsilon^{-3}\})$ in Theorem 5, we can set the per-iteration failure probability to $\delta\epsilon^3$. In more details, we can set the per-iteration failure probability for Lemma 28 to be $\delta\epsilon^3/3$, and set that for Lemma 1 to be the same, i.e., $\delta\epsilon^3/3$.

Corollary 3 (Evaluation Complexity of Algorithm 4 For Finite-Sum Problem (3.2))**.** *Consider any $0<\epsilon_g,\epsilon_H,\delta<1$. Let $\delta_{g,t}$ and δ_H be as in Condition 7 and set $\delta_0=\delta\epsilon^3/3$. It is not hard to see*

$$\delta_{g,t} \in \mathcal{O}(\epsilon^2) \quad \textit{and} \quad \delta_H \in \mathcal{O}(\epsilon).$$

Let Lemma 28 has the failure probability with δ_0. Furthermore, for such $\delta_{g,t},\delta_H$ and δ_0, let the sample-size $|\mathcal{S}_g|$ and $|\mathcal{S}_H|$ be as in Lemma 1 and form the sub-sampled gradient and Hessian as in **H** *as in (2.14). For Problem (3.2), under Assumption 3 and Condition 8 terminates in at most $\hat{K}_2 \in \mathcal{O}(\epsilon^{-3})$ iterations, upon which, with probability $1-\delta$, we have that $\|\nabla F(\boldsymbol{x})\| \leq \mathcal{O}(\epsilon^2)$, and $\lambda_{\min}(\nabla^2 F(\boldsymbol{x})) \geq -\mathcal{O}(\epsilon)$.*

The total number of gradient, and Hessian-vector evaluation (T) is bounded by:

$$\begin{aligned}
T &= \underbrace{(\left\lceil \frac{24(f(\boldsymbol{x}_0)-f_{\text{low}})}{\min\{\bar{c}_{\text{sol}},\bar{c}_{\text{nc}}\}}\epsilon^{-3}\right\rceil + 4)}_{\hat{K}_2} \cdot (\underbrace{\frac{16K_g^2}{\delta_{g,t}^2}\log\frac{1}{\delta}}_{\textit{Gradient Sampling}} + \underbrace{\frac{16K_H^2}{\delta_H^2}\log\frac{2d}{\delta}}_{\textit{Hessian Sampling}} \cdot (\underbrace{\mathcal{O}(\epsilon^{-1/2})}_{\textit{Procedure } 8} + \underbrace{\mathcal{O}(\epsilon^{-1/2})}_{\textit{Procedure } 9}))\\
&= \mathcal{O}(\epsilon^{-3}) \cdot (\mathcal{O}(\epsilon^{-4}) + \mathcal{O}(\epsilon^{-5/2}))\\
&= \mathcal{O}(\epsilon^{-7}).
\end{aligned}$$

3.3 Numerical Evaluation

In this section, we evaluate the performance of Algorithm 3 and 4 on two model problems: non-linear least square (NLS) and multi-layer perceptron (MLP). We aim to illustrate the efficiency of gradient and Hessian approximations as well as of using the pre-defined step size. For our evaluation, we consider the following methods:

- `SGD`: Stochastic gradient descent with momentum [223]. The momentum parameter is set to the value of 0.0 in NLS and 0.9 in MLP. The gradient sampling size is 1% out of the full dataset in NLS, and 1000 in MLP.

• `Adagrad`: An adaptive first-order method developed in [73]. The gradient sampling size is 1000 in MLP.

• `Adam`: A modification of Adagrad which has become the method of choice within the machine learning community [125]. The two momentum terms in ADAM are set to be 0.9 and 0.999, which are typically chosen in practice. The gradient sampling size is 1000 in MLP.

• `Full GN`: Gauss-Newton method [91] with full pre-conditioned matrix and full gradient.

• `SubH GN`: Sub-sampling Gauss-Newton method [91] with approximated pre-conditioned matrix. The sampling size of pre-conditioned matrix is 5% out of the full dataset in NLS.

• `LBFGS`: Limited-memory BFGS method [145] with history size of 100.

• `Full NTCG`: Newton Method with Capped-CG solver [199] with full gradient and full Hessian.

• `SubH NTCG` (**this work**): Newton Method with Capped-CG solver [199] with full gradient and approximated Hessian. The sub-sampling size of Hessian is 1% in NLS and 1000 in MLP.

• `Inexact NTCG Full-Eval` (**this work**): Newton Method with Capped-CG solver with approximated gradient and approximated Hessian. The back-tracking line search uses the full dataset to evaluate the objective function. The gradient sampling is adaptively changed based on the following criterion: if $\|\mathbf{g}_t\| \geq 1.2\|\mathbf{g}_{t-1}\|$ or $\|\mathbf{g}_t\| \leq \|\mathbf{g}_{t-1}\|/1.2$, we respectively decrease or increase the sample size for gradient estimation by a factor of 1.2. Otherwise, the sample size stays the same as in the previous iteration. The sub-sampling size of Hessian is 1% in NLS and 1000 in MLP.

• `Inexact NTCG Sub-Eval` (**this work**): Newton Method with Capped-CG solver with approximated gradient and approximated Hessian. The back-tracking line search uses the same samplers as the gradient approximation to evaluate the function decreasing. The gradient sampling is adaptively changed based on the following criterion: if $\|\mathbf{g}_t\| \geq 1.2\|\mathbf{g}_{t-1}\|$ or $\|\mathbf{g}_t\| \leq \|\mathbf{g}_{t-1}\|/1.2$, we respectively decrease or increase the sample size for gradient estimation by a factor of 1.2. Otherwise, the sample size stays the same as the previous iteration. The sub-sampling size of Hessian is 1% in NLS and 1000 in MLP. Note that, our work here does not provide theoretical guarantee for this variant of NTCG. However, practically, we have found that this variant is highly effective. Intuitively, there is no need for very accurate functional evaluation at the beginning of iterations. As training continues, since the gradient sampling size increases, we will finally require very accurate functional evaluation.

• `Inexact NTCG Fixed` (**this work**): Newton Method with Capped-CG solver with approximated gradient and approximated Hessian. The step size is chosen as a constant.

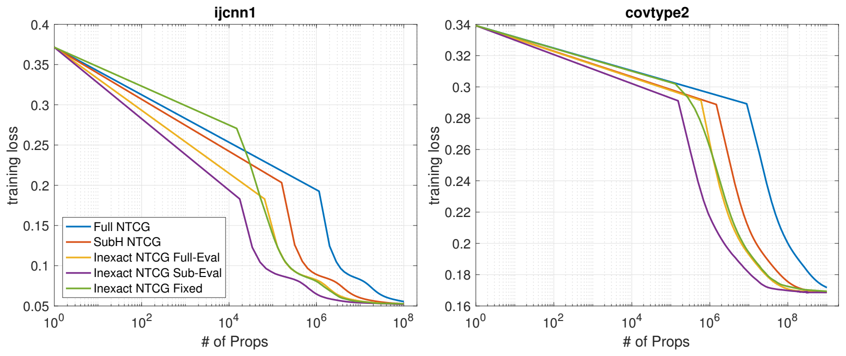

Figure 3.1: Comparison between our Inexact NTCG (`SubH NTCG`, `Inexact NTCG Full-Eval`, `Inexact NTCG Sub-Eval`, and `Inexact NTCG Fixed`) with `Full NTCG` on `ijcnn1` and `covertype`.

Particularly, the step size is 0.04 for $d_{\text{type}} = \text{NC}$ and 0.2 for $d_{\text{type}} = \text{SOL}$ in NLS. The gradient sampling is adaptively changed based on the following criterion: if $\|\mathbf{g}_t\| \geq 1.2\|\mathbf{g}_{t-1}\|$ or $\|\mathbf{g}_t\| \leq \|\mathbf{g}_{t-1}\|/1.2$, we respectively decrease or increase the sample size for gradient estimation by a factor of 1.2. Otherwise, the sample size stays the same as the previous iteration. The sub-sampling size of Hessian is 1% in NLS.

Similar to [243, 252], the performance of all the algorithms is measured by tallying the total *number of propagations*, i.e., the number of oracle calls of function, gradient and Hessian-vector products, as this is a machine/implementation independent measure of algorithm complexity.

3.3.1 Non-Linear Least Squares

We use the same setting as Section 2.3.2.

Inner Comparison. The comparison between different NTCG algorithms is shown in Figure 3.1. We can observe that all variants of NTCG except `Full NTCG` converges to similar training loss within the same amount of computation. Our Inexact NTCG family does so with much fewer number of propagation calls as compared to SubH. Although `Inexact NTCG Fixed` has faster per-iteration cost, as compared to other two Inexact NTCG, its actual convergence speed is much slower than that of `Inexact NTCG Sub-Eval`. This is caused by the per-step function decreasing of `Inexact NTCG Sub-Eval` being much better than that of `Inexact NTCG Fixed`.

Outer Comparison. Figure 3.2 depicts the results of comparison between `Inexact NTCG Sub-Eval` and other methods. As one can see, the performance of `Inexact NTCG Sub-Eval` is comparable to Full GN, which is faster than `SGD` and `LBFGS` and slower than `SubH GN`.

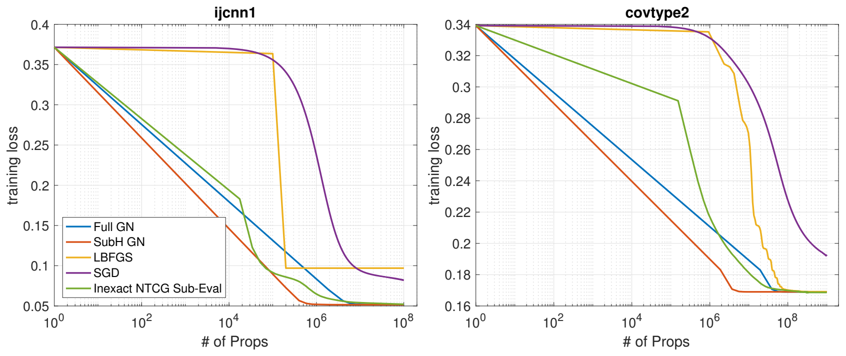

Figure 3.2: Comparison between `Inexact NTCG Sub-Eval` with other methods (`Full GN`, `SubH GN`, `LBFGS`, `SGD`) on `ijcnn1` and `covertype`.

3.3.2 Multi-Layer Perceptron

We use the same setting as Section 2.3.1.

We use several fixed step size in range $\alpha = [0.0001, 0.001, 0.01, 0.1]$ for all first-order methods, i.e., `SGD`, `Adagrad`, and `ADAM`. As is clearly observed here, the performance of first-order methods strongly depends on the step-size. For example, in Figure 3.3 (g)-(i), a finely-tuned `ADAM` can have superior performance. However, if the step-size is not chosen appropriately, the performance of `ADAM` could be unpleasantly erratic. This leads to the need for extremely expensive hyper-parameter sweeps, which methods such as those we present can avoid.

3.4 Conclusion

In this paper, we have considered inexact variants of Netwon-CG algorithm in which, to increase efficiency, the gradient and Hessian are all suitably approximated. For all of our proposed variants, we showed that the iteration complexities needed to achieve approximate second-order criticality are the same (up to some constant) as that of the exact variants. Also, we proved that a fixed step-size—instead of needing to use a back-tracking line search algorithm—exists to guarantee the convergent property. The advantages, and perhaps shortcomings, of our algorithms were also numerically demonstrated.

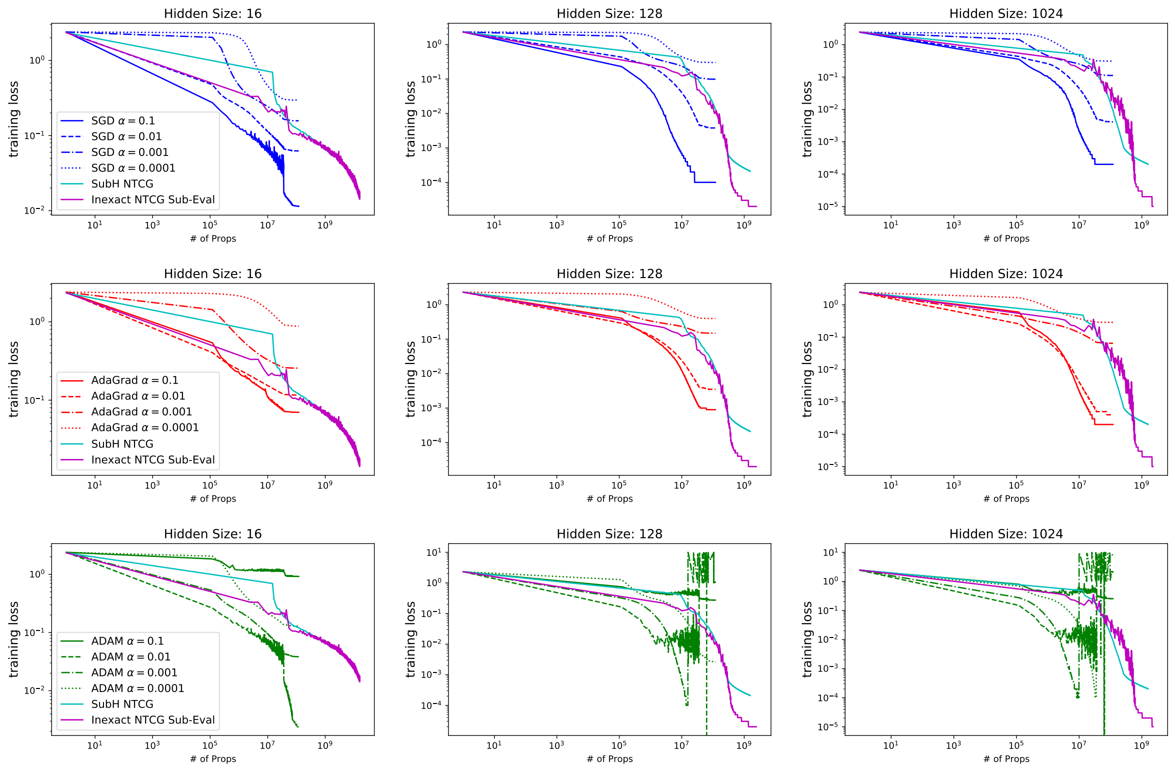

Figure 3.3: The comparison between our methods, `SubH NTCG` and `Inexact NTCG Sub-Eval`, with other first-order methods, `SGD` (first row), `Adagrad` (second row), and `ADAM` (thrid row) on MLP with different hidden size, 16 (first column), 128 (second column), and 1024 (third column).

Chapter 4

AdaHessian: An Adaptive Second Order Optimizer for Machine Learning

4.1 Introduction

The high dimensional and non-convex nature of many machine learning tasks has rendered many classical optimization methods inefficient for training and/or evaluating Neural Network (NN) models. After decades of research, first-order methods, and in particular variants of Stochastic Gradient Descent (SGD), have become the main workhorse for training NN models. However, they are by no means an ideal solution for training NN models. There are often a lot of ad-hoc rules that need to be followed very precisely to converge (hopefully) to a point with good generalization properties. Even the choice of the first-order optimizer has become an ad-hoc rule which can significantly affect the performance. For example, SGD with momentum is typically used in Computer Vision (CV); Adam is used for training transformer models for Natural Language Processing (NLP); and Adagrad is used for Recommendation Systems (RecSys). Using the wrong SGD variant can lead to significant performance degradation. Another challenging ad-hoc rule is the choice of hyperparameters and hyperparameter tuning methods, even after an optimizer is chosen. Hyperparameters include learning rate, decay schedule, choice of momentum parameters, number of warmup iterations, etc. As a result of these and other issues, one has to *babysit* the optimizer to make sure that training converges to an *acceptable* training loss, without any guarantee that a given number of iterations is enough to reach a local minima.

Importantly, one may *not* observe the above problems for certain popular learning tasks, such as ResNet50 training on ImageNet. The reason is that, for these tasks, years of industrial scale hyperparameter tuning has lead to what may be called *ideal SGD behaviour*. That is, for this problem, hyperparameters have been brute-force engineered to compensate for the deficiencies of first-order methods. Such a brute force approach is computationally and financially not possible for many large-scale learning problems—certainly it is not possible to do routinely—and this has made it challenging to train and apply NN models reliably.

Many of these issues stem from the fact that first-order methods only use gradient information and do not consider the curvature properties of the loss landscape, thereby leading to their suboptimal behaviour. second-order methods, on the other hand, are specifically designed to capture and exploit the curvature of the loss landscape and to incorporate both gradient and Hessian information. They are among the most powerful optimization algorithms, and they have many favorable properties such as resiliency to *ill-conditioned* loss landscapes, invariance to parameter scaling, and robustness to hyperparameter tuning. The main idea underlying second-order methods involves *preconditioning* the gradient vector before using it for weight update. This has a very intuitive motivation related to the curvature of the loss landscape. For a general problem, different parameter dimensions exhibit different curvature properties. For example, the loss could be very flat in one dimension and very sharp in another. As a result, the step size taken by the optimizer should be different for these dimensions, and we would prefer to take bigger steps for the flatter directions and relatively smaller steps for the sharper directions. This can be illustrated with a simple 2D quadratic function as shown in Figure 4.1, where we show the trajectories of different optimizers. As can be seen, first-order methods need a large number of steps for convergence and/or are hard to converge at all without hyperparameter tuning. However, second-order methods capture this curvature difference, by normalizing different dimensions through rotation and scaling of the gradient vector before the weight update. Nonetheless, this comes at a cost. Despite the theoretically faster convergence rate of second-order methods, they are rarely used for training NN models. This is due in part to their high computational cost.

In this paper, however, we will show that it is possible to compute *approximately* an exponential moving average of the Hessian and use it to precondition the gradient adaptively. The result is AdaHessian, an adaptive optimizer that exceeds the state-of-the-art performance for a wide range of learning problems, including ResNets [104] for CV, transformers [174] for NLP problems, and DLRM [166] models for RecSys tasks. In more detail, the main contributions of our work are the following.

• To reduce the overhead of second-order methods, we approximate the Hessian as a diagonal operator. This is achieved by applying Hutchinson's method to approximate the diagonal of the Hessian. Importantly, this approximation allows us to efficiently apply a root-mean-square exponential moving average to smooth out "rugged" loss surfaces. The advantage of this approach is that it has $\mathcal{O}(d)$ memory complexity.

• We incorporate a block diagonal averaging to reduce the variance of Hessian diagonal elements. In particular, this has no additional computational overhead in the Hutchinson's method, but it favorably affects the performance of the optimizer.

• To reduce AdaHessian overhead, we measure the sensitivity of AdaHessian to different hyperparameters such as learning rate, block diagonal averaging size, and delayed Hessian computation. Interestingly, our results show that AdaHessian is robust to those hyperparameters. See Section 4.5.1 and 4.5.2.

• We extensively test AdaHessian on a wide range of learning tasks. In all tests, AdaHessian significantly outperforms other adaptive optimization methods. Importantly note that these results are achieved even though we use the same learning rate schedule, weight decay, warmup schedule, dropout, as well as first/second-order moment coefficients. In particular, we consider the following tasks.

- **Computer Vision:** AdaHessian achieves significantly higher accuracy, as compared to Adam. For instance, for ResNet32 on Cifar10, AdaHessian achieves 93.08% as opposed to 91.63% achieved by Adam. Furthermore, for ResNet18 on ImageNet, AdaHessian achieves 70.08% accuracy as opposed to 64.53% of Adam. For all tests, AdaHessian achieves similar performance to the ideal SGD behavior, which is a result of hyperparameters having been tuned for many years at the industrial scale. Comparison with other optimizers and other models is discussed in Section 4.4.2.
- **Natural Language Processing:** AdaHessian improves the performance of transformers for machine translation and language modeling tasks, as compared to AdamW. In particular, AdaHessian significantly outperforms AdamW by 0.13/0.33 BLEU on IWSLT14/WMT14, and by 2.7/1.0 PPL on PTB/WikiText-103. Moreover, for SqueezeBERT [117] fine-tuning on GLUE, AdaHessian achieves 0.41 better points than AdamW. See Section 4.4.3, 4.4.4, and 4.4.5 for more details.
- **Recommendation System:** AdaHessian improves the performance of DLRM on the Criteo Ad Kaggle dataset by 0.032% as compared to Adagrad, which is commonly used. See Section 4.4.6 for more details.

• We measure the sensitivity of AdaHessian to different hyperparameters such as learning rate, spatial averaging size, and delayed Hessian computation. Interestingly, our results show that AdaHessian is robust to those hyperparameters. See Section 4.5.1 and 4.5.2 for more details.

We emphasize that our empirical results are achieved even though we use the same learning rate schedule, weight decay, warmup schedule, dropout, batch size, and first/second-order moment coefficients as the heavily-tuned default first-order baseline optimizers. Additional gains could be achieved if one wanted to extensively optimize these hyperparameters.

4.2 Problem Formulation and Related work

We focus on supervised learning tasks where the goal is to solve a non-convex stochastic optimization problem of the form:

$$\min_{\theta} \mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} l_i(x_i, y_i; \theta), \tag{4.1}$$

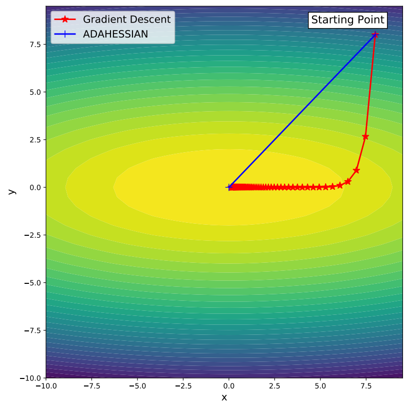

Figure 4.1: The trajectory of gradient descent and AdaHessian on a simple 2D quadratic function $f(x, y) = 10x^2 + y^2$. Gradient descent converges very slowly, even though this problem has a reasonable condition number. However, AdaHessian converges to the optimum in just one step. This is because second-order methods normalize the curvature difference between x and y axis by preconditioning the gradient vector before the weight update (by rescaling and rotating the gradient vector).

where $\theta \in \mathbb{R}^d$ denotes the model parameters, $l_i(x_i, y_i; \theta)$ is the loss function, (x_i, y_i) is the paired input data and its corresponding ground truth label, and N is the total number of data points in the training dataset. Furthermore, we denote the gradient of the loss w.r.t. model parameters as $\mathbf{g} = \frac{1}{N_B}\sum_{i=1}^{N_B} \frac{\partial l_i}{\partial \theta}$, and the corresponding second derivative (i.e., Hessian) as $\mathbf{H} = \frac{1}{N_B}\sum_{i=1}^{N_B} \frac{\partial^2 l_i}{\partial \theta^2}$, where N_B is the size of one mini-batch.

Solving (4.1) for a real learning problem (and not a simple model) is a very challenging task. Despite years of research, we have not yet been able to resolve several seemingly ad-hoc tricks that are required to converge (hopefully) to a *good* solution. Next, we briefly discuss the different popular optimization methods proposed in recent years to address the challenges associated with solving (4.1). This is by no means a comprehensive review, and we refer the interested reader to [26] for a thorough review.

4.2.1 Adaptive first-order Methods

Due to their simplicity and effectiveness, first-order optimization methods [195, 168, 73, 260, 125, 146] have become the de-facto algorithms used in deep learning. There are multiple variations, but these methods can be represented using the following general update formula:

$$\theta_{t+1} = \theta_t - \eta_t m_t / v_t, \tag{4.2}$$

where η_t is the learning rate, and m_t, and v_t denote the so called first and second moment terms, respectively. A simple and popular update method is SGD, originally proposed in

1951 as a root-solving algorithm [195]:

$$m_t = \beta m_{t-1} + (1-\beta)\mathbf{g}_t \quad \text{and} \quad v_t \equiv 1. \tag{4.3}$$

Here, $\mathbf{g}_t$ is the gradient of a mini-batch at t-th iteration and β is the momentum hyperparameter.

Using SGD to solve (4.1) is often very challenging, as the convergence of the iterative formulae in (4.2) is very sensitive to the right choice of the learning rate, its decay schedule, and the momentum parameter. To address this, several methods have been proposed to take into account the knowledge of the geometry of the data by scaling gradient coordinates, using the past gradient information. This can be viewed in one of two equivalent ways: either as automatically adjusting the learning rate in (4.2); or as an adaptive *preconditioning* of the gradient. One notable method is Adagrad [73, 156], which accumulates all the gradients from the first iteration and applies the square root of the result to precondition the current gradient. The update formulae in this case become[1]:

$$m_t = \mathbf{g}_t \qquad \text{and} \qquad v_t = \sqrt{\sum_{i=1}^{t} \mathbf{g}_i \mathbf{g}_i}. \tag{4.4}$$

While Adagrad works well for sparse settings, its performance significantly degrades for dense settings, which is the case for many machine learning tasks. In particular, this stems from the accumulation of all previous gradients for the preconditioner (4.4). This results in a monotonic increase in the magnitude of the second moment, v_t, which effectively translates into a rapid decay of the learning rate. To address this, several methods have been proposed where the intuition is to limit the accumulation to a small window of past iterations, and in particular exponentially reduce the weight of earlier iterations. Notable works incorporating this method are RMSProp, ADADelta, and Adam [227, 260, 125]. In particular, for Adam [125], the two moments for the update rule are the following:

$$m_t = \frac{(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}\mathbf{g}_i}{1-\beta_1^t} \qquad \text{and} \qquad v_t = \sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}\mathbf{g}_i\mathbf{g}_i}{1-\beta_2^t}}, \tag{4.5}$$

where $0 < \beta_1,\ \beta_2 < 1$ are two hyperparameters sometimes referred to as first and second moment coefficients. In particular, note that the sum over past gradients is scaled by β_2 which exponentially reduces the contribution of early gradients. A summary of the different m_t and v_t used by common first-order optimizers is given in Table 4.1. A notable variant here is AdamW [146], which shows that decoupling weight decay from the update equation of Adam can lead to a noticeable performance improvement. Recently, AdamW has become the preferred optimizer for NLP tasks, and in particular for training transformers [232]. There are also many other variants of adaptive first-order methods [47, 265, 147, 213, 218].

[1]Throughout the chapter, without further notification, for two vectors, e.g., a and b, we use both ab and $a \odot b$ to denote the element-wise product, and $\langle a, b\rangle$ denotes the inner product.

Table 4.1: Summary of the first and second moments used in different optimization algorithms for updating model parameters ($\theta_{t+1} = \theta_t - \eta m_t / v_t$). Here β_1 and β_2 are first and second moment hyperparameters.

Optimizer	m_t	v_t
SGD [195]	$\beta_1 m_{t-1} + (1-\beta_1)\mathbf{g}_t$	1
Adagrad [73]	$\mathbf{g}_t$	$\sqrt{\sum_{i=1}^t \mathbf{g}_i \mathbf{g}_i}$
Adam [125]	$\frac{(1-\beta_1)\sum_{i=1}^t \beta_1^{t-i}\mathbf{g}_i}{1-\beta_1^t}$	$\sqrt{\frac{(1-\beta_2)\sum_{i=1}^t \beta_2^{t-i}\mathbf{g}_i\mathbf{g}_i}{1-\beta_2^t}}$
RMSProp [227]	$\mathbf{g}_t$	$\sqrt{\beta_2 v_{t-1}^2 + (1-\beta_2)\mathbf{g}_t\mathbf{g}_t}$
AdaHessian	$\frac{(1-\beta_1)\sum_{i=1}^t \beta_1^{t-i}\mathbf{g}_i}{1-\beta_1^t}$	$\sqrt{\frac{(1-\beta_2)\sum_{i=1}^t \beta_2^{t-i}\boldsymbol{D}_i^{(s)}\boldsymbol{D}_i^{(s)}}{1-\beta_2^t}}$

Despite all these attempts, it is still not clear which optimizer should work for a *new* learning task/model. This is in fact one of the main baffling practical issues in machine learning, and one for which theory has little to say. For example, SGD is currently the best performing optimizer for some CV tasks. That is, using other variants such as AdamW leads to significantly worse generalization performance. However, for NLP tasks, AdamW has the best performance by a large margin as compared to SGD. The point here is that even the choice of the optimizer has effectively become a hyperparameter.

4.2.2 second-order Methods

second-order methods are among the most powerful optimization methods that have been designed, and there have been several attempts to use their many advantages for training NNs. second-order methods are designed to address *ill-conditioned* optimization problems by automatically rotating and rescaling the gradient. This allows one to choose a better descent direction, and to adjust automatically the learning rate for each parameter. There have also been multiple theoretical studies showing better convergence rate of second-order based methods [24, 252, 197, 247, 245, 244, 57, 2, 238, 3, 38, 48]. In particular, second-order methods can guarantee convergence to second-order critical points, while the vast majority of first-order methods lack such guarantees. For example, theoretically it has been shown that some first-order methods can only converge to an approximate second-order critical point [81, 123, 139, 192].

Newton's method is a classical second-order method where one solves a linear system, essentially meaning that the inverse of the local Hessian is used at every iteration to precondition the gradient vector.[2] One major challenge with this approach is that it can be

[2]To be clear, when we refer to computing an inverse, we mean that we use a numerical method that performs a linear equation solve that effectively amounts to working with the inverse implicitly. Of course, one would never actually compute the Hessian inverse explicitly.

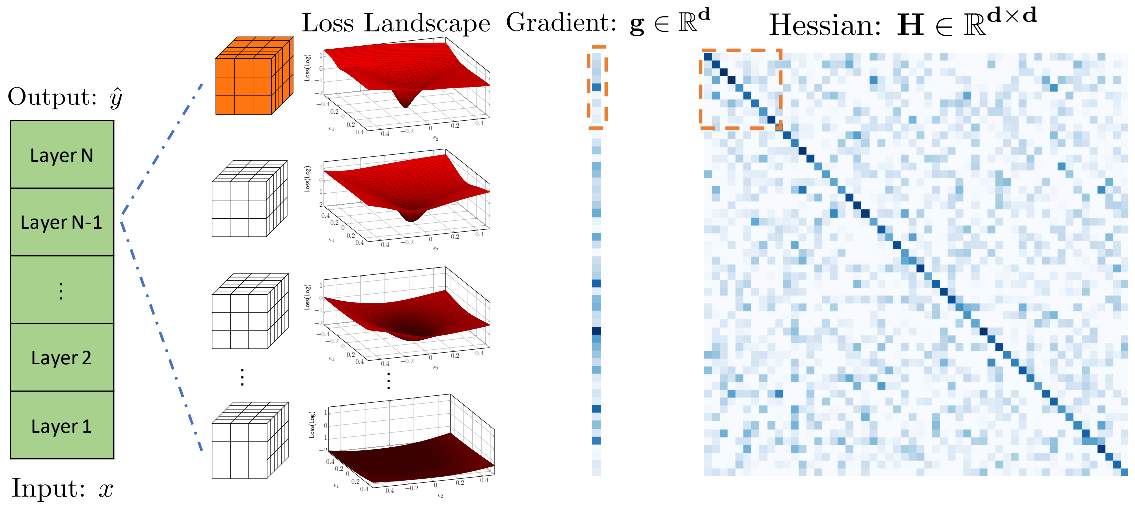
Loss Landscape
Gradient: $\mathbf{g} \in \mathbb{R}^{\mathbf{d}}$
Hessian: $\mathbf{H} \in \mathbb{R}^{\mathbf{d}\times\mathbf{d}}$
Output: $\hat{y}$
Layer N
Layer N-1
Layer 2
Layer 1
Input: x

Figure 4.2: A simple model with N layers (first column); with the convolutional blocks of the N-1 layer shown (second column); and the loss landscape of each block (third column), which can be calculated by perturbing the convolutions's parameters in two different eigendirections. (See [254] for details of how to construct loss landscape.) Note the different loss landscape topologies. first-order methods do not explicitly capture this difference. The entries (3D tensors) colored in orange show the components used for calculating the spatial average of Hessian. The part of the gradient (fourth panel) highlighted in the orange box is the corresponding gradient of the orange convolution kernel; and the part of the Hessian diagonal (fifth panel) highlighted in the orange box is used to compute the spatial average.

expensive to solve the linear system, naïvely requiring cubic computational complexity, not including the cost of forming the Hessian itself and the corresponding quadratic memory complexity. However, the overhead of such a naïve implementation can be improved by using so-called matrix free methods, where the Hessian matrix is never explicitly formed (addressing quadratic memory cost), and its inverse is approximately and only implicitly applied (addressing the cubic computational complexity).

One seminal work here is the limited memory BFGS (LBFGS) [31] method which has a desirable linear computational and memory complexity. This approach approximates the Hessian as a series sum of first-order information from prior iterations. As such, these approaches that do not directly use the Hessian operator are referred to as *Quasi-Newton* methods. While this approach works well for many optimization problems, it does not work well for many machine learning problems. One reason for this is that LBFGS method requires full batch gradients, as stochastic gradients can lead to drastic errors in the approximation [25]. This is one of the main challenges with Quasi-Newton methods applied to machine learning problems [4]. Other approaches such as approximating the Hessian as the Kronecker product of vectors have also been explored [154].

There has also been work on enhancing first-order methods by incorporating the Fisher information matrix [98]. The main idea is to use the Fisher information instead of the

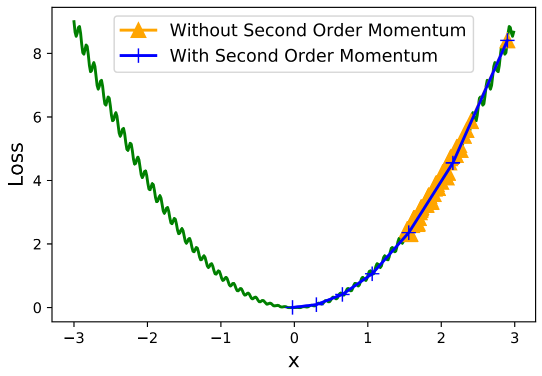

Figure 4.3: Local versus global curvature. Illustration of the local curvature which can be noisy, and the global curvature with a simple 1D problem $f(x) = x^2 + 0.1x\sin(20\pi x)$. Using the exponential moving average of (4.12) is key to avoid the misleading local curvature information. To demonstrate this we test AdaHessian without moving average (orange trajectory) which does not converge even after 1000 iterations. On the other hand, AdaHessian converges in 7 iterations with the moving average enabled.

squared norm of the gradient. A naïve use of Fisher information has computational and memory overhead, but it is possible to also approximate the Fisher information matrix using low rank decomposition [98].

Another line of work has been to incorporate automatically the Hessian operator itself, instead of approximating it using first-order information. A work here is [208] which uses Gauss-Newton Hessian diagonal to adjust adaptively the learning rate. The work of [244] also directly incorporates the Hessian using a trust region method.

While the above approaches are very interesting and result in a good performance for simple models, they do not achieve comparable results for more complex NN architectures. One of the reasons that second-order methods have not been successful yet for machine learning, as opposed to other domains such as scientific computing, is due to the stochastic nature of the problem. Such stochastic noise leads to an erroneous approximation of the Hessian, leading to suboptimal descent directions. SGD is more robust to such noise since we can efficiently incorporate moving averages and momentum. Ideally, if there was a way to apply the same moving average method to the Hessian, then that would help smooth out local curvature noise to get a better approximation to the non-noisy curvature of the loss landscape. However, such an approximation is challenging since the Hessian is a matrix that cannot be explicitly formed to be averaged, whereas it is easy to form the gradient vector.

As we show below, AdaHessian addresses this problem by incorporating the Hutchinson's method along with spatial averaging to reduce the impact of the stochastic noise. The result exceeds the performance of all the above methods for machine learning tasks. Next, we formally introduce the AdaHessian algorithm.

4.3 Methodological Approach

Here, we first provide the formulation for the full Newton method in Section 4.3.1. Then, we describe the three components of AdaHessian, namely Hessian diagonal approximation (Section 4.3.2), spatial averaging (Section 4.3.3), and Hessian momentum (Section 4.3.4). Finally, we discuss the overall formulation of AdaHessian in Section 4.3.5.

4.3.1 A General Hessian Based Descent Direction

For the loss function $f(w) : \mathbb{R}^d \to \mathbb{R}$, let us denote the corresponding gradient and Hessian of $f(w_t)$ at iteration t as $\mathbf{g}_t$, and $\mathbf{H}_t$, respectively.[3] A general descent direction can then be written as follows for a positive-definite Hessian:

$$\Delta w_t = \mathbf{H}_t^{-k}\mathbf{g}_t, \quad \text{where} \;\; \mathbf{H}_t^{-k} = U_t^T \Lambda_t^{-k} U_t. \tag{4.6}$$

Here, we refer to $0 \leq k \leq 1$ as *Hessian power*, and $U_t^T \Lambda_t U_t$ is the eigendecomposition of $\mathbf{H}_t$. Note that for $k = 0$, we recover the gradient descent method; and for $k = 1$, we recover the Newton method. In our empirical tests we consider non-convex machine learning problems, but we provide a standard convergence behaviour of (4.6) in Appendix C.1 for a simple strongly convex and strictly smooth function $f(w)$. (We emphasize that the proof is very standard and we are only including it for completeness.)

The basic idea of Hessian based methods is to *precondition* the gradient with the $\mathbf{H}^{-k}$ and use $\mathbf{H}^{-k}\mathbf{g}$ for the update direction, instead of using the *bare* gradient $\mathbf{g}$ vector. The preconditioner automatically rotates and rescales the gradient vector. This is important since the loss landscape curvature is generally different across different directions/layers and since these directions need not correspond to the canonical axes. This is illustrated in Figure 4.2, where we show a 2D schematic plot of the loss landscape for different convolution channels [254]. Each channel can have a different loss landscape topology. For example, the last channel has a much flatter loss landscape, as compared to other layers. As a result, it is preferable to take a larger step size for the last channel than for the first channel, which has a very "sharp" loss landscape. Problems that exhibit this behaviour are *ill-conditioned*. The role of the Hessian is to automatically normalize this ill-conditionedness by stretching and contracting different directions to accommodate for the curvature differences (full Newton method also rotates the gradient vector along with adjusting the step size).

However, there are two major problems with this approach. The first problem is that a naïve use of the Hessian preconditioner comes at the prohibitively high cost of applying Hessian inverse to the gradient vector at every iteration ($\mathbf{H}^{-k}\mathbf{g}$ term). The second and more challenging problem is that local Hessian (curvature) information can be very misleading for a noisy loss landscape. A simple example is illustrated in Figure 4.3, where we plot a simple parabola with a small sinusoidal noise as the loss landscape (shown in green).

[3]Without confusion, we use the same gradient and Hessian notations for $f(w)$ and $\mathcal{L}(\theta)$. Furthermore, when there is no confusion we will drop subscript t.

As one can see, the local Hessian (curvature) information is completely misleading, as it computes the curvature of the sinusoidal noise instead of global Hessian information for the parabola. Applying such misleading information as the preconditioner would actually result in very small steps to converge to one of the many local minima created by the sinusoidal noise. The same problem exists for the gradient as well, but that can be alleviated by using gradient momentum instead of local gradient information. However, as mentioned before it is computationally infeasible to compute (naïvely) a Hessian momentum. The reason is that we cannot form the Hessian matrix and average it throughout different iterations, as such an approach has quadratic memory complexity in the number of parameters along with a prohibitive computational cost. However, one could use Randomized Numerical Linear Algebra to get a sketch of the Hessian matrix [251, 254, 99]. In particular, we show how this can be done to approximate the Hessian diagonal. However, as we discuss next, both problems can be resolved by using Hessian diagonal instead of the full Hessian.

4.3.2 Hessian Diagonal Approximation

To address the issue that applying the inverse Hessian to the gradient vector at every iteration is computationally infeasible, one could use an inexact Newton method, where an approximate Hessian operator is used instead of the full Hessian [64, 246, 245, 252, 24]. The most simple and computationally efficient approach is to approximate the Hessian as a diagonal operator in (4.6):

$$\Delta w = diag(\mathbf{H})^{-k}\mathbf{g}, \tag{4.7}$$

where $diag(\mathbf{H})$ is the Hessian diagonal, which we denote as $\boldsymbol{D}$.[4] We show that using (4.7) has the same convergence rate as using (4.6) for simple strongly convex and strictly smooth function $f(w)$ (see Appendix C.2). Note that we only include the proof for completeness, and our algorithm AdaHessian can be applied for general machine learning problems.

The Hessian diagonal $\boldsymbol{D}$ can be efficiently computed using the Hutchinson's method. The two techniques we use for this approximation are: (i) a Hessian-free method [251]; and (ii) a randomized numerical linear algebra (RandNLA) method [17, Figure 1]. In particular, the Hessian-free method is an oracle to compute the multiplication between the Hessian matrix $\mathbf{H}$ with a random vector z, i.e.,

$$\frac{\partial \mathbf{g}^T z}{\partial \theta} = \frac{\partial \mathbf{g}^T}{\partial \theta} z + \mathbf{g}^T \frac{\partial z}{\partial \theta} = \frac{\partial \mathbf{g}^T}{\partial \theta} z = \mathbf{H} z. \tag{4.8}$$

Here, the first equality is the chain rule, and the second equality is since z is independent of θ. (4.8) effectively allows us to compute the Hessian times a vector z, without having to form explicitly the Hessian, by backpropotating the $\mathbf{g}^T z$ term. This has the same cost as ordinary gradient backpropogation [251]. Then, with the Hessian matvec oracle, one can compute the Hessian diagonal using Hutchinson's method:

$$\boldsymbol{D} = diag(\mathbf{H}) = \mathbf{E}[z \odot (\mathbf{H} z)], \tag{4.9}$$

[4] Note that $\boldsymbol{D}$ can be viewed as a vector, in which case $\boldsymbol{D}^{-k}\mathbf{g}$ is an element-wise product of vectors. Without clarification, $\boldsymbol{D}$ is treated as a vector for the rest of the paper.

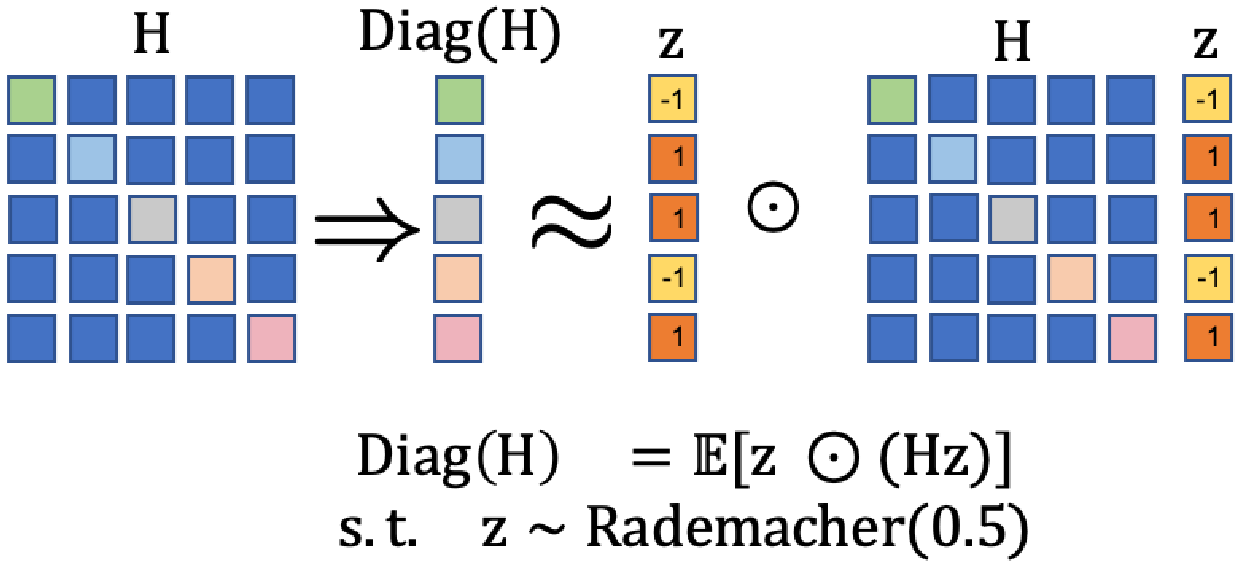

Figure 4.4: Illustration of the diagonal Hessian estimation with Hutchinson's method.

where z is a random vector with Rademacher distribution, and $\mathbf{H}z$ is computed by the Hessian matvec oracle given in (4.8). This process is illustrated in Figure 4.4. It can be proved that the expectation of $z \odot (\mathbf{H}z)$ is the Hessian diagonal [17].

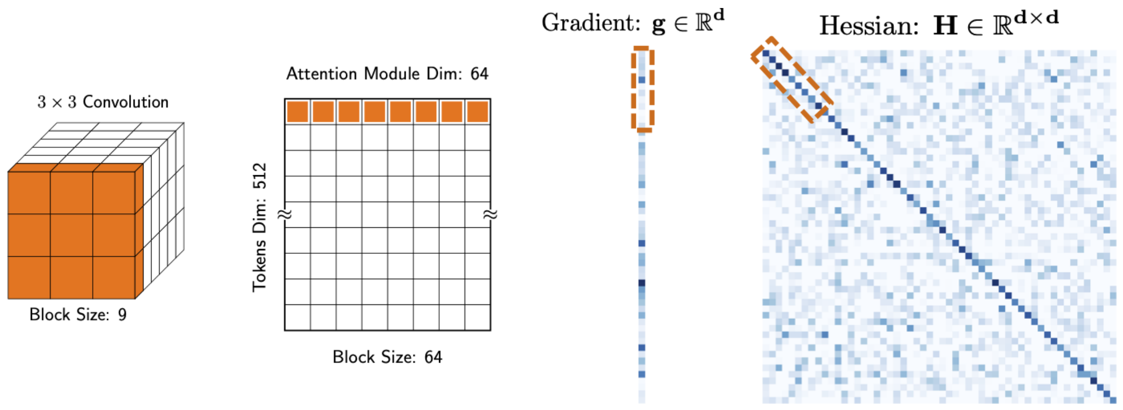

Figure 4.5: Illustration of the block size used to average the Hessian diagonal to smooth spatial variations. For a convolution layer, we average each channel (groups of 9 parameters); and for multi-head attention, we average consecutive elements along the rows (attention dimension). We found that using block averaging helps, although AdaHessian is not very sensitive to this hyperparameter as illustrated in Table 4.7.

Another important advantage, besides computational efficiency, of using the Hessian diagonal is that we can compute its moving average to resolve the local noisy Hessian as mentioned at the end of Section 4.3.1. This allows us to smooth out noisy local curvature information, and to obtain estimates that use global Hessian information instead. We incorporate both spatial averaging and momentum (temporal averaging) to smooth out this noisy Hessian estimate as described next.

4.3.3 Spatial Averaging

The Hessian diagonal can vary significantly for each single parameter dimension of the problem. We found it helpful to perform spatial averaging of Hessian diagonal and use the average to smooth out spatial variations. For example, for a convolutional layer, each convolution parameter can have a very different Hessian diagonal. In AdaHessian we compute the average of the Hessian diagonal for each convolution kernel (3×3) as illustrated in Figure 4.5. Mathematically, we perform a simple spatial averaging on the Hessian diagonal as follows:

$$\boldsymbol{D}^{(s)}[ib+j] = \frac{\sum_{k=1}^{b} \boldsymbol{D}[ib+k]}{b} \qquad \text{for } 1 \le j \le b, 0 \le i \le \frac{d}{b} - 1, \tag{4.10}$$

where $\boldsymbol{D} \in \mathbb{R}^d$ is the Hessian diagonal, $\boldsymbol{D}^{(s)} \in \mathbb{R}^d$ is the spatially averaged Hessian diagonal, $\boldsymbol{D}[i]$ ($\boldsymbol{D}^{(s)}[i]$) refers to the i-th element of $\boldsymbol{D}$ ($\boldsymbol{D}^{(s)}$), b is the spatial average block size, and d is the number of model parameters divisible by b. We show that replacing $\boldsymbol{D}$ in (4.7) by $\boldsymbol{D}^{(s)}$ in (4.10), the update direction has the same convergence rate as using (4.6) for simple strongly convex and strictly smooth function $f(w)$ (see Appendix C.3). Note that we only include the proof for completeness, and our algorithm AdaHessian can be applied for general machine learning problems.

Figure 4.5 provides illustration of spatial averaging for both convolutional and matrix kernels. In general, the block size b is a hyperparameter that can be tuned for different tasks. While this is a new hyperparameter that can help the performance, the performance of AdaHessian is not very sensitive to it (we provide sensitivity results in Section 4.5.1).

Next we describe momentum which is another useful method to smooth out Hessian noise over different iterations.

4.3.4 Hessian Momentum

We can easily apply momentum to Hessian diagonal since it is a vector instead of a quadratically large matrix. This enables us to adopt momentum for Hessian diagonal in AdaHessian. More specifically, let $\bar{\boldsymbol{D}}_t$ denote the Hessian diagonal with momentum that is calculated as:

$$\bar{\boldsymbol{D}}_t = \sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i} \boldsymbol{D}_i^{(s)} \boldsymbol{D}_i^{(s)}}{1-\beta_2^t}}, \tag{4.11}$$

where $\boldsymbol{D}^{(s)}$ is the spatially averaged Hessian diagonal (defined in (4.10)), and $0 < \beta_2 < 1$ is the second moment hyperparameter. Note that this is exactly the same as the momentum

Algorithm 5 AdaHessian

```
1: Input:
       - Initial Parameter: $\theta_0$
       - Learning rate: $\eta$
       - Exponential decay rates: $\beta_1,\ \beta_2$
       - Block size: $b$
       - Hessian Power: $k$
2: Set: $m_0 = 0,\ v_0 = 0$
3: for t = $1, 2, \ldots$ do
4:     $\mathbf{g}_t \leftarrow$ current step gradient
5:     $\boldsymbol{D}_t \leftarrow$ current step estimated diagonal Hessian
6:     Compute $\boldsymbol{D}_t^{(s)}$ based on (4.10)
7:     Update $\bar{\boldsymbol{D}}_t$ based on (4.11)
8:     Update $m_t,\ v_t$ based on (4.12)
9:     $\theta_t = \theta_{t-1} - \eta m_t / v_t$
10: end for
```

term in Adam [125] or RMSProp [227] except that we are using the spatial averaging Hessian diagonal instead of the gradient.

To illustrate the importance of Hessian momentum, we provide a simple example in 1D by considering $f(x) = x2 + 0.1xsin(20\pi x)$, as shown in Figure 4.3. It can be clearly seen that the method without the second-order momentum gets trapped at a local minima even with more than 1000 iterations (orange trajectory). On the contrary, the optimization converges within 7 iterations with Hessian momentum (blue trajectory). (While this example is over-simplified in certain ways, we are using it here only to convey the importance of momentum.)

4.3.5 AdaHessian

To summarize, instead of only applying momentum for gradient, AdaHessian uses *spatial averaging* and *Hessian momentum* to smooth out local variations in Hessian diagonal. More specifically, the first and second-order moments (m_t and v_t) for AdaHessian are computed as follows:

$$\begin{aligned} m_t &= \frac{(1-\beta_1)\sum_{i=1}^{t} \beta_1^{t-i} \mathbf{g}_i}{1-\beta_1^t}, \\ v_t &= (\bar{\boldsymbol{D}}_t)^k = \left(\sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i} \boldsymbol{D}_i^{(s)} \boldsymbol{D}_i^{(s)}}{1-\beta_2^t}} \right)^k, \end{aligned} \tag{4.12}$$

where $0 < \beta_1,\ \beta_2 < 1$ are the first and second moment hyperparameters that are also used in Adam. Note that Adam uses the same formulation except that the spatial averaging Hessian diagonal $\boldsymbol{D}_i^{(s)}$ is replaced with gradient.

Table 4.2: Results of ResNet20/32 on Cifar10 (left two columns) and ResNet18 on ImageNet (last column). On Cifar10: Adam performs consistently worse than SGD; AdamW has slightly worse performance than SGD; and AdaHessian outperforms AdamW and even gets accuracy comparable to SGD. On ImageNet: AdaHessian has significantly better accuracy than Adam (5.53%), AdamW (2.67%), and has similar performance to SGD.

Dataset	**Cifar10**		**ImageNet**
	`ResNet20`	`ResNet32`	`ResNet18`
SGD [201]	92.08 ± 0.08	$\mathbf{93.14 \pm 0.10}$	70.03
Adam [125]	90.33 ± 0.13	91.63 ± 0.10	64.53
AdamW [146]	91.97 ± 0.15	92.72 ± 0.20	67.41
AdaHessian	$\mathbf{92.13 \pm 0.18}$	93.08 ± 0.10	**70.08**

The main overhead of AdaHessian is the Hutchinson's method to approximate Hessian diagonal, $\boldsymbol{D}$. We use one Hutchinson step per iteration to approximate the Hessian diagonal (i.e., one random Rademacher vector z in (4.9)). The cost of this estimation is one Hessian matvec (to compute Hz), which is equivalent to one gradient backpropagation [251, 254].

Also note that it is possible to get a more accurate approximation to Hessian diagonal by using more Hutchinson steps per iteration. However, we found that one step per iteration performs well in practice since the multiple calculations could be performed as Hessian momentum (Section 4.3.4). In fact, as we discuss in Section 4.5.2, it is possible to skip the Hutchinson calculation for few iterations to reduce further its computational overhead, without significant impact on final accuracy.

4.4 Results

4.4.1 Experiment Setup

One of the problems with several formerly proposed optimization methods is that the methods were originally tested with very simple models on very few tasks. When those methods were later tested by the community on more complex models the results were often worse than popular optimization methods. To avoid such a scenario, we extensively test AdaHessian on a wide range of learning tasks, including image classification, neural machine translation (NMT), language modeling (LM), and recommendation system (RecSys). We compare the AdaHessian performance with SGD, Adam, AdamW [146], and Adagrad. Moreover, to enable a fair comparison we will use the same β_1 and β_2 parameters in AdaHessian as in Adam/AdamW for each task, even though those default values may favor Adam (or AdamW) and disfavor AdaHessian. Furthermore, we will use the exact same weight decay and learning rate schedule in AdaHessian as that used by other optimizers. Below we briefly explain each of the learning tasks tested.

Table 4.3: NMT performance (BLEU) on IWSLT14 De-En and WMT14 En-De testsets (higher is better). Unlike in Table 4.2, SGD has significantly worse results than AdamW. Note that AdaHessian outperforms the default and heavily tuned optimizer AdamW by 0.13 and 0.33 on IWSLT14 and WMT14, which is significant for this task.

Model	**IWSLT14** `small`	**WMT14** `base`
SGD	28.57 ± .15	26.04
AdamW [146]	35.66 ± .11	28.19
AdaHessian	**35.79** ± .06	**28.52**

Image Classification We experiment on both Cifar10 (using ResNet20/32) and ImageNet (using ResNet18) datasets. Cifar10 consists of 50k training images and 10k testing images. ImageNet has 1.2M training images and 50k validation images. We follow the settings described in [104] for training. We run each experiment 5 times on Cifar10 and report the mean and standard deviation of the results.

Neural Machine Translation (NMT) We use IWSLT14 German-to-English (De-En) and WMT14 English-to-German (En-De) datasets. Transformer `base` architecture is used for WMT14 (4.5M sentence pairs), and `small` architecture is used for IWSLT14 (0.16M sentence pairs). We follow the settings reported in [174] and use pre-normalization described in [236]. The length penalty is set to 0.6/1.0 and the beam size is set to 4/5 for WMT/IWSLT [173]. We report the average results of the last 10/5 checkpoints respectively. For NMT, BLEU score is used [176]. In particular, we report tokenized case-sensitive BLEU on WMT14 En-De and case-insensitive BLEU IWSLT14 De-En. Furthermore, we use AdamW for this task instead of Adam since the former is the standard optimizer (Adam consistently scores lower).

Language Modeling We use PTB [159] and Wikitext-103 [157] datasets, which contain 0.93M and 100M tokens, respectively. Following [149], a three-layer tensorized transformer core-1 for PTB and a six-layer tensorized transformer core-1 for Wikitext-103 are used in the experiments. We apply the multi-linear attention mechanism with masking and report the perplexity (PPL) on the test set with the best validation model.

Natural Language Understanding We use the GLUE task [234] to evaluate the fine-tuning performance of SqueezeBERT [116]. More specifically, we use 8 different tasks in GLUE and report and final average performance on the validation dataset.

Recommendation System The Criteo Ad Kaggle dataset contains approximately 45 million samples over 7 days. We follow the standard setting and use the first 6 days as the training set and the last day as the test set. Furthermore, we use DLRM, a novel recommendation model that has been recently released by Facebook [166]. The testing metric for Recommendation Systems is Click Through Rate (CTR), measured on training and test sets.

We refer the interested reader to Appendix C.4 for more detailed experimental settings.

Table 4.4: LM performance (PPL) on PTB and Wikitext-103 test datasets (lower is better). The PPL of AdaHessian is 2.7 and 1.0 lower than that of AdamW.

Model	**PTB**	**Wikitext-103**
	`Three-Layer`	`Six-Layer`
SGD	59.9 ± 3.0	78.5
AdamW [146]	54.2 ± 1.6	20.9
AdaHessian	**51.5** ± 1.2	**19.9**

Next we report the experimental results on each of these tasks.

4.4.2 Image Classification

The results on Cifar10 are shown in Table 4.2. First, note the significantly worse performance of Adam, as compared to SGD even on this simple image classification dataset. Particularly, Adam has 1.75%/1.51% lower accuracy for ResNet20/32 than SGD. AdamW achieves better results than Adam, but its performance is still slightly worse than SGD. However, AdaHessian achieves significantly better results as compared to Adam (1.80%/1.45% for ResNet20/32), even though we use the same β_1 and β_2 parameters in AdaHessian as in Adam. That is, we did not tune these two hyperparameters, even though tuning them could potentially lead to even better performance.[5] Compared with SGD, AdaHessian achieves comparable accuracy for both ResNet20 (0.05% higher) and ResNet32 (0.06% lower). The training and testing curves of different optimizers for ResNet20/32 on Cifar10 are shown in Figure C.1.

Next, we use the best learning rate obtained by training ResNet20/32 on Cifar10 to optimize ResNet18 on ImageNet for all four optimizers. We try two different learning rate schedules for all four optimizers, and we use the one with the better result. The two learning rate schedules are quite standard, i.e., the step decay schedule and the plateau based schedule [179]. The final result is reported in Table 4.2. Again note that the final performances of Adam and AdamW are much worse than that of SGD and AdaHessian. We plot the training and testing curve in Figure C.2.

It is worthwhile to note that our learning rate tuning is performed at an academic scale, but AdaHessian still significantly exceeds other adaptive methods and reaches the same performance level as SGD which has been tuned at the industrial scale.

[5]In fact, in Table 4.8 we achieve 92.40 for ResNet20 which is higher than what we report in Table 4.2. This is to emphasize that we only tuned learning rate in Table 4.2. Still AdaHessian achieves significantly better results than Adam.

Table 4.5: Comparison of AdamW and AdaHessian for SqueezeBERT on the development set of the GLUE benchmark. As can be seen, the average performance of AdaHessian is 0.41 higher as compared to AdamW. The result of AdamW$^+$ is directly from [116] and the result of AdamW* is reproduced by us.

	RTE	**MPRC**	**STS-B**	**SST-2**	**QNLI**	**QQP**	**MNLI-m**	**MNLI-mm**	Avg.
AdamW$^+$ [116]	71.8	89.8	89.4	**92.0**	**90.5**	89.4	82.9	82.3	86.01
AdamW*	79.06	90.69	90.00	91.28	90.30	**89.49**	82.61	81.84	86.91
AdaHessian	**80.14**	**91.94**	**90.59**	91.17	89.97	89.33	**82.78**	**82.62**	**87.32**

4.4.3 Neural Machine Translation

We use BLEU [176] as the evaluation metric for NMT. Following standard practice, we measure tokenized case-sensitive BLEU and case-insensitive BLEU for WMT14 En-De and IWSLT14 De-En, respectively. For a fair comparison, we do not include other external datasets.

The NMT results are shown in Table 4.3. The first interesting observation is that here SGD performs much worse than AdamW (which is opposite to its behaviour for image classification problems where SGD has superior performance; see Appendix 4.4.2). As pointed out in the introduction, even the choice of the optimizer has become another hyperparameter. In particular, note that the BLEU scores of SGD are 7.09 and 2.15 lower than AdamW on IWSLT14 and WMT14, which is quite significant. Similar observations about SGD were also reported in [263].

Despite this, AdaHessian achieves state-of-the-art performance for NMT with transformers. In particular, AdaHessian outperforms AdamW by 0.13 BLEU score on IWSLT14. Furthermore, the accuracy of AdaHessian on WMT14 is 28.52, which is 0.33 higher than that of AdamW. We also plot the training losses of AdamW and AdaHessian on IWSLT14/WMT14 in Figure C.3. As one can see, AdaHessian consistently achieves lower training loss. These improvements are quite significant for NMT, and importantly these are achieved even though AdaHessian directly uses the same β_1 and β_2, as well as the same number of warmup iterations as in AdamW.

4.4.4 Language Modeling

We report the language modeling results in Table 4.4, using the tensorized transformer proposed in [149]. Similar to NMT, note that the perplexity (PPL) of SGD is more than 57 points worse than AdamW on Wikitext-103. That is, similar to the NMT task, SGD performs worse than AdamW. However, AdaHessian achieves more than 1.8/1.0 better PPL than that of AdamW on PTB/Wikitext-103, respectively.

We also show the detailed training loss curves in Figure C.4. AdaHessian achieves consistently lower loss values than AdamW throughout the training process on both PTB and Wikitext-103. Similar to NMT, the β_1/β_2 as well as the warmup phase of AdaHessian are kept the same as AdamW.

4.4.5 Natural Language Understanding

We report the NLU results in Table 4.5, using the SqueezeBERT model [117] tested on GLUE datasets [234]. As can be seen, AdaHessian has better performance than AdamW on 5 out of 8 tasks. Particularly, on RTE and MPRC, AdaHessian achieves more than 1 point as compared to AdamW. On average, AdaHessian outperforms AdamW by 0.41 points. Note that similar to NMT and LM, except learning rate and block size, AdaHessian directly uses the same hyperparameters as AdamW. Interestingly, note that these results are better than those reported in SqueezeBERT [116], even though we only change the optimizer to AdaHessian instead of AdamW.

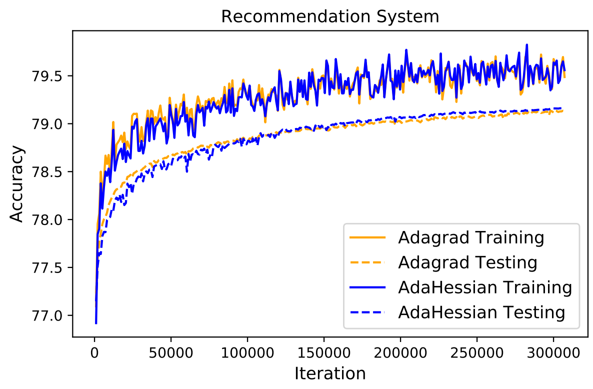

Figure 4.6: Training and Testing Accuracy curves of Adagrad and AdaHessian on Criteo Ad Kaggle dataset. As can be seen, the test accuracy of AdaHessian is better (0.032%) than that of Adagrad. This is quite significant for this task.

4.4.6 Recommendation System

We solely focus on modern recommendation systems, and in particular on the DLRM model widely adopted in industry [166]. These systems include a large embedding layer followed by a series of dense FC layers. In training, a sparse set of rows of the embedding layer is used and only those rows are updated. These rows do change from one iteration to the next. For such a sparse setting, we use Adagrad to update the embedding table, and we use AdaHessian to update the rest of the FC network in the experiments. (Pytorch currently does not support second-order backpropagation for the sparse gradient to the embedding.) AdaHessian uses the same hyperparameters for updating the embedding table as in the Adagrad experiment without tuning. The training and testing accuracy curves are reported in Figure 4.6. The testing accuracy of AdaHessian is 79.167%, which is 0.032% higher than Adagrad. It should be noted that this is a quite significant accuracy increase for Recommendation Systems [237].

Table 4.6: Robustness of AdamW and AdaHessian to the learning rate on IWSLT14. We scale the base learning rate used in Section 4.4.3. As can be seen, AdaHessian is much more robust to large learning rate variability as compared to AdamW.

LR Scaling	**0.5**	**1**	**2**	**3**	**4**	**5**	**6**	**10**
AdamW	**35.42** ± **.09**	35.66 ± .11	**35.37** ± **.07**	**35.18** ± **.07**	**34.79** ± **.15**	14.41 ± 13.25	0.41 ± .32	Diverge
AdaHessian	35.33 ± .10	**35.79** ± **.06**	35.21 ± .14	34.74 ± .10	34.19 ± .06	**33.78** ± **.14**	**32.70** ± **.10**	**32.48** ± **.83**

Table 4.7: Block Size effect of AdaHessian on IWSLT14. With various block sizes, the performance of AdaHessian is very stable and no worse than that of AdamW (35.66 ± .11).

Block Size	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**
AdaHessian	35.67 ± .10	35.66 ± .07	35.78 ± .07	35.77 ± .08	35.67 ± .08	**35.79** ± **.06**	35.72 ± .06	35.67 ± .11

4.5 Discussion

As reported in the previous section, AdaHessian achieves state-of-the-art performance on a wide range of tasks. Two important issues are the sensitivity of AdaHessian to the hyper-parameters of learning rate and block size. This is discussed next.

4.5.1 Learning Rate and Block Size Effects

Here, we explore the effects of the learning rate and block size b on AdaHessian. We first start with the effect of learning rate, and test the performance of AdaHessian and AdamW with different learning rates. The results are reported in Table 4.6 for IWSLT14 dataset, where we scale the original learning rate with a constant factor, ranging from 0.5 to 20 (the original learning rate is the same as in Section 4.4.3). It can be seen that AdaHessian is more robust to the large learning rates. Even with 10× learning rate scaling, AdaHessian still achieves 32.48 BLEU score, while AdamW diverges even with 6× learning rate scaling. This is a very desirable property of AdaHessian, as it results in reasonable performance for such a wide range of learning rates.

We also test the effect of the spatial averaging block size (parameter b in (4.10)). As a reminder, this parameter is used for spatially averaging the Hessian diagonal as illustrated in Figure 4.5. The sensitivity results are shown in Table 4.7 where we vary the block size from 1 to 128. While the best performance is achieved for the block size of 32, the performance variation for other block sizes is rather small. Moreover, all the results are still no worse than the result with AdamW.

4.5.2 AdaHessian Overhead

Here, we discuss and measure the overhead of AdaHessian. In terms of computational complexity, AdaHessian requires twice the flops as compared to SGD. This 2× overhead comes from the cost of computing the Hessian diagonal, when one Hutchinson step is performed per

Table 4.8: Comparison between AdaHessian theoretical and measured speed, as compared to Adam and SGD, tested on Cifar10. We also measured the speed up for different Hessian computation frequencies. As one can see, AdaHessian is *not* orders of magnitude slower than SGD, despite the widely-held incorrect belief about the efficiency of Hessian based methods. Furthermore, by increasing the Hessian computation frequency, the run time can improve from 3.23× to 1.45×, as compared to SGD for ResNet32. The real measurement is performed on one RTX Titan GPU.

Hessian Computation Frequency	**1**	**2**	**3**	**4**	**5**
Theoretical Per-iteration Cost (×SGD)	2×	1.5×	1.33×	1.25×	1.2×
ResNet20 (Cifar10)	92.13 ± .08	92.40 ± .04	92.06 ± .18	92.17 ± .21	92.16 ± .12
Measured Per-iteration Cost (×SGD)	2.42×	1.71×	1.47×	1.36×	1.28×
Measured Per-iteration Cost (×Adam)	2.27×	1.64×	1.42×	1.32×	1.25×
ResNet32 (Cifar10)	93.08 ± .10	92.91 ± .14	92.95 ± .17	92.93 ± .24	93.00 ± .10
Measured Per-iteration Cost (×SGD)	3.23×	2.12×	1.74×	1.56×	1.45×
Measured Per-iteration Cost (×Adam)	2.91×	1.96×	1.64×	1.48×	1.38×

optimization iteration. Each Hutchinson step requires computing one Hessian matvec (the $\mathbf{H}z$ term in (4.9)). This step requires one more gradient backpropagation, hence leading to twice the theoretical complexity.

We have also measured the actual runtime of AdaHessian in PyTorch on a single RTX Titan GPU machine, as reported in the second column of Table 4.8. For ResNet20, AdaHessian is 2.42× slower than SGD (and 2.27× slower than Adam). As one can see, AdaHessian is not orders of magnitude slower than first-order methods. The gap between the measured and theoretical speed is likely due to the fact that Pytorch [178] (and other existing frameworks) are highly optimized for first-order methods. Even then, if one considers the fact that SGD needs a lot of tuning, this overhead may not be large.

It is also possible to reduce the AdaHessian overhead. One simple idea is to reduce the Hutchinson calculation frequency from 1 Hessian matvec per iteration to every multiple iterations. For example, for a frequency of 2, we perform the Hutchinson step at every other optimization iteration. This reduces the theoretical computational cost to 1.5× from 2×. One can also further reduce the frequency to 5, for which this cost reduces to 1.2×.

We studied how such reduced Hutchinson calculation frequency approach would impact the performance. We report the results for training ResNet20/ResNet32 on the Cifar10 in Table 4.8, when we vary the Hutchinson frequency from 1 to 5. As one can see, there is a small performance variation, but the AdaHessian overhead significantly decreases as compared to SGD and Adam.

4.6 Conclusions

In this work, we proposed AdaHessian, an adaptive Hessian based optimizer. AdaHessian incorporates an approximate Hessian diagonal, with spatial averaging and momentum to precondition the gradient vector. This automatically rescales the gradient vector resulting in better descent directions. One of the key novelties in our approach is the incorporation spatial averaging for Hessian diagonal along with an exponential moving average in time. These enable us to smooth noisy local Hessian information which could be highly misleading.

We extensively tested AdaHessian on various datasets and tasks, using state-of-the-art models. These include IWSLT14 and WMT14 for neural machine translation, PTB and Wikitext-103 for language modeling, GLUE for natural language understanding, Cifar10 and ImageNet for image classification , and Criteo Ad Kaggle for recommendation system. AdaHessian consistently achieves comparable or higher generalization performance as compared to the highly tuned default optimizers used for these different tasks.

Stepping back, it is important for every work to state its limitations (in general, but in particular in this area). The current limitation of AdaHessian is that it is $2-3\times$ slower than first-order methods such as SGD and Adam. We briefly explored how this overhead could be reduced, but more work is needed in this area. However, AdaHessian consistently achieves comparable or better accuracy. For example, for LM task, AdaHessian achieves up to 2.7 better PPL, as compared to AdamW, which is significant for this task.

Finally, from a higher-level perspective, we should note that there has been significant development within second-order methods, both theory and practice, even though these methods were widely viewed as being inapplicable for machine learning even just a few years ago. Some examples include Hessian based model compression [135, 103, 69, 70], adversarial attacks [255], and studies of the loss landscape topology for different NN architectures [206, 254], to name just a few. AdaHessian is an important step in this area, and we expect that it will enable still further progress. We have open sourced AdaHessian and we hope that it would help this progress [110].

Part II

Machine Learning and Data Science Analysis

Chapter 5

Hessian-based Analysis of Large Batch Training and Robustness to Adversaries

5.1 Introduction

During the training of a Neural Network (NN), we are given a set of input data $\mathbf{x}$ with the corresponding labels y drawn from an unknown distribution $\mathcal{P}$. In practice, we only observe a set of discrete examples drawn from $\mathcal{P}$, and train the NN to learn this unknown distribution. This is typically a non-convex optimization problem, in which the choice of hyper-parameters would highly affect the convergence properties. In particular, it has been observed that using large batch size for training often results in convergence to points with poor convergence properties. The main motivation for using large batch is the increased opportunities for data parallelism which can be used to reduce training time [90, 82]. Recently, there have been several works that have proposed different methods to avoid the performance loss with large batch [90, 219, 257]. However, these methods do not work for all networks and datasets. This has motivated us to revisit the original problem and study how the optimization with large batch size affects the convergence behavior.

We first start by analyzing how the Hessian spectrum and gradient change during training for small batch and compare it to large batch size and then draw connection with robust training.

In particular, we aim to answer the following questions:

Q1 How is the training for large batch size different than small batch size? Equivalently, what is the difference between the local geometry of the neighborhood that the model converges when large batch size is used as compared to small batch?

A1 We backpropagate the second-derivative and compute its spectrum during training. The results show that despite the arguments regarding prevalence of saddle-points plaguing optimization [62, 81], that is actually not the problem with large batch size training, even

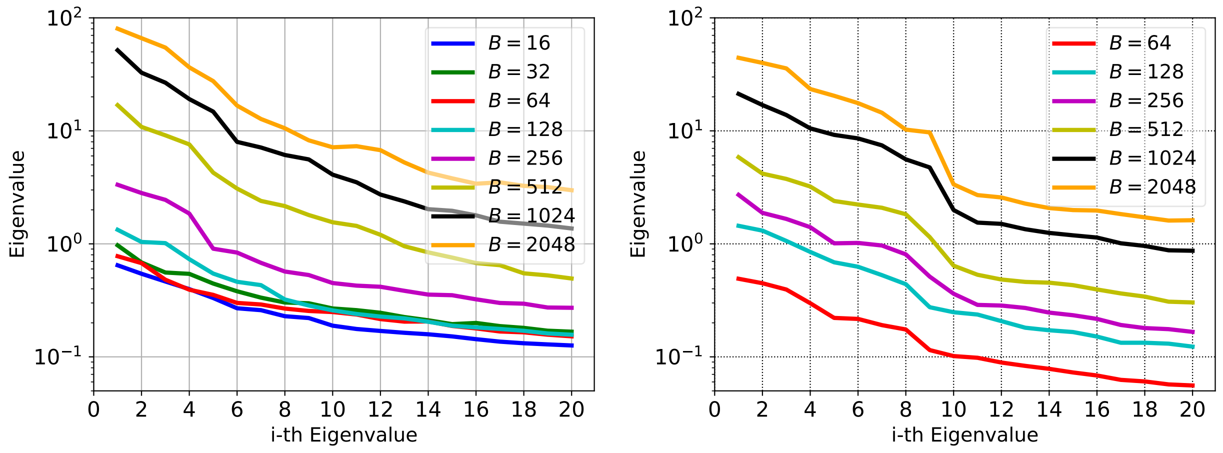

Figure 5.1: Top 20 eigenvalues of the Hessian is shown for C1 on CIFAR-10 (left) and M1 on MNIST (right) datasets. The spectrum is computed using power iteration with relative error of $1e^{-4}$.

when batch size is increased to the gradient descent limit. In [124], an approximate numerical method was used to approximate the maximal eigenvalue at a point. Here, by directly computing the spectrum of the true Hessian, we show that large batch size progressively gets trapped in areas with noticeably larger spectrum (and not just the dominant eigenvalue). For details please see Section 5.3, especially Figure 5.1, 5.2, and 5.4.

Q2 What is the connection between robust optimization and large batch size training? Equivalently, how does the batch size affect the robustness of the model to adversarial perturbation?

A2 We show that robust optimization is antithetical to large batch training, in the sense that it favors areas with small spectrum (aka flat minimas). We show that points converged with large batch size are significantly more prone to adversarial attacks as compared to a model trained with small batch size. Furthermore, we show that robust training progressively favors the opposite, leading to points with flat spectrum and robust to adversarial perturbation. We provide empirical and theoretical proof that the inner loop of the robust optimization, where we find the worst case, is a saddle-free optimization problem *almost everywhere*. Details are discussed in Section 5.4, especially Table 5.1, D.3, and Figure 5.4, 5.6.

Limitations: We believe it is critical for every paper to clearly state limitations. In this work, we have made an effort to avoid reporting just the best results, and repeated all the experiments at least three times and found all the findings to be consistent. Furthermore, we performed the tests on multiple datasets and multiple models, including a residual network, to avoid getting results that may be specific to a particular test. The main limitation is that we do not propose a solution for large batch training. We do offer analytical insights into the relationship between large batch and robust training, but we do not fully resolve the problem of large batch training. There have been several approaches to increasing batch size proposed so far [90, 219, 257], but they only work for particular cases and require extensive hyper-parameter tuning. We are performing an in-depth follow up study to use the results of this paper to better guide large batch size training.

5.2 Related Work

Deep neural networks have achieved good performance for a wide range of applications. The diversity of the different problems that a DNN can be used for, has been related to their efficiency in function approximation [162, 63, 134, 6]. However the work of [261] showed that not only the network can perform well on a real dataset, but it can also memorize randomly labeled data very well. Moreover, the performance of the network is highly dependent on the hyper-parameters used for training. In particular, recent studies have shown that Neural Networks can easily be fooled by imperceptible perturbations to input data [89]. Moreover, multiple studies have found that large batch size training suffers from poor generalization capability [90, 257].

Here we focus on the latter two aspects of training neural networks. [124] presented results showing that large batches converge to a "sharper minima". It was argued that even if the sharp minima has the same training loss as the flat one, small discrepancies between the test data and the training data can easily lead to poor generalization performance [124, 67]. The fact that "flat minimas" generalize well goes back to the earlier work of [108]. The authors relate flat minima to the theory of minimum description length [194], and proposed an optimization method to actually favor flat minimas. There have been several similar attempts to change the optimization algorithm to find "better" regions [66, 46]. For instance, [46] proposed entropy-SGD, which uses Langevin dynamics to augment the loss functional to favor flat regions of the "energy landscape". The notion of flat/sharpness does not have a precise definition. A detailed comparison of different metrics is discussed in [67], where the authors show that sharp minimas can also generalize well. The authors also argued that the sharpness can be arbitrarily changed by reparametrization of the weights. However, this won't happen when considering the same model and just changing the training hyper-parameters which is the case here. In [219, 220], the authors proposed that the training can be viewed as a stochastic differential equation, and argued that the optimum batch size is proportional to the training size and the learning rate.

As our results show, there is an interleaved connection by studying when NNs do not work well. [225, 89] found that they can easily fool a NN with very good generalization by slightly perturbing the inputs. The perturbation magnitude is most of the time imperceptible to human eye, but can completely change the networks prediction. They introduced an effective adversarial attack algorithm known as Fast Gradient Sign Method (FGSM). They related the vulnerability of the Neural Network to linear classifiers and showed that RBF models, despite achieving much smaller generalization performance, are considerably more robust

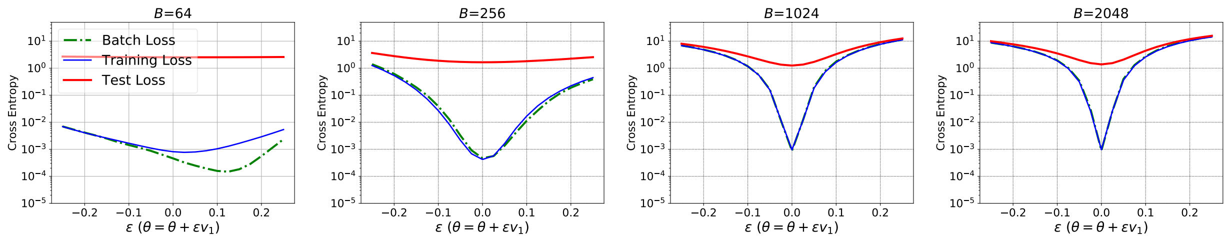

Figure 5.2: The landscape of the loss is shown along the dominant eigenvector, v_1, of the Hessian for C1 on CIFAR-10 dataset. Here ϵ is a scalar that perturbs the model parameters along v_1.

to FGSM attacks. The FGSM method was then extended in [130] to an iterative FGSM, which performs multiple gradient ascend steps to compute the adversarial perturbation. Adversarial attack based on iterative FGSM was found to be stronger than the original one step FGSM. Various defenses have been proposed to resist adversarial attacks [158, 88, 96, 20, 79]. We will later show that there is an interleaved connection between robustness of the model and the large batch size problem.

The structure of this paper is as follows: We present the results by first analyzing how the spectrum changes during training, and test the generalization performances of the model for different batch sizes in Section 5.3. In Section 5.4, we discuss details of how adversarial attack/training is performed. In particular, we provide theoretical proof that finding adversarial perturbation is a saddle-free problem under certain conditions, and test the robustness of the model for different batch sizes. Also, we present results showing how robust training affects the spectrum with empirical studies. Finally, in Section 5.5 we provide concluding remarks.

5.3 Large Batch, Generalization Gap and Hessian Spectrum

Setup: The architecture for the networks used is reported in Table D.2. In the text, we refer to each architecture by the abbreviation used in this table. Unless otherwise specified, each of the batch sizes are trained until a training loss of 0.001 or better is achieved. Different batches are trained under the same conditions, and no weight decay or dropout is used.

We first focus on large batch size training versus small batch and report the results for large batch training for C1 network on CIFAR-10 dataset, and M1 network on MNIST are shown in Table 5.1, and Table D.3, respectively. As one can see, after a certain point increasing batch size results in performance degradation on the test dataset. This is in line with results in the literature [124, 90].

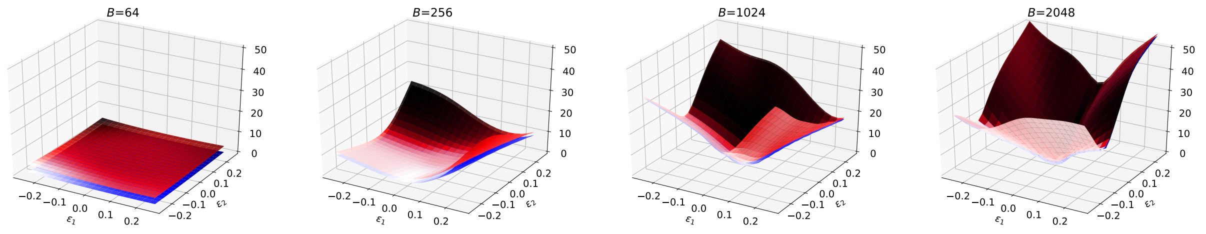

Figure 5.3: The landscape of the loss is shown when the C1 model parameters are changed along the first two dominant eigenvectors of the Hessian with the perturbation magnitude ϵ_1 and ϵ_2.

As discussed before, one popular argument about large batch size's poor generalization accuracy has been that large batches tend to get attracted to "sharp" minimas of the training loss. In [124] an approximate metric was used to measure curvature of the loss function for a given model parameter. Here, we directly compute the Hessian spectrum, using power iteration by back-propagating the matvec of the Hessian [155]. Unless otherwise noted, we continue the power iterations until a relative error of $1e^{-4}$ reached for each eigenvalue. The Hessian spectrum for different batches is shown in Figure 5.1. Moreover, the value of the dominant eigenvalue, denoted by λ_1^θ, is reported in Table 5.1, and Table 5.2, respectively (Additional result for MNIST tested using LeNet-5 is given in appendix. Please see Table D.3). From Figure 5.1, we can clearly see that for all the experiments, large batches have a noticeably larger Hessian spectrum both in the dominant eigenvalue as well as the rest of the 19 eigenvalues. However, note that curvature is a very local measure. It would be more informative to study how the loss functional behaves in a neighborhood around the point that the model has converged. To visually demonstrate this, we have plotted how the total loss changes when the model parameters are perturbed along the dominant eigenvector as shown in Figure 5.2, and Figure D.1 for C1 and M1 models, respectively. We can clearly see that the large batch size models have been attracted to areas with higher curvature for both the test and training losses.

This is reflected in the visual figures. We have also added a 3D plot, where we perturb the parameters of C1 model along both the first and second eigenvectors as shown in Figure 5.3. The visual results are in line with the numbers shown for the Hessian spectrum (see λ_1^θ) in Table 5.1, and Table D.3. For instance, note the value of λ_1^θ for the training and test loss for $B = 256,\ 2048$ in Table 5.1 and compare the corresponding results in Figure 5.3.

A recent argument has been that saddle-points in high dimension plague optimization for neural networks [62, 81]. We have computed the dominant eigenvalue of the Hessian along with the total gradient during training and report it in Figure 5.4. As we can see, large batch size progressively gets attracted to areas with larger spectrum, but it clearly does not get stuck in saddle points [138].

Table 5.1: Result on CIFAR-10 dataset using C1, C2 network. We show the Hessian spectrum of different batch training models, and the corresponding performances on adversarial dataset generated by training/testing dataset (testing result is given in parenthese).

	Batch	Acc.	λ_1^θ	$\lambda_1^{\mathbf{x}}$	$\|\nabla_{\mathbf{x}}\mathcal{J}\|$	Acc $\epsilon = 0.02$	Acc $\epsilon = 0.01$
C1 Cifar-10	16	100 (77.68)	0.64 (32.78)	2.69 (200.7)	0.05 (20.41)	48.07 (30.38)	72.67 (42.70)
	32	100 (76.77)	0.97 (45.28)	3.43 (234.5)	0.05 (23.55)	49.04 (31.23)	72.63 (43.30)
	64	100 (77.32)	0.77 (48.06)	3.14 (195.0)	0.04 (21.47)	50.40 (32.59)	73.85 (44.76)
	128	100 (78.84)	1.33 (137.5)	1.41 (128.1)	0.02 (13.98)	33.15 (25.2)	57.69 (39.09)
	256	100 (78.54)	3.34 (338.3)	1.51 (132.4)	0.02 (14.08)	25.33 (19.99)	50.10 (34.94)
	512	100 (79.25)	16.88 (885.6)	1.97 (100.0)	0.04 (10.42)	14.17 (12.94)	28.54 (25.08)
	1024	100 (78.50)	51.67 (2372)	3.11 (146.9)	0.05 (13.33)	8.80 (8.40)	23.99 (21.57)
	2048	100 (77.31)	80.18 (3769)	5.18 (240.2)	0.06 (18.08)	4.14 (3.77)	17.42 (16.31)
C2 Cifar-10	256	100 (79.20)	0.62 (28)	12.10 (704.0)	0.10 (41.95)	0.57 (0.38)	0.73 (0.47)
	512	100 (80.44)	0.75 (57)	4.82 (425.2)	0.03 (26.14)	0.34 (0.25)	0.54 (0.38)
	1024	100 (79.61)	2.36 (142)	0.523 (229.9)	0.04 (17.16)	0.27 (0.22)	0.46 (0.35)
	2048	100 (78.99)	4.30 (307)	0.145 (260.0)	0.50 (17.94)	0.18 (0.16)	0.33 (0.28)

5.4 Large Batch, Adversarial Attack and Robust training

We first give a brief overview of adversarial attack and robust training and then present results connecting these with large batch size training.

5.4.1 Robust Optimization and Adversarial Attack

Here we focus on white-box adversarial attack, and in particular the optimization-based approach both for the attack and defense. Suppose $\mathcal{M}(\theta)$ is a learning model (the neural network architecture), and $(\mathbf{x}, y)$ are the input data and the corresponding labels. The loss functional of the network with parameter θ on $(\mathbf{x}, y)$ is denoted by $\mathcal{J}(\theta, \mathbf{x}, y)$. For adversarial attack, we seek a perturbation $\Delta\mathbf{x}$ (with a bounded L_∞ or L_2 norm) such that it maximizes $\mathcal{J}(\theta, \mathbf{x}, y)$:

$$\max_{\Delta\mathbf{x}\in\mathcal{U}} \mathcal{J}(\theta, \mathbf{x} + \Delta\mathbf{x}, y), \tag{5.1}$$

where $\mathcal{U}$ is an admissibility set for acceptable perturbation (typically restricting the magnitude of the perturbation). A typical choice for this set is $\mathcal{U} = \mathbf{B}(\mathbf{x}, \epsilon)$, a ball of radius ϵ centered at $\mathbf{x}$. A popular method for approximately computing $\Delta\mathbf{x}$, is Fast Gradient Sign Method [89], where the gradient of the loss functional is computed w.r.t. inputs, and the perturbation is set to:

$$\Delta\mathbf{x} = \epsilon\,\mathrm{sign}(\frac{\partial J(\mathbf{x}, \theta)}{\partial\mathbf{x}}). \tag{5.2}$$

This is not the only attack method possible. Other approaches include an iterative FGSM method (FGSM-10)[130] or using other norms such as L_2 norm instead of L_∞ (We denote the L_2 method by L_2Grad in our results). Here we also use a second-order attack, where we use the Hessian w.r.t. input to precondition the gradient direction with second-order information; please see Table D.1 in Appendix for details.

One method to defend against such adversarial attacks, is to perform robust training [225, 150]:

$$\min_{\theta} \max_{\Delta\mathbf{x}\in\mathcal{U}} \mathcal{J}(\theta, \mathbf{x}+\Delta\mathbf{x}, y). \tag{5.3}$$

Solving this min-max optimization problem at each iteration requires first finding the worst adversarial perturbation that maximizes the loss, and then updating the model parameters θ for those cases. Since adversarial examples have to be generated at every iteration, it would not be feasible to find the exact perturbation that maximizes the objective function. Instead, a popular method is to perform a single or multiple gradient ascents to approximately compute $\Delta\mathbf{x}$. After computing $\Delta\mathbf{x}$ at each iteration, a typical optimization step (variant of SGD) is performed to update θ.
Next we show that solving the maximization part is actually a saddle-free problem *almost everywhere*. This property assures us that the Hessian w.r.t input does not have negative eigenvalue which allows us to use CG for performing Newton solver for our 2nd order adversarial perturbation tests in Section 5.4.4. [1]

5.4.2 Adversarial perturbation: A saddle-free problem

Recall that our loss functional is $\mathcal{J}(\theta; \mathbf{x}, y)$. We make following assumptions for the model to help show our theoretical result,

Assumption 4. *We assume the model's activation functions are strictly ReLu activation, and all layers are either convolution or fully connected. Here, Batch Normalization layers are accepted. Note that even though the ReLu activation has discontinuity at origin, i.e. $x = 0$, ReLu function is twice differentiable almost everywhere.*

The following theorem shows that the problem of finding an adversarial perturbation that maximized $\mathcal{J}$, is a saddle-free optimization problem, with a Positive-Semi-Definite (PSD) Hessian w.r.t. input almost everywhere. For details on the proof please see Appendix. D.1.

Theorem 6. *With Assumption. 4, for a DNN, its loss functional $\mathcal{J}(\boldsymbol{\theta}, \boldsymbol{x}, y)$ is a saddle-free function w.r.t. input $\boldsymbol{x}$ almost everywhere, i.e.*

$$\frac{\nabla^2 \mathcal{J}(\boldsymbol{\theta}, \boldsymbol{x}, y)}{\nabla \boldsymbol{x}^2} \succeq 0.$$

From the proof of Theorem 6, we could immediately get the following proposition of DNNs:

[1]This results might also be helpful for finding better optimization strategies for GANS.

Table 5.2: Result on CIFAR-100 dataset using CR network. We show the Hessian spectrum of different batch training models, and the corresponding performances on adversarial dataset generated by training/testing dataset (testing result is given in parenthese).

Batch	Acc.	λ_1^θ	Acc $\epsilon = 0.02$	Acc $\epsilon = 0.01$
64	99.98 (70.81)	0.022 (10.43)	61.54 (34.48)	78.57 (39.94)
128	99.97 (70.9)	0.055 (26.50)	58.15 (33.73)	77.41 (38.77)
256	99.98 (68.6)	1.090 (148.29)	39.96 (28.37)	66.12 (35.02)
512	99.98 (68.6)	1.090 (148.29)	40.48 (28.37)	66.09 (35.02)

Table 5.3: Accuracy of different models across different adversarial samples of MNIST, which are obtained by perturbing the original model $\mathbf{M}_{ORI}$

	$\mathbf{D}_{clean}$	$\mathbf{D}_{FGSM}$	$\mathbf{D}_{FGSM10}$	$\mathbf{D}_{L_2GRAD}$	$\mathbf{D}_{FHSM}$	$\mathbf{D}_{L_2HESS}$	MEAN of Adv
$\mathbf{M}_{ORI}$	99.32	60.37	77.27	14.32	82.04	33.21	53.44
$\mathbf{M}_{FGSM}$	99.49	96.18	97.44	63.46	97.56	83.33	87.59
$\mathbf{M}_{FGSM10}$	**99.5**	96.52	**97.63**	66.15	97.66	84.64	88.52
$\mathbf{M}_{L_2GRAD}$	98.91	**96.88**	97.39	**86.23**	**97.66**	**92.56**	**94.14**
$\mathbf{M}_{FHSM}$	99.45	94.41	96.48	52.67	96.89	77.58	83.60
$\mathbf{M}_{L_2HESS}$	98.72	95.02	96.49	77.18	97.43	90.33	91.29

Proposition 1. *Based on Theorem 6 with Assumption 4, if the input* $\boldsymbol{x} \in \boldsymbol{R}^d$ *and the number of the output class is* c*, i.e.* $y \in \{1, 2, 3 \ldots, c\}$*, then the Hessian of DNNs w.r.t. to* $\boldsymbol{x}$ *is almost a rank* c *matrix almost everywhere; see Appendix D.1 for details.*

5.4.3 Large Batch Training and Robustness

Here, we test the robustness of the models trained with different batches to an adversarial attack. We use Fast Gradient Sign Method for all the experiments (we did not see any difference with FGSM-10 attack). The adversarial performance is measured by the fraction of correctly classified adversarial inputs. We report the performance for both the training and test datasets for different values of $\epsilon = 0.02, 0.01$ (ϵ is the metric for the adversarial perturbation magnitude in L_∞ norm). The performance results for C1, and C2 models on CIFAR-10, CR model on CIFAR-100, are reported in the last two columns of Tables 5.1,and 5.2 (MNIST results are given in appendix, Table D.3). The interesting observation is that for all the cases, large batches are considerably more prone to adversarial attacks as compared to small batches. This means that not only the model design affects the robustness of the model, but also the hyper-parameters used during optimization, and in particular the properties of the point that the model has converged to.

From this result, there seems to be a strong correlation between the spectrum of the Hessian w.r.t. θ and how robust the model is. However, we want to emphasize that in

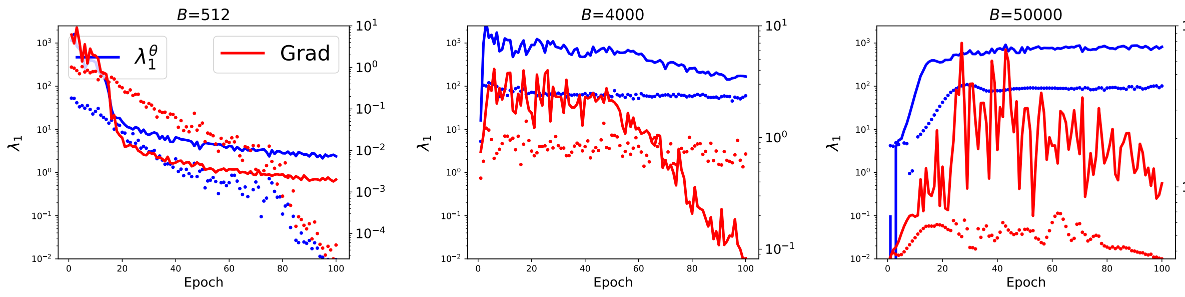

Figure 5.4: Changes in the dominant eigenvalue of the Hessian w.r.t weights and the total gradient is shown for different epochs during training. Note the increase in λ_1^θ (blue curve) for large batch v.s. small batch. In particular, note that the values for total gradient along with the Hessian spectrum show that large batch does not get "stuck" in saddle points, but areas in the optimization landscape that have high curvature. More results are shown in Figure D.10. The dotted points show the corresponding results when using robust optimization, which makes the solver stay in areas with smaller spectrum.

general there is no correlation between the Hessian w.r.t. weights and the robustness of the model w.r.t. the input. For instance, consider a two variable function $\mathcal{J}(\boldsymbol{\theta}, \mathbf{x})$ (we treat θ and $\mathbf{x}$ as two single variables), for which the Hessian spectrum of θ has no correlation to robustness of $\mathcal{J}$ w.r.t. $\mathbf{x}$. This can be easily demonstrated for a least squares problem, $L = \|\theta\mathbf{x} - \mathbf{y}\|_2^2$. It is not hard to see the Hessian of θ and $\mathbf{x}$ are, $\mathbf{x}\mathbf{x}^T$ and $\theta\theta^T$, respectively. Therefore, in general we cannot link the Hessian spectrum w.r.t. weights to robustness of the network. However, the numerical results for all the neural networks show that models that have higher Hessian spectrum w.r.t. θ are also more prone to adversarial attacks. A potential explanation for this would be to look at how the gradient and Hessian w.r.t. input (i.e. $\mathbf{x}$) would change for different batch sizes. We have computed the dominant eigenvalue of this Hessian using power iteration for each individual input sample for both training and testing datasets. Furthermore, we have computed the norm of the gradient w.r.t. $\mathbf{x}$ for these datasets as well. These two metrics are reported in $\lambda_1^{\mathbf{x}}$, and $\|\nabla_x \mathcal{J}\|$; see Table 5.1 for details. The results on all of our experiments show that these two metrics actually do not correlate with the adversarial accuracy. For instance, consider C1 model with $B = 512$. It has both smaller gradient and smaller Hessian eigenvalue w.r.t. $\mathbf{x}$ as compared to $B = 32$, but it performs acidly worse under adversarial attack. One possible reason for this could be that the decision boundaries for large batches are less stable, such that with small adversarial perturbation the model gets fooled.

Table 5.4: Accuracy of different models across different samples of CIFAR-10, which are obtained by perturbing the original model $\mathbf{M}_{ORI}$

	$\mathbf{D}_{clean}$	$\mathbf{D}_{FGSM}$	$\mathbf{D}_{FGSM10}$	$\mathbf{D}_{L_2GRAD}$	$\mathbf{D}_{FHSM}$	$\mathbf{D}_{L_2HESS}$	MEAN of Adv
$\mathbf{M}_{ORI}$	**79.46**	15.25	4.46	12.37	29.64	22.93	16.93
$\mathbf{M}_{FGSM}$	71.82	63.05	63.44	57.68	**66.04**	62.36	62.51
$\mathbf{M}_{FGSM10}$	71.14	**63.32**	**63.88**	**58.25**	65.95	**62.70**	**62.82**
$\mathbf{M}_{L_2GRAD}$	63.52	59.33	59.73	57.35	60.44	58.98	59.16
$\mathbf{M}_{FHSM}$	74.34	47.65	43.95	38.45	62.75	55.77	49.71
$\mathbf{M}_{L_2HESS}$	71.59	50.05	46.66	42.95	62.87	58.42	52.19

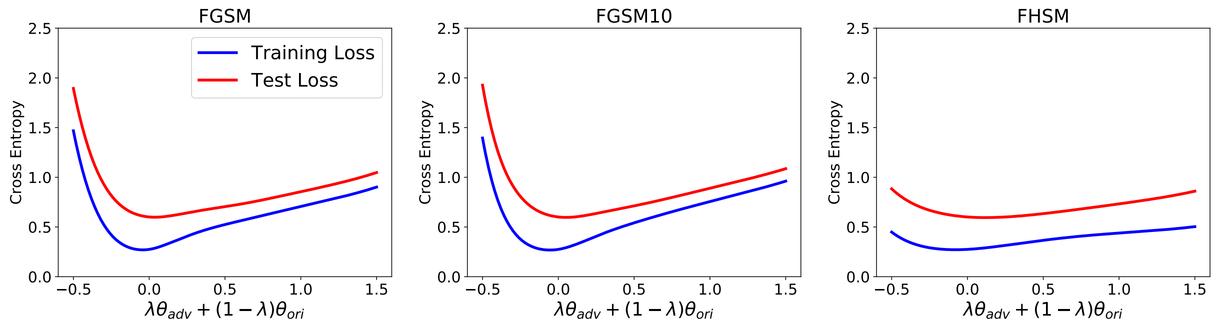

Figure 5.5: 1-D Parametric plot for C3 model on CIFAR-10. We interpolate between parameters of $\mathbf{M}_{ORI}$ and $\mathbf{M}_{ADV}$, and compute the cross entropy loss on the y-axis.

5.4.4 Adversarial Training and Hessian Spectrum

In this part, we study how the Hessian spectrum and the landscape of the loss functional change after adversarial training is performed. Here, we fix the batch size (and all other optimization hyper-parameters) and use five different adversarial training methods as described in Section 5.4.1.

For the sake of clarity let us denote $\mathbf{D}$ to be the test dataset which can be the original clean test dataset or one created by using an adversarial method. For instance, we denote $\mathbf{D}_{FGSM}$ to be the adversarial dataset generated by FGSM, and $\mathbf{D}_{clean}$ to be the original clean test dataset.

Setup: For the MNIST experiments, we train a standard LeNet on MNIST dataset [27] (using M1 network). For the original training, we set the learning rate to 0.01 and momentum to 0.9, and decay the learning rate by half after every 5 epochs, for a total of 100 epochs. Then we perform an additional five epochs of adversarial training with a learning rate of 0.01. The perturbation magnitude, ϵ, is set to 0.1 for L_∞ attack and 2.8 for L_2 attack. We also present results for C3 model [35] on CIFAR-10, using the same hyper-parameters, except that the training is performed for 100 epochs. Afterwards, adversarial training is performed for a subsequent 10 epochs with a learning rate of 0.01 and momentum of 0.9 (the

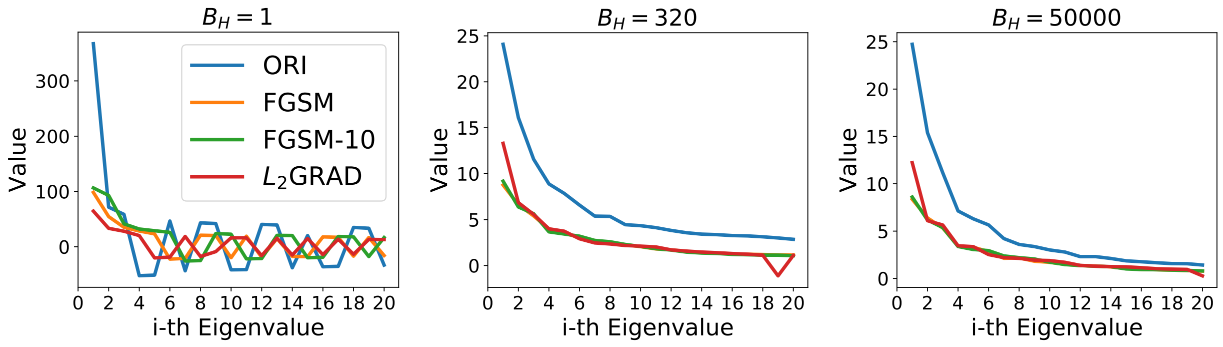

Figure 5.6: Spectrum of the sub-sampled Hessian of the loss functional w.r.t. weights. The results are computed for different batch sizes, which are randomly chosen, of $B = 1,\ 320,\ 50000$ of C1.

learning rate is decayed by half after five epochs). Furthermore, the adversarial perturbation magnitude is set to $\epsilon = 0.02$ for L_∞ attack and 1.2 for L_2 attack [211].

The results are shown in Table 5.3, 5.4. We can see that after adversarial training the model becomes more robust to these attacks. Note that the accuracy of different adversarial attacks varies, which is expected since the various strengths of different attack method. In addition, all adversarial training methods improve the robustness on adversarial dataset, though they lose some accuracy on $\mathbf{D}_{clean}$, which is consistent with the observations in [89]. As an example, consider the second row of Table 5.3 which shows the results when FGSM is used for robust training. The performance of this model when tested against the L_2GRAD attack method is 63.46% as opposed to 14.32% of the original model ($\mathbf{M}_{ORI}$). The rest of the rows show the results for different algorithms. In Section D.4, we give further discussion about the performances of first-order and second-order attacks.

The main question here is how the landscape of the loss functional is changed after these robust optimizations are performed? We first show a 1-D parametric interpolation between the original model parameters θ and that of the robustified models, as shown in Figure 5.5 (see Figure D.5 for all cases) and D.4. Notice the robust models are at a point that has smaller curvature as compared to the original model. To exactly quantify this, we compute the spectrum of the Hessian as shown in Figure 5.6, and D.6. Besides the full Hessian spectrum, we also report the spectrum of sub-sampled Hessian. The latter is computed by randomly selecting a subset of the training dataset. We denote the size of this subset as B_H to avoid confusion with the training batch size. In particular, we report results for $B_H = 1$ and $B_H = 320$. There are several important observations here. First, notice that the spectrum of the robust models is noticeably smaller than the original model. This means that the min-max problem of (5.3) favors areas with lower curvature. Second, note that even though the total Hessian shows that we have converged to a point with positive curvature (at least based on the top 20 eigenvalues), but that is not necessarily the case when we look at individual samples (i.e. $B_H = 1$). For a randomly selected batch of $B_H = 1$, we see that we have actually converged to a point that has both positive and negative curvatures, with a non-zero gradient (meaning it is not a saddle point). To the best of our knowledge this is a new finding, but one that is expected as SGD optimizes the expected loss instead of individual ones.

Now going back to Figure 5.4, we show how the spectrum changes during training when we use robust optimization. We can clearly see that with robust optimization the solver is pushed to areas with smaller spectrum. This is a very interesting finding and shows the possibility of using robust training as a systematic means to bias the solver to avoid *sharp* minimas. A preliminary result is shown in Table D.4, where we can see that robust optimization performs better for large batch size training as opposed to the baseline, or when we use the method proposed by [90]. However, we emphasize that the goal is to perform analysis to better understand the problems with large batch size training. More extensive tests are needed before one could claim that robust optimization performs better than other methods.

5.5 Conclusion

In this work, we studied NNs through the lens of the Hessian operator. In particular, we studied large batch size training and its connection with stability of the model in the presence of white-box adversarial attacks. By computing the Hessian spectrum we provided several points of evidence that show that large batch size training tends to get attracted to areas with higher Hessian spectrum. We reported the eigenvalues of the Hessian w.r.t. whole dataset, and plotted the landscape of the loss when perturbed along the dominant eigenvector. Visual results were in line with the numerical values for the spectrum. Our empirical results show that adversarial attacks/training and large batches are closely related. We provided several empirical results on multiple datasets that show large batch size training is more prone to adversarial attacks (more results are provided in the supplementary material). This means that not only is the model design important, but also that the optimization hyper-parameters can drastically affect a network's robustness. Furthermore, we observed that robust training is antithetical to large batch size training, in the sense that it favors areas with noticeably smaller Hessian spectrum w.r.t. θ.

The results show that the robustness of the model does not (at least directly) correlate with the Hessian w.r.t. $\mathbf{x}$. We also found that this Hessian is actually a PSD matrix, meaning that the problem of finding the adversarial perturbation is actually a saddle-free problem *almost everywhere* for cases that satisfy this criterion 4. Furthermore, we showed that even though the model may converge to an area with positive curvature when considering all of the training dataset (i.e. total loss), if we look at individual samples then the Hessian can actually have significant negative eigenvalues. From an optimization viewpoint, this is due to the fact that SGD optimizes the expected loss and not the individual per sample loss.

Chapter 6

PyHessian: Neural Networks Through the Lens of the Hessian

6.1 Introduction

Residual neural networks [104] (ResNets) are widely used Neural Networks (NNs) for various learning tasks. The two main architectural components of ResNets are residual connections [104] and Batch Normalization (BN) layers [118]. However, going beyond motivating stories to characterize precisely when and why these two popular architectural ingredients help or hurt training/generalization—especially in terms of *measurable* properties of the model—is still largely unsolved. Relatedly, characterizing whether other suggested architectural changes will help or hurt training/generalization is still done in a largely ad hoc manner. For example, it is often motivated by plausible but untested intuitions, and it is not characterized in terms of measurable properties of the model.

In this work, we present and apply PyHessian, an open source scalable framework with which one can directly analyze Hessian information, i.e., second-derivative information w.r.t. model parameters, in order to address these and related questions. PyHessian computes Hessian information by applying known techniques from Numerical Linear Algebra (NLA) [12, 86, 142] and Randomized NLA (RandNLA) [151, 72, 71, 251, 10, 230] (that are approximate but come with rigorous theory). PyHessian enables computing Hessian information—including top Hessian eigenvalues, Hessian trace, and Hessian eigenvalue spectral density (ESD), and it supports distributed implementation—allowing distributed-memory execution on both cloud (e.g., AWS, Google Cloud) and supercomputer systems, for fast and efficient Hessian computation.

As an application of PyHessian, we use it to analyze the impact of residual connections and BN on the trainability of NNs, leading to new insights. In more detail, our main contributions are the following:

- We introduce PyHessian, a new framework for direct and efficient computation of Hessian information, including the top eigenvalue, the trace, and the full ESD [111]. We also apply

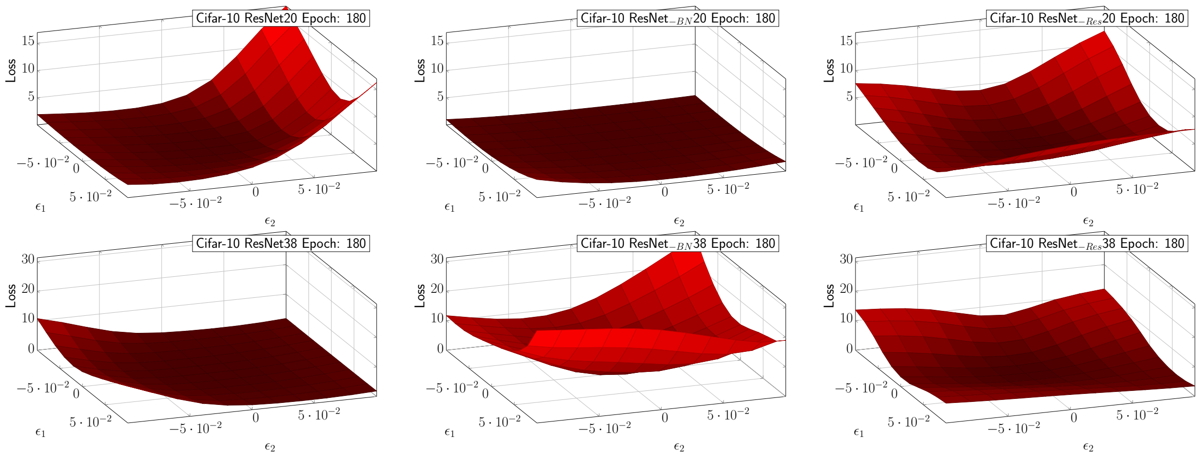

Figure 6.1: The parametric loss landscapes of ResNet20 (top) and ResNet38 (bottom) on Cifar-10 is plotted by perturbing the model parameters at the end of training across the first and second Hessian eigenvector. Results for the original ResNet architecture (left), ResNet without BN (middle; denoted as ResNet$_{-BN}$), and ResNet without residual connection (right; denoted as ResNet$_{-Res}$). It can be clearly seen that removing BN from ResNet20 actually leads to a smoother loss landscape, which is opposite to the common belief that adding BN leads to a smoother loss landscape [206]. We only observed the claimed smoothness property for the deeper ResNet38 model (second row). This smoothness can be quantified by measuring the trace of the Hessian operator, reported in Figure 6.2, as well as the full Hessian ESD, shown in Figure 6.3 and E.9. We also visualize the loss landscape throughout training for different epochs as shown in Figure E.14 and E.16, which provide further evidence. Models trained on Cifar-100 also exhibit a similar behavior (as shown in Figure E.18, E.19 and E.20).

PyHessian to study how residual connections and BN affect training.

• We observe that removing BN layers from ResNet (denoted below as ResNet$_{-BN}$) leads to **rapid increase of the Hessian spectrum** (the top eigenvalue, the trace, and the ESD support range). This increase is significantly more rapid for deeper models. See Figure 6.2, E.7, E.8, and E.9 on Cifar-10 as well as Figure E.5, E.11, E.12, and E.13 on Cifar-100.

• We observe that, for shallower networks (ResNet20), removing the BN layer results in a flatter Hessian spectrum, as compared to standard ResNet20 with BN. See Figure 6.2 and 6.3 on Cifar-10 and Figure E.5 and E.11 on Cifar-100. This observation is the **opposite of the common belief** that the addition of BN layers make the loss landscape smoother (which we observe to hold only for deeper networks).

• We observe that, for deeper networks (in our case, ResNet32/38), removing BN results in converging to sharper local minima, as compared to ResNet with BN. See Figure 6.2, 6.3, E.15 and E.16 on Cifar-10 as well as Figure E.5, E.11, E.19 and E.20 on Cifar-100.

• We show that removing residual connections from ResNet generally makes the top eigenvalue, the trace, and the Hessian ESD support range increase slightly. This increase is consistent for both shallower and deeper models (ResNet20/32/38/56). See Figure 6.2, 6.3, E.10, E.14, E.15, E.16, and E.17 on Cifar-10 as well as Figure E.5, E.11, E.18, E.19, and E.20 on Cifar-100.

• We perform Hessian analysis for different stages of ResNet models (details in Section 6.4.1), and we find that generally **BN is more important for the final stages** than for earlier stages. In particular, removing BN from the last stage significantly degrades testing performance, with a strong correlation with the Hessian trace. See the comparison between the orange curve and the blue curve in Figure 6.4 and E.4, the accuracy reported in Table 6.2 on Cifar-10 (Figure E.6, and the accuracy reported on Cifar-100 in Table E.2).

6.2 Related work

Here, we review work related to Hessian-based analysis for NN training and inference, as well as work that studies the impact of different architectural components on the topology of the NN loss landscape.

Hessian and Large-scale Hessian Computation: Hessian-based analysis/computation is widely used in scientific computing. However, due to the (incorrect) belief that Hessian-based computations are infeasible for large NN problems, the majority of work in ML (except for quite small problems) performs only first-order analysis.[1] However, using implicit or matrix-free methods, *it is not even necessary to form the Hessian matrix explicitly in order to extract second-order information.* Instead, it is possible to use stochastic methods from RandNLA to extract this information, without explicitly forming the Hessian matrix. For example, [12, 10] proposed fast algorithms for trace computation; and [142, 230] provided efficient randomized algorithms to estimate the ESD of a positive semi-definite matrix. These algorithms only require an oracle for computing the product of the Hessian matrix with a given random vector. *It is possible to compute this so-called "matvec" and extract Hessian information without explicitly forming the Hessian [16, 153]. In particular, using the so-called R-operator, the Hessian matvec can be computed with the same computational graph used for backpropagating the gradient [153].*

Hessian eigenvalues of small NN models were analyzed [202, 203]; and the work of [181] studied the geometry of NN loss landscapes by computing the distribution of Hessian eigenvalues at critical points. More recently, [251] used a deflated power-iteration method to compute the top eigenvalues for deep NNs during training. Moreover, the work of [84] measured the Hessian ESD, based on the Stochastic Lanczos algorithm of [142, 230]. Here, we

[1]The naïve view arises since the Hessian matrix is of size (say) $m \times m$. Thus, like most linear algebra computations, exact full spectral computations (which are sufficient but never necessary) cost $\mathcal{O}(m^3)$ time.

extend the analysis of [84, 251] by studying how the depth of the NN model as well as its architecture affect the Hessian spectrum (in terms of top eigenvalue, trace, and full ESD). Furthermore, we also perform block diagonal Hessian spectrum analysis, and we observe a fine-scale relationship between the Hessian spectrum and the impact of adding/removing residual connections and BN.

Hessian-based analysis has also been used in the context of NN training and inference. For example, [136] analytically computes Hessian information for a single linear layer and uses the Hessian spectrum to determine the optimal learning rate to accelerate training. In [135], the authors approximated the Hessian as a diagonal operator and used the inverse of this diagonal matrix to prune NN parameters. Subsequently, [103] used the inverse of the full Hessian matrix to develop an "Optimal Brain Surgeon" method for pruning NN parameters. The authors argued that a diagonal approximation may not be very accurate, as off-diagonal elements of the Hessian are important; and they showed that capturing these off-diagonal elements does indeed lead to better performance, as compared to [135]. In the recent work of [68], a layer-wise pruning method was proposed. This restricts the Hessian computations to each layer, and it provides bounds on the performance drop after pruning. More recently, [70, 214, 69] proposed a Hessian-based method for quantizing[2] NN models, achieving significantly better performance, as compared to first-order based methods.

(Quasi-)Newton (second-order) methods [2, 64, 180, 183, 186, 5, 26] have been extensively explored for convex optimization problems [28]. In particular, in the seminal work of [170, 145], a Quasi-Newton method was proposed to accelerate first-order based optimization methods. The idea is to precondition the gradient vector with the inverse of the Hessian. However, instead of directly using the Hessian, a series of approximate rank-1 updates are used instead. Follow up work of [209] extended this method and proposed a stochastic BFGS algorithm. More recently, the work of [25] proposed an adaptive batch size Limited-memory BFGS method [145] for large-scale machine learning problems; and an adaptive batch size method based on measuring directly the spectrum of the Hessian has been proposed [253] for large-scale NN training.

Hessian-based methods have also been explored for non-convex problems, including trust-region (TR) [57], cubic regularization (CR) [169], and its adaptive variant (ARC) [39, 40]. For these problems, [32, 77, 197, 247] provide sketching/sampling techniques for Newton methods, where guarantees are established for sampling size and convergence rates; and [245, 247, 244, 252] show that sketching/sampling methods can significantly reduce the need for data in approximate Hessian computation.

One important concern for applying second-order methods to training is the cost of computing Hessian information at every iteration. The work of [154] proposed the so-called Kronecker-Factored Approximations (K-FAC) method, which approximates the Fisher information matrix into a Kronecker product. However, the approach comes with several new hyperparameters, which can actually be more expensive to tune, compared to first-order

[2]Quantization is a process in which the precision of the parameters is reduced from single precision (32-bits) to a lower precision (such as 8-bits).

methods [148].

A major limitation in most of this prior work is that tests are typically restricted to small/simple NN models that may not be representative of NN workloads that are encountered in practice. This is in part due to the lack of a scalable and easily programmable framework that could be used to test second-order methods for a wide range of state-of-the-art models. Addressing this is the main motivation behind our development of PyHessian, which is released as open-source software and is available to researchers [111]. In this paper, we illustrate how PyHessian can be used for analyzing the NN behaviour during training, even for very deep state-of-the-art models. Future work includes using this framework for second-order based optimization, by testing it on modern NN models, as well as fairly gauging the benefit that may arise from such methods, in light of the cost for any extra hyperparameter tuning that may be needed [148].

Residual Connections and Batch Normalization: Residual connections [104] and BN [118] are two of the most important ingredients in modern convolutional NNs. There have been different hypothesis offered for why these two components help training/generalization. First, the original motivation for residual connections was that they allow gradient information to flow to earlier layers of the NN, thereby reducing the vanishing gradient problem during training. The empirical study of [141] found that deep NNs with residual connections exhibit a significantly smoother loss landscape, as compared to models without residual connections. This was achieved by the so-called filter-normalized random direction method to plot 3D loss landscapes, i.e., not through direct analysis of the Hessian spectrum. This result is interesting, but it is hard to draw conclusions with perturbations in two directions, for a model that has millions of parameters (and thus millions of possible perturbation directions).

Second, the original motivation for why BN helps training/generalization was originally attributed to reducing the so-called Internal Covariance Shift (ICS) [118]. However, this was disputed in the recent study of [206]. In particular, the work of [206] used first-order analysis to analyze the loss landscape, and found that adding a BN layer results in a smoother loss landscape. Importantly, they found that adding BN does not reduce the so called ICS. Again, while interesting, such first-order analysis may not fully capture the topology of the landscape; and, as we will show with our second-order analysis, **this smoothness claim is not correct in general**.

The work of [206] also performed an interesting theoretical analysis, showing a connection between adding the BN layer and the Lipschitz constant of the gradient (i.e., the top Hessian eigenvalue). It was argued that adding the BN layer leads to a smaller Lipschitz constant. However, the theoretical analysis is only valid for per-layer Lipschitz constant, as it ignores the complex interaction between different layers. It cannot be extended to the Lipschitz constant of the entire model (and, as we will show, this result does not hold for shallow networks).

6.3 Methodology

For a supervised learning problem, we seek to minimize:

$$\min_{\theta} L(\theta) = \frac{1}{N}\sum_{i=1}^{N} l(M(x_i), y_i, \theta), \tag{6.1}$$

where $\theta \in \mathbf{R}^m$ is the learnable weight parameter, $l(M(x), y, \theta)$ is the loss function, (x, y) is the input pair, M is the NN architecture, and N is the size of training data. Below we first discuss how PyHessian computes the second-order statistics, and we then discuss the impact of architectural components on the trainability of the model.

6.3.1 Neural Network Hessian Matvec

For a NN with m parameters, the gradient of the loss w.r.t. model parameters is a vector

$$\frac{\partial L}{\partial \theta} = g_\theta \in \mathbf{R}^m,$$

and the second derivative of the loss is a matrix,

$$H = \frac{\partial^2 L}{\partial \theta^2} = \frac{\partial g_\theta}{\partial \theta} \in \mathbf{R}^{m\times m},$$

commonly called the Hessian. A typical NN model involves millions of parameters, and thus even forming the Hessian is computationally infeasible. However, it is possible to compute properties of the Hessian spectrum *without* explicitly forming the Hessian matrix. Instead, all we need is an oracle to compute the application of the Hessian to a random vector v. This can be achieved by observing the following:

$$\frac{\partial g_\theta^T v}{\partial \theta} = \frac{\partial g_\theta^T}{\partial \theta} v + g_\theta^T \frac{\partial v}{\partial \theta} = \frac{\partial g_\theta^T}{\partial \theta} v = Hv. \tag{6.2}$$

Here, the first equality is the chain rule, the second is due to the independence of v to θ, and the third equality is the definition of the Hessian. Importantly, note that the cost of this Hessian matrix-vector multiply (hereafter referred to as Hessian matvec) is the same as one gradient backpropagation. Having this oracle, we can easily compute the top k Hessian eigenvalues using power iteration [251]; see Algorithm 10. However, for a typical NN with millions of parameters, the top eigenvalues may not be representative of how the loss landscape behaves. Therefore, we also compute the trace and ESD of the Hessian, as described below.

6.3.2 Hutchinson Method for Hessian Trace Computation

The trace of the Hessian can be computed using RandNLA, and in particular with Hutchinson's method [12, 10] for the fast computation of the trace, using only Hessian matvec computations (as given in (6.2)). In particular, since we are interested in the Hessian, i.e., a symmetric matrix, suppose we have a random vector v, whose components are i.i.d. sampled from a Rademacher distribution (or Gaussian distribution with mean 0 and variance 1). Then, we have the identity

$$\begin{aligned} Tr(H) = Tr(HI) = Tr(H\mathbf{E}[vv^T]) = \mathbf{E}[Tr(Hvv^T)] \\ = \mathbf{E}[v^T Hv], \end{aligned} \tag{6.3}$$

where I is the identity matrix of appropriate size. That is, the trace of H can be estimated by computing $\mathbf{E}[v^T Hv]$, where we compute the expectation by drawing multiple random samples. Note that Hv can be efficiently computed from (6.2), and then $v^T Hv$ is simply a dot product between the Hessian matvec and the original vector v. See Algorithm 11 for a description.

6.3.3 Full Eigenvalue Spectral Density

To provide finer-grained information on the Hessian spectrum than is provided by the top eigenvalues or the trace, we need to compute the full empirical spectral density (ESD) of the Hessian eigenvalues, defined as

$$\phi(t) = \frac{1}{m} \sum_{i=1}^{m} \delta(t - \lambda_i), \tag{6.4}$$

where $\delta(\cdot)$ is the Dirac distribution and λ_i is the i^{th} eigenvalue of H, in descending order.

Algorithm 6 Stochastic Lanczos Quadrature for ESD Computation

```
1: Input:
       - Parameter: $\theta$
       - Degree m and $n_v$
2: Compute the gradient of $\theta$ by backpropagation, i.e., compute $g_\theta = \frac{dL}{d\theta}$.
3: for i = $1, 2, \dots n_v$ do
4:     Draw a random vector $v$ from N(0,1) and normalize it (same dimension as $\theta$).
5:     Get the tridiagonal matrix $T$ through Lanczos algorithm.
6:     Compute $\tau_k^{(i)}$ and $\tilde{\lambda}_k^{(i)}$ from $T$
7:     $\phi_\sigma^{z_i} = \sum_{k=1}^{q} \tau_k f(\tilde{\lambda}_k; t, \sigma)$
8: end for
9: Output: $\phi(t) = \frac{1}{n_v} \sum_{l=1}^{n_v} \left( \sum_{i=1}^{q} \tau_i^{(l)} f(\tilde{\lambda}_i^{(l)}; t, \sigma) \right)$
```

Recent work in NLA/RandNLA has provided efficient matrix-free algorithms to estimate this ESD [142, 86, 230] through Stochastic Lanczos Quadrature (SLQ). Here, we briefly describe SLQ in simple terms. This approach was also used in [84] to compute the Hessian ESD. For more details, see [142, 86, 230].

Here is a summary of our approach to compute the ESD $\phi(t)$. First, we approximate $\phi(t)$ (of (6.4)) by $\phi_\sigma(t)$ ((6.5) below) by applying a Gaussian kernel (*first approximation*),

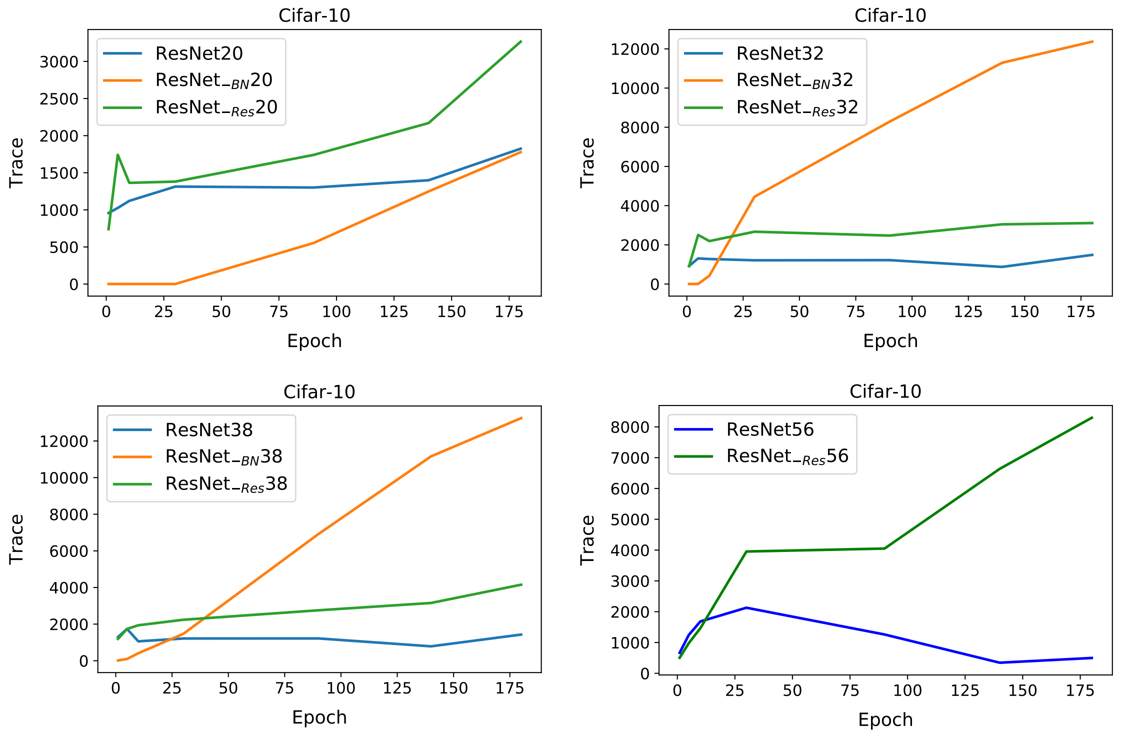

Figure 6.2: The Hessian trace of the entire network for ResNet/ResNet$_{-BN}$/ResNet$_{-Res}$ with depth 20/32/38/56 on Cifar-10. For each depth, ResNet (blue) is the baseline. It can be clearly seen that removing BN from the architecture (orange) generally results in a rapid increase of the Hessian trace. This increase is more pronounced for deeper networks such as ResNet32 and ResNet38. Importantly, the Hessian trace of ResNet20 without BN is lower than the original model (blue). This is in contrast to the claim of [206]. Also, we generally observe that residual connections smooth the Hessian trace for both shallow and deep networks (compare blue and green lines). Results on Cifar-100 also exhibit the same behaviour (as shown in Figure E.5).

and we express this in the same expectation form as in the Hutchinson algorithm ((6.9) below). Next, since the computation inside the expectation depends directly on t and the unknown eigenvalues (denoted by λ_is), we further simplify the problem by using Gaussian quadrature ((6.13) below) (*second approximation*). Then, since the weights and λ_is in the Gaussian quadrature are still unknown, we use the stochastic Lanczos algorithm to approximate the weights and λ_is ((6.14) below) (*third approximation*). Finally, we approximate the expectation of the eigenvalue distribution as a sum ((6.15) below) (*forth approximation*).

In more detail, for the first approximation, we apply a Gaussian kernel, f, with variance

σ^2 to (6.4) to obtain

$$\phi_\sigma(t) = \frac{1}{m}\sum_{i=1}^{m} f(\lambda; t, \sigma), \tag{6.5}$$

where $f(\lambda; t, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} exp(-(t-\lambda)^2/(2\sigma^2))$ is the Gaussian kernel. Clearly, $\phi_\sigma(t) \to \phi(t)$, as $\sigma \to 0$. Thus, if we had an algorithm to approximate (6.5), then we could take the limit and reduce the standard deviation of the Gaussian kernel to approximate (6.4). In our context, the question of how to compute $\phi_\sigma(t)$ amounts to computing the density distribution of the Hessian convolved with a Gaussian kernel.

To do this, observe that

$$Tr(f(H)) = Tr(Qf(\Lambda)Q^T) = Tr(f(\Lambda)), \tag{6.6}$$

where $Q\Lambda Q^T$ is the eigendecomposition of H, and let $f(H)$ be the matrix function defined as

$$f(H) \triangleq Qf(\Lambda)Q^T \triangleq Q\text{diag}(f(\lambda_1), ..., f(\lambda_m))Q^T. \tag{6.7}$$

We can plug (6.6) into (6.5) to get

$$\phi_\sigma(t) = \frac{1}{m} Tr(f(H; t, \sigma)). \tag{6.8}$$

For a given value of t, the trace $Tr(f(H; t, \sigma))$ can be efficiently computed using the Hutchinson algorithm (described in §6.3.2). That is, we draw a random Rademacher vector v and compute the expectation $\mathbf{E}[v^T f(H; t, \sigma)v]$ to get

$$\phi_\sigma(t) = \frac{1}{m}\mathbf{E}[v^T f(H; t, \sigma)v]. \tag{6.9}$$

However, this is still intractable, as the trace computation needs to be repeated for every value of t (which scales with the number of model parameters).

To get around this, we relax this problem further [142, 230]. Define $\phi^v_\sigma(t) = v^T f(H; t, \sigma)v$, in which case we have

$$\begin{aligned} \phi^v_\sigma(t) &= v^T f(H; t)v = v^T Q f(\Lambda; t) Q^T v \\ &= \sum_{i=1}^{m} \mu_i^2 f(\lambda_i; t), \end{aligned} \tag{6.10}$$

where μ_i is the magnitude (or dot product) of v along the i^{th} eigenvector of H. Now let us define a probability distribution w.r.t. α with the cumulative distribution function, $\pi(\alpha)$, as the following piece-wise function:

$$\pi(\alpha) = \begin{cases} 0 & \alpha \le \lambda_m, \\ \sum_{i=1}^{j} \mu_i^2 & \lambda_j \le \alpha \le \lambda_{j-1}, \\ \sum_{i=1}^{m} \mu_i^2 & \lambda_1 \le \alpha. \end{cases} \tag{6.11}$$

Then, by the Riemann-Stieltjes integral, it follows that

$$\phi_\sigma^v(t) = \int_{\lambda_m}^{\lambda_1} f(\alpha; t) d\pi(\alpha). \tag{6.12}$$

This integral can be estimated by the Gauss quadrature rule [87],

$$\phi_\sigma^v(t) \approx \sum_{i=1}^{q} \omega_i f(t_i; t, \sigma), \tag{6.13}$$

where (ω_i, t_i) is the weight-node pair to estimate the integral. The stochastic Lanczos algorithm can then be used to estimate accurately this quantity [230, 86, 142]. Specifically, for the q-step Lanczos algorithm, we have q eigenpairs $(\tilde{\lambda}_i, \tilde{v}_i)$. Let $\tau_i = (\tilde{v}_i[1])^2$, where $\tilde{v}_i[1]$ is the first component of $\tilde{v}_i$, in which case it follows that

$$\phi_\sigma^v(t) \approx \sum_{i=1}^{q} \omega_i f(t_i; t, \sigma) \approx \sum_{i=1}^{q} \tau_i f(\tilde{\lambda}_i; t, \sigma). \tag{6.14}$$

Therefore, as in the Hutchinson algorithm, with multiple different runs (e.g., n_v times) of Lanczos algorithm, ϕ_σ can be approximated by

$$\phi_\sigma(t) = Tr(f(H)) \approx \frac{1}{n_v} \sum_{l=1}^{n_v} \left(\sum_{i=1}^{q} \tau_i^{(l)} f(\tilde{\lambda}_i^{(l)}; t, \sigma) \right). \tag{6.15}$$

See Algorithm 6 for a description of the SLQ algorithm.

6.4 Results

Here, we provide extensive experiments to study the impact of BN and residual connection on the Hessian spectrum. We first start by discussing the experimental settings in §6.4.1, followed by presenting the Hessian spectrum results for the entire model in §6.4.2 as well different ResNet stages in §6.4.3.

6.4.1 Experimental Setting

Using PyHessian, we measure all three Hessian spectrum metrics (i.e., top eigenvalues, trace, and full ESD) throughout the training process of SGD with momentum. We consider various ResNet [104] architectures, and in particular ResNet20/32/38/56 on the Cifar-10, and we analyze these models and variants with/without BN and with/without residual connections. We also experimented with same networks tested on Cifar-100 dataset, and all of the observations were consistent. These results are presented in Appendix.

For clarity, we refer to the networks without BN as ResNet$_{-BN}$, and we refer to the networks without residual connections as ResNet$_{-Res}$. We train each model with various initial learning rates, and we pick the best performing result for analysis. See Appendix E.3 for more details on training settings. We analyze the spectrum throughout training at all checkpoints. The accuracy of each model is reported in Table 6.1, and the testing curve is shown in Figure E.2.

Table 6.1: Accuracy of ResNet, ResNet$_{-BN}$, and ResNet$_{-Res}$, with different depths, on Cifar-10. The accuracy drops if the BN layer is removed (denoted by ResNet$_{-BN}$), and this degradation is more pronounced for deeper models. In fact, ResNet$_{-BN}$ 56 cannot be trained at all. Removing the residual connections (denoted as ResNet$_{-Res}$) also results in slight performance degradation. Accuracy of models on Cifar-100 is reported in Table E.1.

Model\Depth	20	32	38	56
ResNet	92.01%	92.05%	92.37%	93.59%
ResNet$_{-BN}$	87.27%	66.57%	53.65%	N/A
ResNet$_{-Res}$	90.66%	89.8%	88.92%	87.38%

6.4.2 Full Network Hessian Analysis

We start with the original ResNet model with BN and residual connections. Hereafter we refer to this as ResNet. The behaviour of the Hessian trace throughout training is shown in Figure 6.2. Furthermore, we show the evolution of the Hessian ESD throughout training in Figure 6.3 for Cifar-10.

6.4.2.0.1 Batch Normalization As discussed before, a BN layer is crucial for training NN models, and removing this component can adversely affect the generalization performance, as is shown in Table 6.1. The drop in performance is very significant for deeper models. For example, we could not even train ResNet56 on Cifar-10 without a BN layer, even with hyperparameter tuning.

The first interesting observation is that removing BN layer (denoted by ResNet$_{-BN}$) exhibits different behaviour for shallower versus deeper models. For example, for ResNet20 we see that removing BN results in smaller trace and Hessian ESD values, as compared to baseline, as shown in Figure 6.2 (orange curve versus blue curve), and 6.3 (second versus first column). In more detail, from the evolution plot of Figure 6.3 throughout training, it can be seen that the ESD of ResNet$_{-BN}$ 20 initially reduces significantly and centers around zero. That is, the model gets attracted to areas with a significantly large number of small/degenerate Hessian directions. This continues until epoch 30, at which point the training gets attracted to regions of the loss landscape with several non-degenerate Hessian directions.

This clearly shows that training without BN makes training harder, but it does not necessarily mean that the Hessian spectrum is going to be larger than the baseline model, despite the claim made by [206]. In fact, we only observe the smoothing behaviour proposed

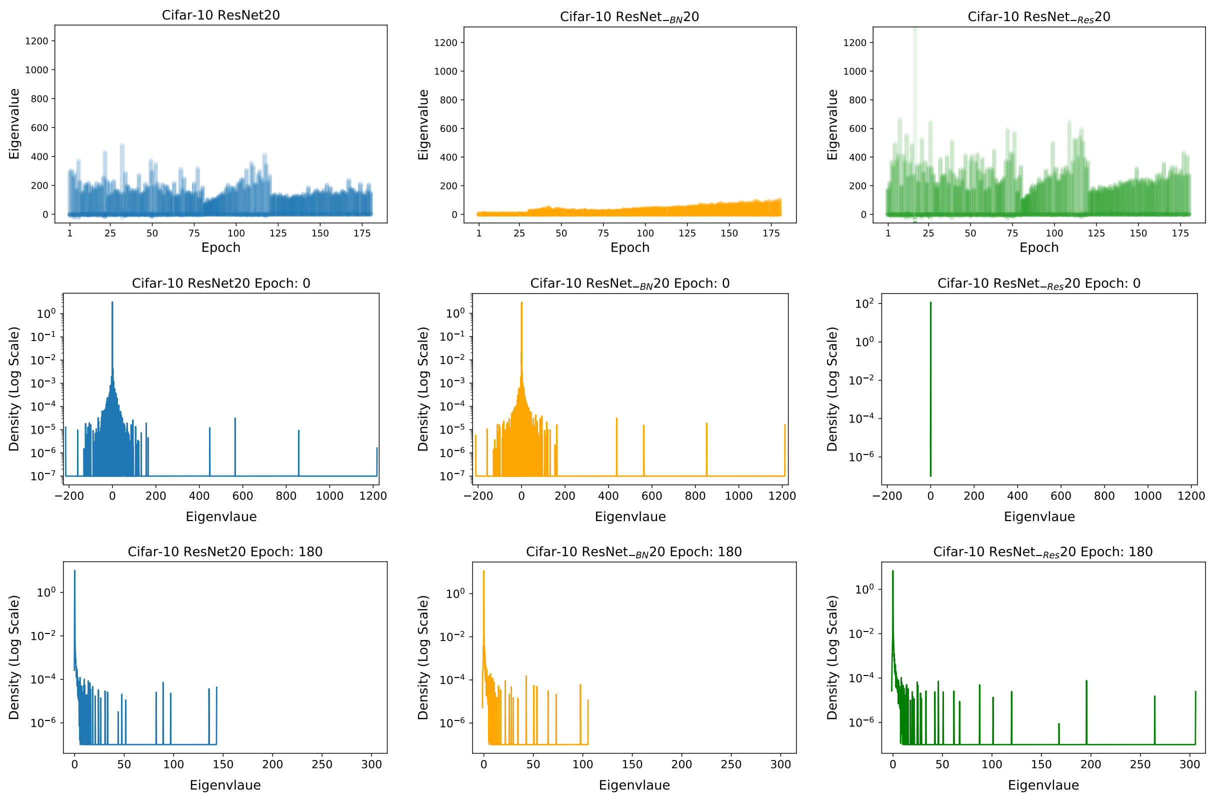

Figure 6.3: (first row) We show the Hessian ESD throughout training for ResNet/ResNet$_{-BN}$/ResNet$_{-Res}$ (each shown in a different column) with depth 20 on Cifar-10. For a fixed epoch, every point corresponds to a Hessian eigenvalue. These plots show several important phenomena. First note that removing BN (middle column) does not lead to a non-smooth loss landscape as was claimed by [206]. We can clearly see that this is true throughout training. However, removing the residual connection leads makes the loss landscape non-smooth throughout training (middle/last row). We show the Hessian ESD at epoch 0 and epoch 180. This clearly shows that removing BN leads to a maximum eigenvalue of 100, whereas the baseline has a maximum Hessian eigenvalue of 150. See Figure E.7, where we plot the Hessian ESD for several other epochs throughout training. We observed the same behaviour on Cifar-100 dataset (as shown in Figure E.11).

by [206] for deeper NN models. For example, observe the Hessian trace plot of ResNet32/38, shown in Figure 6.2. Here, the Hessian trace of ResNet$_{-BN}$ 32 increases to 10000 from zero, as compared to 2000 for ResNet. The Hessian ESD also exhibits the same behaviour, as shown in Figure E.8 and E.9. We can clearly see that the range of eigenvalues of ResNet$_{-BN}$ is significantly larger, as compared to ResNet.

The Hessian ESD of ResNet32 and ResNet38 *throughout the training process* is shown in Figure E.8, E.9. Again, we see the interesting behaviour that without the BN layer, the spectrum initially converges to degenerate Hessian directions, before finding non-degenerate directions in later epochs of training. The Hessian trace and the range of the Hessian ESD significantly increases as the model gets deeper.

These plots show the numerical values of the Hessian spectrum. However, the results could be more intuitively presented via parametric plots of the loss landscape. We plot the parametric 3D loss landscapes of ResNet20/38 on Cifar-10 with/without BN in Figure 6.1 (compare left and middle columns). These plots are computed by perturbing the model parameters across the first and second eigenvectors of the Hessian. For ResNet20, it can be clearly seen that removing the BN layer (middle plot) results in convergence to a flatter local minimum, as compared to ResNet20 with BN. This observation is the opposite of the common belief that adding BN layer makes the loss landscape smoother [206]. However, for ResNet38, we can also see that removing the BN layer results in convergence to a point with a higher value of loss. The visualizations corroborate our finding that initially ResNet$_{-BN}$ finds points with degenerate Hessian directions, before converging to a point with non-degenerate directions. We provide more visualizations for ResNet20 (Figure E.14), ResNet32 (Figure E.15), and ResNet38 (Figure E.16), which show the same behaviour.

In summary, our empirical results highlight two points. First, our findings show several fine-scale behaviours when the BN layer is removed. Importantly, we find that the observation made in [206] only holds for deeper models, and are not necessarily true for shallow networks. Second, using the scalable Hessian-based techniques implemented in PyHessian, one can test the hypotheses that these or other claims hold more generally. For example, we observed exactly similar behaviour for Cifar-100 dataset, as shown in Appendix E.5.

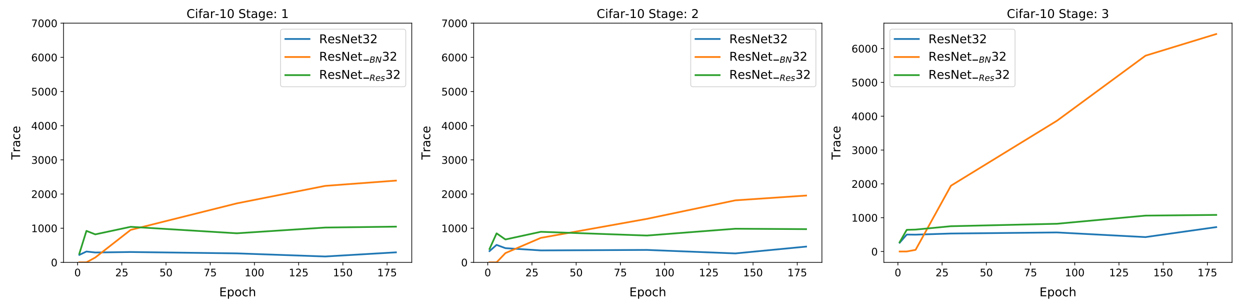

Figure 6.4: Stage-wise Hessian trace of ResNet/ResNet$_{-BN}$/ResNet$_{-Res}$ 32 on Cifar-10. (See Figure E.4 for depth 20/32; and see Figure E.1 for stage illustration.) Removing BN layer from the third stage significantly increases the trace, compared to removing BN layer from the first/second stage. This has a direct correlation with the final generalization performance, as shown in Table 6.2. ResNet/ResNet$_{-BN}$/ResNet$_{-Res}$ on Cifar-100 has the similar trend as shown in Figure E.6.

Table 6.2: Accuracy of ResNet models on Cifar-10 with different depths is shown in the first row. Accuracy of the corresponding architectures, but with BN removed from one of the stages, is shown in the next three rows, respectively. (See Figure E.1 for stage definition.) For instance, the last row is a ResNet model with no BN layer in the third stage. We observe a general correlation between the accuracy drop and stage based Hessian analysis, shown in Figure 6.4. In particular, we see that stages which significantly affect accuracy also exhibit a significant increase in the Hessian trace. Models on Cifar-100 have the similar trend, as shown in Table E.2.

Model\Depth	20	32	38	56
ResNet	92.01%	92.05%	92.37%	93.59%
RM BN stage 1	91.28%	91.98%	92.20%	92.19%
RM BN stage 2	91.49%	91.94%	91.70%	92.20%
RM BN stage 3	90.59%	88.57%	86.96%	73.77%

6.4.2.0.2 Residual Connection We next study the impact of residual connections on the smoothness of the loss landscape. Removing residual connections leads to slightly poorer generalization, as shown in Table 6.1, although the degradation is much smaller than removing the BN layer.

We report the behaviour of the Hessian trace for ResNet$_{-Res}$ in Figure 6.2 for ResNet20/32/38/56 on Cifar-10. It can clearly be seen that the trace of ResNet$_{-Res}$ is consistently higher than that of ResNet, for both shallow and deep models on different datasets.

In addition, from the Hessian ESD in Figure 6.3, E.7, E.8, E.9, and E.10, we can see that the top eigenvalues as well as the support range of ESD of ResNet$_{-Res}$ increases for deeper models. These results are in line with the findings of [141].

We also visualize the loss landscape of these models in Figure 6.1, E.14, E.15, E.16, and E.17. It can clearly be seen that the converging point for ResNet$_{-Res}$ becomes sharper, as compared with ResNet, as the depth grows.

Again, our empirical results highlight two points. First, we make observations that provide a finer-scale understanding of seemingly-contradictory claims in the previous literature. Second, using the scalable Hessian-based techniques that are implemented in PyHessian, one can ask these questions and test the hypotheses that these or other claims hold more generally. Similar to the previous section, we saw exactly similar behaviour for Cifar-100, as reported in Appendix E.5.

6.4.3 Stage-wise Hessian Analysis

We also analyzed the impact of removing BN and residual connection from different stages of the model. We define each stage of ResNet as blocks with the same activation resolution, as schematically shown in Figure E.1.

We plot the Hessian trace for the three stages of ResNet32 on Cifar-10 in Figure 6.4 (similar plots for ResNet20/38/56 on Cifar-10 is shown in Figure E.4). We can clearly see that removing the BN from the last stage of ResNet32 results in a more rapid increase in the Hessian trace, as compared to removing BN from the first or second stage. Interestingly, this has a direct correlation with the final generalization performance reported in Table 6.2. We can see that removing BN in the third stage results in higher accuracy drop, as compared to removing it from other stages. A similar trend exists for other models (ResNet20/38); and we generally observe the same behaviour on Cifar-100, as reported in Figure E.6 and Table E.2.

As for the residual connections, we can see that removing them results in a relatively smaller increase in the Hessian trace, and correspondingly the impact of removing the residual connections on accuracy is smaller, as compared to removing BN. See Table 6.3 for Cifar-10.

Table 6.3: Accuracy of ResNet on Cifar-10 is reported for baseline (first row), along with architectures where the residual connection is removed at different stages.

Model\Depth	20	32	38	56
ResNet	92.01%	92.05%	92.37%	93.59%
RM Res stage 1	91.52%	92.27%	91.74%	91.79%
RM Res stage 2	91.06%	91.07%	91.08%	91.28%
RM Res stage 3	91.54%	92.09%	92.14%	92.34%

6.4.4 Summary of Results

Table 6.4 presents a summary of the tables and figures used in this work and their corresponding properties, i.e., Accuracy, Trace, ESD, and Loss Landscape.

Table 6.4: Navigation summary for all figures and tables used throughout this paper.

	Cifar-10	Cifar-100
Accuracy	Table 6.1, Figure E.2	Table E.1, Figure E.3
RM BN Acc.	Table 6.2	Table E.2
RM Res Acc	Table 6.3	Table E.3
Trace	Figure 6.2	Figure E.5
Stage-wise Trace	Figure 6.4, E.4	Figure E.6
ESD	Figure 6.3, E.7, E.8, E.9, E.10	Figure E.11, E.12, E.13
Loss Landscape	Figure 6.1, E.14, E.15, E.16, E.17	Figure E.18, E.19, E.20

6.5 Conclusions

We have developed PyHessian, an open-source framework for analyzing NN behaviour through the lens of the Hessian [111]. PyHessian enables direct and efficient computation of Hessian-based statistics, including the top eigenvalues, the trace, and the full ESD, with support for distributed-memory execution on cloud/supercomputer systems. Importantly, since it uses matrix-free techniques, PyHessian accomplishes this without the need to form the full Hessian. This means that we can compute second-order statistics for state-of-the-art NNs in times that are only marginally longer than the time used by popular stochastic gradient-based techniques.

As a typical application, we have also shown how PyHessian can be used to study in detail the impact of popular NN architectural changes (such as adding/modifying BN and residual connections) on the NN loss landscape. Importantly, we found that adding BN layers does not necessarily result in a smoother loss landscape, as claimed by [206]. We have observed this phenomenon only for deeper models, where removing the BN layer results in convergence to "sharp" local minima that have high training loss and poor generalization, but it does not seem to hold for shallower models. We also showed that removing residual connections resulted in a slightly coarser loss landscape, a finding which we illustrated with parametric 3D visualizations, and which all three Hessian spectrum metrics confirmed. We have open-sourced PyHessian to encourage reproducibility and as a scalable framework for research on second-order optimization methods, on practical diagnostics for NN learning/generalization, as well as on analytics tools for NNs more generally.

Part III

Novel Application

Chapter 7

HAWQ-V3: Hessian Aware trace-Weighted integer-only Quantization of Neural Networks

7.1 Introduction

An important step toward realizing pervasive deep learning is enabling real-time inference, both at the edge and in the cloud, with low energy consumption and state-of-the-art model accuracy. This will have a significant impact on applications such as real-time intelligent healthcare monitoring, autonomous driving, audio analytics, and speech recognition. Over the past decade, we have observed significant improvements in the accuracy of Neural Networks (NNs) for various tasks. However, the state-of-the-art models are often prohibitively large and too compute-heavy to be deployed for real-time use. A promising approach to address this is through quantization [94, 101], where low-precision quantized integer values are used to express the model parameters and feature maps. That can help reduce the model footprint, and improve inference speed and energy consumption.

However, existing quantization algorithms often use *simulated quantization*, where the parameters are stored with quantization, but are cast to floating point for inference. As a result, all or part of the inference operations (e.g. convolution, matrix operations, batch norm layers, residual connections) are performed using floating point precision. This of course limits the speed up as we cannot utilize low precision logic. To address this, we build upon existing integer-only quantization methods [120], and propose systematic methods to extend them to low and mixed-precision quantization with Hessian as the sensitive metric [69]. In particular, we make the following contributions:

- We develop HAWQ-V3, a mixed-precision integer-only quantization framework with integer-only multiplication, addition, and bit shifting with static quantization. Importantly, no floating point and no integer division calculation is performed in the entire inference. This includes the batch norm layers and residual connections, which are typically kept at floating point precision in prior integer-only quantization work [70]. While keeping these operations in floating point helps accuracy, this is not allowed for integer-only hardware. We show that ignoring this and attempting to deploy a model that uses floating point residual on integer-only hardware can lead to more than 90% mismatch (Figure F.3). HAWQ-V3 completely avoids this by using a novel approach to perform residual connections in pure integer-only arithmetic. See Section 7.3.3 and Appendix F.7 for details.

- We propose a novel hardware-aware mixed-precision quantization formulation that uses an Integer Linear Programming (ILP) problem to find the best bit-precision setting. The ILP solver minimizes the model perturbation using the Hessian metric while observing application-specific constraints on model size, latency, and total bit operations. Compared to the contemporary work of [115], our approach is hardware-aware and uses direct hardware measurement to find a bit precision setting that has the optimal balance between latency and accuracy. See Section 7.3.4 and Appendix F.9 for details.
- To verify the feasibility of our approach, we deploy the quantized integer-only models using Apache TVM [51] for INT8, INT4, and mixed-precision settings. To the best of our knowledge, our framework is the first that adds INT4 support to TVM. By profiling the latency of different layers, we show that we can achieve an average of 1.47× speed up with INT4, as compared to INT8 on a T4 GPU for ResNet50. See Section 7.3.5 and Table 7.2 for more details.
- We extensively test HAWQ-V3 on a wide range of workloads, including ResNet18, ResNet50, and InceptionV3, and show that we can achieve a substantial performance improvement, as compared to the prior state-of-the-art. For instance, we achieve an accuracy of 78.50% with INT8 quantization, which is more than 4% higher than prior integer-only work for InceptionV3. Furthermore, we show that mixed-precision INT4/8 quantization can be used to achieve higher speed up as compared to INT8 inference with minimal impact on accuracy. For example, for ResNet50 we can speedup latency by 23% as compared to INT8 and still achieve 76.73% accuracy. See Section 7.4 and Table 7.1, 7.2 for more details.

7.2 Related Work

There have been significant efforts recently to improve the trade-off between accuracy and efficiency of NN models. These can be broadly categorized as follows: (i) Designing new NN architectures [117, 204, 226]; (ii) Co-designing NN architecture and hardware together [100, 83, 241, 109]; (iii) Pruning redundant filters [135, 102, 161, 140, 152, 248]; (iv) knowledge distillation [107, 160, 185, 256]; and (v) using quantization (reduced precision). Here, we provide a more detailed overview of this related work.

Quantization. A common solution is to compress NN models with quantization [7, 114, 190, 267, 268, 120, 262, 70, 54], where low-bit precision is used for weights/activations. Quantization reduces model size without changing the original network architecture, and it could potentially permit the use of low-precision matrix multiplication or convolution.

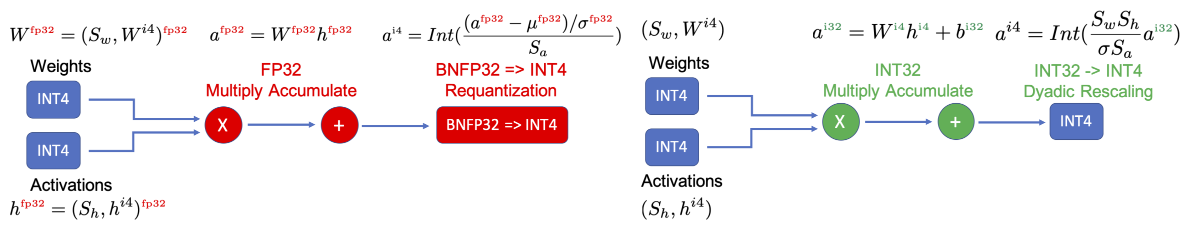
$W^{\text{fp32}} = (S_w, W^{i4})^{\text{fp32}}$
Weights
INT4
INT4
Activations
$h^{\text{fp32}} = (S_h, h^{i4})^{\text{fp32}}$
$a^{\text{fp32}} = W^{\text{fp32}} h^{\text{fp32}}$
FP32 Multiply Accumulate
X
+
$a^{\text{i4}} = Int(\frac{(a^{\text{fp32}} - \mu^{\text{fp32}})/\sigma^{\text{fp32}}}{S_a})$
BNFP32 => INT4 Requantization
BNFP32 => INT4
(S_w, W^{i4})
Weights
INT4
INT4
Activations
(S_h, h^{i4})
$a^{\text{i32}} = W^{\text{i4}} h^{\text{i4}} + b^{\text{i32}}$
INT32 Multiply Accumulate
X
+
$a^{i4} = Int(\frac{S_w S_h}{\sigma S_a} a^{\text{i32}})$
INT32 -> INT4 Dyadic Rescaling
INT4

Figure 7.1: Illustration of fake vs true quantization for convolution and batch normalization folding. For simplicity, we ignore the affine coefficient of BN. (Left) In the simulated quantization (aka fake quantization approach), weights and activations are simulated as integers with floating point representation, and all the multiplication and accumulation happen in FP32 precision. Furthermore, the BN parameters (i.e. μ and σ) are stored and computed using FP32 precision. This is undesirable but can significantly help accuracy since BN parameters are sensitive to quantization. However, with this approach, one cannot benefit from low-precision ALUs. (Right) An illustration of the integer-only pipeline with dyadic arithmetic for convolution and BN folding. The standard deviation (σ) in BN is merged into the quantization scale of the weights, and the mean is quantized to INT32 and merged as a bias into the weights (denoted by b^{i32}) Note that with this approach, all the weights and activations are stored in integer format, and all the multiplications are performed with INT4 and accumulated in INT32 precision. Finally, the accumulated result is requantized to INT4 with dyadic scaling (denoted by $\frac{S_w S_h}{\sigma S_a}$). Importantly, no floating point or even integer division is performed. See Section 7.3.2 and Appendix F.4 for more details.

While the gains on speed/power increase for low-precision quantization, low-precision quantization suffers from accuracy degradation. To address this, recent work uses non-uniform quantizers [262], channel-wise quantization [128], and progressive quantization-aware fine-tuning [267]. Other works try to include periodic regularization to assist quantization [167, 76], apply post training quantization [14, 33, 115], or improve accuracy by changing the channel counts accordingly for different layers [54]. Despite these advances, performing uniform ultra low-bit quantization still results in a significant accuracy degradation. A promising direction is to use mixed-precision quantization [269, 235, 70, 215], where some layers are kept at higher precision, while others are kept at a lower precision. However, a challenge with this approach is finding the right the mixed-precision setting for the different layers. A brute force approach is not feasible since the search space is exponentially large in the number of layers.

HAQ [235] proposes to search this space by applying a Reinforcement Learning algorithm, while [242] uses a Differentiable Neural Architecture Search. However, these searching methods require large computational resources, and their performance is very sensitive to hyper-parameters and even initialization. To address these issues, HAWQ [70, 69] introduces an automatic way to find good mixed-precision settings based on the sensitivity obtained using the Hessian spectrum. However, the Pareto frontier method in [69] is not flexible enough to satisfy simultaneously different requirements on hardware. To address this, we propose here an ILP solution that can generate mixed-precision settings with various constraints (such as model size, BOPS, and latency), and which can be solved within seconds on commodity hardware. The contemporary work of [115] also proposes to use an ILP. However, their approach is not hardware aware, and their approach uses FP32 casting.

Another issue is that the quantized weights and activations need to be converted to floating point precision during inference, as shown in Figure 7.1. This high-precision casting can have high overhead and limits inference speed, especially for hardware with limited on-chip memory. Using FP32 ALUs also requires a larger die area in the chip, further limiting the peak computational capacity of the hardware. The work of [120] addresses this casting problem by using integer-only quantization in INT8 precision. However, there are several shortcomings associated with their approach (which are addressed in HAWQ-V3). First, [120] does not support low-precision or mixed-precision quantization. We show that this is useful in practice, as it can improve the inference speed by up to 50% with a small impact on accuracy. Second, both [120] and HAWQ are hardware agnostic and do not co-design/adapt the quantization for the target hardware. In contrast, the ILP approach in HAWQ-V3 is hardware aware, and it directly takes this into account when determining mixed-precision bit setting. Third, as we discuss in Section 7.3.2, the approach used in [120] leads to sub-optimal accuracy for INT8 quantization, while our approach can achieve up to 5% higher accuracy for INT8 inference. Finally, to address the absence of low-precision support in previous works [120, 70], we extend TVM to support INT4 and mixed-precision quantization, and we validate our results by directly running the quantized model with low bit-width on the hardware. See Appendix F.1 for the discussion of different deployment frameworks.

7.3 Methodology

Assume that the NN has L layers with learnable parameters, denoted as $\{W_1, W_2, ..., W_L\}$, with θ denoting the combination of all such parameters. For a supervised setting, the goal is to optimize the following empirical risk minimization loss function:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum\nolimits_{i=1}^{N} l(x_i, y_i; \theta), \tag{7.1}$$

where (x, y) is the input data and the corresponding label, $l(x, y; \theta)$ is the loss function (e.g., MSE or Cross Entropy loss), and N is the total number of data points. We assume that we have the trained model parameters θ given in floating point precision. Our goal is to quantize the model with the optimal trade-offs among memory footprint, speed, and accuracy. Below, we first define quantization and then present HAWQ-V3.

Uniform Quantization. Quantization restricts NN weights/activations to a finite set of values as follows:

$$Q(r) = \text{Int}\big(r/S\big) - Z, \tag{7.2}$$

where Q is the quantization operator, r is a real valued number (activation or a weight), S is a real valued scaling factor, and Z is the zero point, chosen such that the 0 value would exactly map to quantized values. Furthermore, Int maps a floating point value to an integer value through a rounding operation (e.g., round to nearest and truncation).

This formulation for Q corresponds to uniform quantization. However, some work in the literature has also explored non-uniform quantization [55, 262, 177, 235]. Although non-uniform quantization may achieve higher accuracy for a fixed bit-width, such approaches are typically difficult to deploy on hardware to reduce latency.[1] As such, for HAWQ-V3, we only focus on uniform quantization. Meanwhile, for HAWQ-V3, we use (i) symmetric quantization for weights and asymmetric quantization for activations; and (ii) static quantization for all the scaling factors S. Please see Appendix F.2 for more details.

7.3.1 Quantized Matrix Multiplication and Convolution

Consider a layer with hidden activation denoted as h and weight tensor denoted as W, followed by ReLU activation. First, h and W are quantized to $S_h q_h$ and $S_w q_w$, where S_h and S_w are the real valued quantization scales, q_h and q_W are the corresponding quantized integer values. The output result, denoted with a, can be computed as follows:

$$a = S_w S_h (q_w * q_h), \tag{7.3}$$

where $q_w * q_h$ is the matrix multiplication (or convolution) calculated with integer in low precision (e.g., INT4) and accumulated in INT32 precision. This result is then requantized and sent to the next layer as follows:

$$q_a = \text{Int}\left(\frac{a}{S_a}\right) = \text{Int}\left(\frac{S_w S_h}{S_a}(q_w * q_h)\right), \tag{7.4}$$

where S_a is the pre-calculated scale factor for the output activation.

In HAWQ-V3, the $q_w * q_h$ operation is performed with low-precision integer-only multiplication and INT32 accumulation, and the final INT32 result is quantized by scaling it with $S_w S_h / S_a$. The latter is a floating point scaling that needs to be multiplied with the accumulated result (in INT32 precision). A naive implementation requires floating point multiplication for this stage. However, this can be avoided by enforcing the scaling to be a dyadic number. Dyadic numbers are rational numbers with the format of $b/2^c$, where b, c are two integer numbers. As such, a dyadic scaling in (7.4) can be efficiently performed using INT32 integer multiplication and bit shifting. Given a specific $S_w S_h / S_a$, we use DN (representing Dyadic Number) to denote the function that can calculate the corresponding b and c:

$$b/2^c = \text{DN}\left(S_w S_h / S_a\right). \tag{7.5}$$

An advantage of using dyadic numbers besides avoiding floating point arithmetic, is that it removes the need to support division (which typically has an order of magnitude higher latency than multiplication) in the hardware. This approach is used for INT8 quantization in [120], and we enforce all the rescaling to be dyadic for low-precision and mixed-precision quantization as well.

[1]However, they can reduce total model footprint.

7.3.2 Batch Normalization

Batch normalization (BN) is an important component of most NN architectures, especially for computer vision applications. BN performs the following operation to an input activation a:

$$\text{BN}(a) = \beta \frac{a - \mu_B}{\sigma_B} + \gamma \tag{7.6}$$

where μ_B and σ_B are the mean and standard deviation of a, and β, γ are trainable parameters. During inference, these parameters (both statistics and trainable parameters) are fixed, and therefore the BN operations could be fused with the convolution (see Appendix F.4). However, an important problem is that quantizing the BN parameters often leads to significant accuracy degradation. As such, many prior quantization methods keep BN parameters in FP32 precision (e.g., [69, 33, 54, 55, 262, 177], just to name a few). This makes such approaches not suitable for integer-only hardware. While using such techniques help accuracy, HAWQ-V3 completely avoids that. We fuse the BN parameters with the convolution and quantized them with integer-only approach (Please see Figure 7.1 where we compare simulated qauntization and HAWQ-V3 for BN and convolution.).

Another important point to discuss here is that we found the BN folding used in [120] to be sub-optimal. In their approach BN and CONV layers are fused together while BN running statistics are still kept updating. This actually requires computing each convolution layer twice, once without BN and then with BN (as illustrated in [120, Figure C8]). However, we found that this is unnecessary and degrades the accuracy. Instead, in HAWQ-V3, we follow a simpler approach where we first keep the Conv and BN layer unfolded, and allow the BN statistics to update. After several epochs, we then freeze the running statistics in the BN layer and fold the CONV and BN layers (please see Appendix F.4 for details). As we will show in Section 7.4, this approach results in better accuracy as compared to [120].

7.3.3 Residual Connection

Residual connection [104] is another important component in many NN architectures. Similar to BN, quantizing the residual connections can lead to accuracy degradation, and as such, some prior quantization works perform the operation in FP32 precision [55, 262, 235]. There is a common misunderstanding that this may not be a big problem. However, this actually leads to complete loss of signal, especially for low precision quantization. The main reason for this is that quantization is not a linear operation, that is $Q(a+b) \neq Q(a)+Q(b)$ (a, b are floating point numbers). As such, performing the accumulation in FP32 and then quantizing is not the same as accumulating quantized values. Therefore, it is not possible to deploy quantization methods that keep residual connection in FP32 in integer-only hardware (we provide more detailed discussion of this in Appendix F.6 and also quantify the resulting error which can be more than 90%).

We avoid this in HAWQ-V3, and use INT32 for the residual branch. We perform the following steps to ensure that the addition operation can happen with dyadic arithmetic. Let

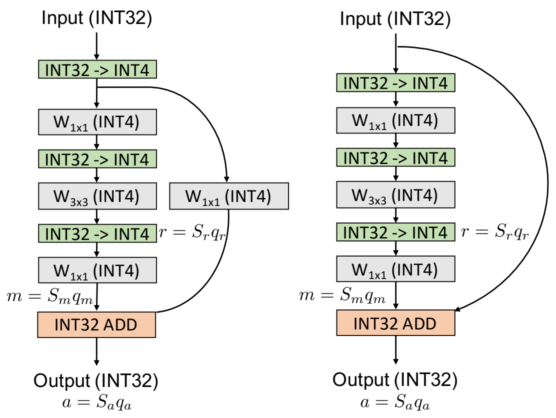

Figure 7.2: Illustration of HAWQ-V3 for a residual block with and without transition layer. Input feature map is given in INT32 precision, which is requantized to INT4 precision (green boxes) before any convolution layer (gray boxes). The BN layer is folded into the convolution. The residual addition is performed in INT32 precision, and the final accumulated result is re-scaled and sent to the next layer. For blocks with a transition layer, we only quantize the input once to INT4 and we use the same result for both 1×1 convolutions.

us denote the activation passing through the residual connection as $r = S_r q_r$.[2] Furthermore, let us denote the activation of the main branch before residual addition as $m = S_m q_m$, and the final output after residual accumulation by $a = S_a q_a$. Then we will have:

$$q_a = \text{DN}\left(S_m/S_a\right) q_m + \text{DN}\left(S_r/S_a\right) q_r. \tag{7.7}$$

Note that with this approach, we only need to perform a dyadic scaling of q_m and add the result with the dyadically scaled q_r. All of these operations can happen with integer-only arithmetic. Also we should note that in our approach all the scales are statically known. These steps are schematically illustrated in Figure 7.2 for a residual connection with/without downsampling. Similar approach is performed for concatenation layer as well (see Appendix F.5).

7.3.4 Mixed Precision and Integer Linear Programming

Uniformly quantizing all the layers to low bit-width (e.g. INT4) could lead to significant accuracy degradation. However, it is possible to benefit from low-precision quantization by keeping a subset of sensitive layers at high precision [70]. The basic idea is to keep sensitive layers at higher precision and insensitive layers at lower precision. An important component of HAWQ-V3 is that we directly consider hardware-specific metrics such as latency, to select the bit-precision configuration. This is important since a layer's latency

[2]This is either the input or the output activation after the downsampling layer.

does not necessarily halve when quantized from INT8 to INT4 precision. In fact, as we discuss in Section 7.4, there are specific layer configurations that do not gain any speed up when quantized to low precision, and some that superlinearly benefit from quantization. As such, quantizing the former will not lead to any latency improvement, and will only hurt accuracy. Therefore, it is better to keep such layers at high precision, even if they have low sensitivity. These trade-offs between accuracy and latency should be taken into consideration when quantizing them to low precision. Importantly, these trade-offs are hardware-specific as latency in general does not correlate with the model size and/or FLOPS. However, we can consider this by directly measuring the latency of executing a layer in quantized precision on the target hardware platform. This trade-off is schematically shown in Figure 7.3 (and later quantified in Figure F.4). We can use an Integer Linear Programming (ILP) problem to formalize the problem definition of finding the bit-precision setting that has optimal trade-off as described next.

Assume that we have B choices for quantizing each layer (i.e., 2 for INT4 or INT8). For a model with L layers, the search space of the ILP will be B^L. The goal of solving the ILP problem is to find the best bit configuration among these B^L possibilities that results in optimal trade-offs between model perturbation Ω, and user-specified constraints such as model size, BOPS, and latency. Each of these bit-precision settings could result in a different model perturbation. To make the problem tractable, we assume that the perturbations for each layer are independent of each other (i.e., $\Omega = \sum_{i=1}^{L} \Omega_i^{(b_i)}$, where $\Omega_i^{(b_i)}$ is the i-th layer's perturbation with b_i bit)[3]. This allows us to precompute the sensitivity of each layer separately, and it only requires BL computations. For the sensitivity metric, we use the Hessian based perturbation proposed in [69, Eq. 2.11], which is defined as

$$\Omega_i = \frac{1}{n_i} Tr(\mathbf{H}_i), \tag{7.8}$$

where n_i is the number of parameters in the $i-th$ layer and $\mathbf{H}_i$ is the Hessian matrix of loss w.r.t. the weight in the i-th layer. The ILP problem tries to find the right bit precision that minimizes this sensitivity, as follows:

$$\text{Objective:} \quad \min_{\{b_i\}_{i=1}^{L}} \sum_{i=1}^{L} \Omega_i^{(b_i)}, \tag{7.9}$$

$$\text{Subject to:} \quad \sum_{i=1}^{L} M_i^{(b_i)} \le \text{ Model Size Limit}, \tag{7.10}$$

$$\sum_{i=1}^{L} G_i^{(b_i)} \le \text{ BOPS Limit}, \tag{7.11}$$

$$\sum_{i=1}^{L} Q_i^{(b_i)} \le \text{ Latency Limit}. \tag{7.12}$$

Here, $M_i^{(b_i)}$ denotes the size of i-th layer with b_i bit quantization, $Q_i^{(b_i)}$ is the associated latency, and $G_i^{(b_i)}$ is the corresponding BOPS required for computing that layer. The latter

[3]Similar assumption can be found in [70, 69].

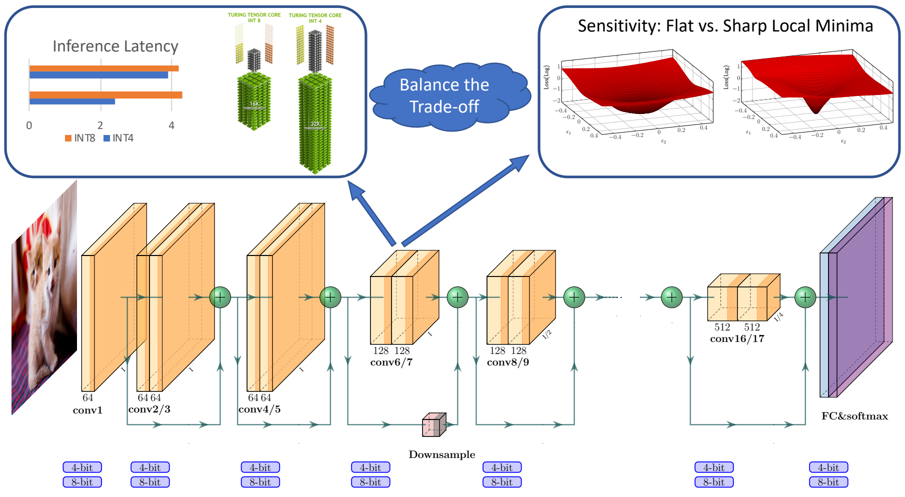

Figure 7.3: Illustration of inference speed and generalization performance trade-off of ResNet18. For each layer, we need to consider the speedup of INT4 vs INT8 and the sensitivity based on the second-order (Hessian) sharpness [69] of this layer.

measures the total Bit Operations for calculating a layer [11]:

$$G_i^{(b_i)} = b_{w_i} b_{a_i} \text{MAC}_i,$$

where MAC_i is the total Multiply-Accumulate operations for computing the i-th layer, and b_{w_i}, b_{a_i} are the bit precision used for weight and activation.[4] Note that it is not necessary to set all these constraints at the same time. Typically, which constraint to use depends on the end-user application.

We solve the ILP using open source PULP library [198] in Python, where we found that for all the configurations tested in the paper, the ILP solver can find the solution in less than 1 second given the sensitivity metric. For comparison, the RL based method of [235] could take tens of hours to find the right bit-precision setting. Meanwhile, as can be seen, our ILP solver can be easily used for multiple constraints. However, the complexity of Pareto frontier proposed by [69] is exponentially increasing for multiple constraints. In Section 7.4.2, we show the results with different constraints.

We should also mention that the contemporary work of [115], also proposed an ILP formulation. However, our approach is hardware-aware and we directly deploy and measure the latency of each layer in hardware.

[4] b_{w_i} and b_{a_i} are always the same in HAWQ-V3. As such, HAWQ-V3 does not need to cast lower-precision integer numbers, e.g., INT4, to higher-precision integer numbers, e.g., INT8, which is more efficient than [69, 33, 235].

7.3.5 Hardware Deployment

Model size alone is not a good metric to measure the efficiency (speed and energy consumption) of NNs. In fact, it is quite possible that a small model would have higher latency and consume a larger amount of energy for inference. The same is also true for FLOPs. The reason is that neither model size nor FLOPs can account for cache misses, data locality, memory bandwidth, underutilization of hardware, etc. To address this, we need to deploy and directly measure the latency.

We target Nvidia Turing Tensor Cores of T4 GPU for deployment, as it supports both INT8 and INT4 precision and has been enhanced for deep learning network inference. The only API available is the WMMA kernel call which is a micro-kernel for performing matrix-matrix operations in INT4 precision on Tensor Cores. However, there is also no existing compiler that would map a NN quantized to INT4 to Tensor Cores using WMMA instructions. To address this challenge, another contribution of our work is extending TVM [51] to support INT4 inference with/without mixed precision with INT8. This is important so we can verify the speed benefits of mixed-precision inference. To accomplish this, we had to add new features in both graph-level IR and operator schedules to make INT4 inference efficient. For instance, when we perform optimizations such as memory planning, constant folding, and operator fusion, at the graph-level IR, 4-bit data are involved. However, on byte-addressable machines, manipulating 4-bit data individually leads to inefficiency in storage and communication. Instead, we pack eight 4-bit elements into an INT32 data type and perform the memory movement as a chunk. In the final code generation stage, the data type and all memory access will be adjusted for INT32. By adopting similar scheduling strategies to Cutlass [172], we implement a new direct convolution schedule for Tensor Cores for both 8-bit and 4-bit data in TVM. We set the knobs for the configurations such as thread size, block size, and loop ordering so that the auto-tuner in TVM could search for the best latency settings.

Another important point is that we have completed the pipeline to test directly the trained weights and to avoid using random weights for speed measurements. This is important, since small discrepancies between the hardware implementation may go unnoticed from the quantization algorithm in the NN training framework (PyTorch in our case) which does not use TVM for the forward and backward propagation. To avoid any such issue, we made sure that the results between TVM and PyTorch match for every single layer and stage to machine-precision accuracy, and we verified the final Top-1 accuracy when executed in the hardware with integer-only arithmetic. In Appendix F.7, we present the error accumulation of feature maps for ResNet50 with INT4 quantization, which uses fake quantization in PyTorch and is deployed in TVM.

7.4 Results

In this section, we first discuss ImageNet results on various models (ResNet18/50 and InceptionV3) for INT8, INT4, and mixed-precision INT4/8 with/without distillation. Afterward, we study the different use cases of the ILP formulation, and the corresponding trade-offs between model size, latency, and accuracy. Detailed discussion on the implementation and set up is provided in Appendix F.8. For all the experiments we made sure to report and compare with the highest accuracy known for the baseline NN model in FP32 (i.e., we use a strong baseline for comparison). This is important since using a weak baseline accuracy could lead to misleading quantization accuracy.

Table 7.1: Quantization results for ResNet18/50 and InceptionV3. Here, we abbreviate Integer-Only Quantization as "Int", Uniform Quantization as "Uni", the Baseline Accuracy as "BL", Weight Precision and Activation Precision as "Precision", Model Size as "Size" (in MB), Bit Operations as "BOPS" (in G), and Top-1 Accuracy as "Top-1". Also, "WxAy" means weight with x-bit and activation with y-bit, and 4/8 means mixed precision with 4 and 8 bits. "MP" means mixed precision with bitwidth ranging from 1-bit to 8-bit, and "W1*" means the bitwidth is 1-bit but the network architecture is changed (by using more channels). Our result with/without distillation is represented as HAWQ-V3+Dist/HAWQ-V3.

	Method	Int	Uni	BL	Precision	Size	BOPS	Top-1
[ResNet18]	RVQuant [177]	✗	✗	69.91	W8A8	11.1	116	70.01
	HAWQ-V3	✓	✓	71.47	W8A8	11.1	116	**71.56**
	PACT [55]	✗	✓	70.20	W5A5	7.2	50	69.80
	LQ-Nets [262]	✗	✗	70.30	W4A32	5.8	225	70.00
	HAWQ-V3	✓	✓	71.47	W4/8A4/8	6.7	72	70.22
	HAWQ-V3+Dist	✓	✓	71.47	W4/8A4/8	6.7	72	**70.38**
	CalibTIB[115]	✗	✓	71.97	W4A4	5.8	34	67.50
	HAWQ-V3	✓	✓	71.47	W4A4	5.8	34	**68.45**
	Method	Int	Uni	BL	Precision	Size	BOPS	Top-1
[ResNet50]	Integer Only [120]	✓	✓	76.40	W8A8	24.5	247	74.90
	RVQuant [177]	✗	✗	75.92	W8A8	24.5	247	75.67
	HAWQ-V3	✓	✓	77.72	W8A8	24.5	247	**77.58**
	PACT [55]	✗	✓	76.90	W5A5	16.0	101	76.70
	LQ-Nets [262]	✗	✗	76.50	W4A32	13.1	486	76.40
	RVQuant [177]	✗	✗	75.92	W5A5	16.0	101	75.60
	HAQ [235]	✗	✗	76.15	WMPA32	9.62	520	75.48
	OneBitwidth [54]	✗	✓	76.70	W1*A8	12.3	494	76.70
	HAWQ-V3	✓	✓	77.72	W4/8A4/8	18.7	154	75.39
	HAWQ-V3+Dist	✓	✓	77.72	W4/8A4/8	18.7	154	**76.73**
	CalibTIB[115]	✗	✓	77.20	W4A4	13.1	67	73.70
	HAWQ-V3	✓	✓	77.72	W4A4	13.1	67	**74.24**
	Method	Int	Uni	BL	Precision	Size	BOPS	Top-1
[InceptionV3]	Integer Only [120]	✓	✓	78.30	W8A8	22.7	366	74.20
	RVQuant [177]	✗	✗	74.19	W8A8	22.7	366	74.22
	HAWQ-V3	✓	✓	78.88	W8A8	22.7	366	**78.76**
	Integer Only [120]	✓	✓	78.30	W7A7	20.1	280	73.70
	HAWQ-V3	✓	✓	78.88	W4/8A4/8	19.6	265	74.65
	HAWQ-V3+Dist	✓	✓	78.88	W4/8A4/8	19.6	265	**74.72**
	HAWQ-V3	✓	✓	78.88	W4A4	12.3	92	70.39

7.4.1 Low Precision Integer-Only Quantization Results

We first start with ResNet18/50 and InceptionV3 quantization on ImageNet, and compare the performance of HAWQ-V3 with other approaches, as shown in Table 7.1.

Uniform 8-bit Quantization. Our 8-bit quantization achieves similar accuracy compared to the baseline. Importantly, for all the models HAWQ-V3 achieves higher accuracy than the integer-only approach of [120]. For instance, on ResNet50, we achieve 2.68% higher accuracy as compared to [120]. This is in part due to our BN folding strategy that was described in Section 7.3.2.

Uniform 4-bit Quantization. To the best of our knowledge, 4-bit results of HAWQ-V3 are the first integer-only quantization results reported in the literature. The accuracy results for ResNet18/50, and InceptionV3 are quite high, despite the fact that all of the inference computations are restricted to be integer multiplication, addition, and bit shifting. While there is some accuracy drop, this should not be incorrectly interpreted that uniform INT4 is not useful. On the contrary, one has to keep in mind that certain use cases have strict latency and memory footprint limit for which this may be the best solution. However, higher accuracy can be achieved through mixed-precision quantization.

Mixed 4/8-bit Quantization. The mixed-precision results improve the accuracy by several percentages for all the models, while slightly increasing the memory footprint of the model. For instance, the mixed-precision result for ResNet18 is 1.88% higher than its INT4 counterpart with just a 1.9MB increase in model size. Further improvements are also possible with distillation (denoted as HAWQ-V3+Dist in the table). For ResNet50, the distillation can boost the mixed-precision by 1.34%. We found that distillation helps most for mixed-precision quantization, and we found little to no improvement for uniform INT8, or uniform INT4 quantization cases.[5]

Overall, the results show that HAWQ-V3 achieves comparable accuracy to prior quantization methods including both uniform and mixed-precision quantization (e.g., PACT, RVQuant, OneBitwidth, HAQ which use FP32 arithmetic, and/or non-standard bit precision such as 5 bits, or different bit-width for weights and activations). Similar observations hold for InceptionV3, as reported in Table 7.1.

7.4.2 Mixed-precision Results with Different Constraints

Here, we discuss various scenarios where different constraints could be imposed for quantization, and the interesting trade-offs associated with each scenario. The ILP problem in (7.9) has three constraints of model size, BOPS, and latency. We consider three different thresholds for each of the constraints and study how the ILP balances the trade-offs to obtain an optimal quantized model. We also focus on the case, where the practitioner is not satisfied with the performance of the INT4 quantization and wants to improve the performance (accuracy, speed, and model size) through mixed-precision quantization (INT4 and

[5]We used simple distillation without extensive tuning. One might be able to improve the results further with more sophisticated distillation algorithms.

Table 7.2: Mixed-precision quantization results for ResNet18 and ResNet50 with different constraints. Here, we abbreviate constraint level as "Level". Model Size as "Size", Bit Operations as "BOPS", the speedup as compared to INT8 results as "Speed", and Top-1 Accuracy as "Top-1", The last column of Top-1 represents results of HAWQ-V3 and HAWQ-V3+Dist.

		Level	Size (MB)	BOPS (G)	Speed	Top-1
[ResNet18]	INT8	–	11.2	114	1x	71.56
	Size	High	**9.9**	103	1.03x	71.20/71.59
		Medium	**7.9**	98	1.06x	70.50/71.09
		Low	**7.3**	95	1.08x	70.01/70.66
	BOPS	High	8.7	**92**	1.12x	70.40/71.05
		Medium	6.7	**72**	1.21x	70.22/70.38
		Low	6.1	**54**	1.35x	68.72/69.72
	Latency	High	8.7	92	**1.12x**	70.40/71.05
		Medium	7.2	76	**1.19x**	70.34/70.55
		Low	6.1	54	**1.35x**	68.56/69.72
	INT4	–	5.6	28	1.48x	68.45
		Level	Size (MB)	BOPS (G)	Speed	Top-1
[ResNet50]	INT8	–	24.5	247	1x	77.58
	Size	High	**21.3**	226	1.09x	77.38/ 77.58
		Medium	**19.0**	197	1.13x	75.95/76.96
		Low	**16.0**	168	1.18x	74.89/76.51
	BOPS	High	22.0	**197**	1.16x	76.10/76.76
		Medium	18.7	**154**	1.23x	75.39/76.73
		Low	16.7	**110**	1.30x	74.45/76.03
	Latency	High	22.3	199	**1.13x**	76.63/76.97
		Medium	18.5	155	**1.21x**	74.95/76.39
		Low	16.5	114	**1.28x**	74.26/76.19
	INT4	–	13.1	67	1.45x	74.24

INT8). The ILP formulation enables the practitioner to set each or all of these constraints. Here, we present results when only one of these constraints is set at a time. The results are shown in Table 7.2, which is split into three sections of Size (model size), BOPS, and Latency. Each section represents the corresponding constraint as specified by the practitioner. The ILP solver then finds the optimal mixed-precision setting to satisfy that constraint, while maximizing accuracy. See Appendix F.9 for the example of the latency constraint for

ResNet18.

We start with the model size and BOPS constraints for ResNet18. The model size of pure INT4 quantization is 5.6MB, and INT8 is 11.2MB. However, the accuracy of INT4 quantization is 68.45% which maybe low for a particular application. The practitioner then has the option to set the model size constraint to be slightly higher than pure INT4. One option is to choose 7.9MB which is almost in between INT4 and INT8. For this case, the ILP solver finds a bit-precision setting that results in 71.09% accuracy which is almost the same as INT8. This model is also 6% faster than INT8 quantization.

Another possibility is to set the speed/latency as a constraint. The results for this setting are represented under the “Latency” row in Table 7.2. For example, the practitioner could request the ILP to find a bit-precision setting that would result in 19% faster latency as compared to the INT8 model (see “Medium” row). This results in a model with an accuracy of 70.55% with a model size of only 7.2MB. A similar constraint could also be made for BOPS.

Several very interesting observations can be made from these results. (i) The correlation between model size and BOPS is weak which is expected. That is a larger model size does not mean higher BOPS and vice versa. For example, compare Medium-Size and High-BOPS for ResNet18. The latter has lower BOPS despite being larger (and is actually faster as well). (ii) The model size does not directly correlate with accuracy. For example, for ResNet50, High-BOPS has a model size of 22MB and accuracy of 76.76%, while High-Size has a smaller model size of 21.3MB but higher accuracy of 77.58%.

In summary, although directly using INT4 quantization may result in large accuracy degradation, we can achieve significantly improved accuracy with much faster inference as compared to INT8 results. This gives the practitioner a wider range of choices beyond just INT8 quantization. Finally, we should mention that the accuracy and speed for all of the results shown for ResNet18/50 and InceptionV3 have been verified by directly measuring them when executed in quantized precision in hardware through TVM. As such, these results are actually what the practitioner will observe, and these are not simulated results.

7.5 Conclusions

In this work, we presented HAWQ-V3, a new low-precision integer-only quantization framework, where the entire inference is executed with only integer multiplication, addition, and bit shifts. In particular, no FP32 arithmetic or even integer division is used in the entire inference. We presented results for uniform and mixed-precision INT4/8. For the latter, we proposed a hardware-aware ILP based method that finds the optimal trade-off between model perturbation and application specific constraints such as model size, inference speed, and total BOPS. The ILP problem can be solved very efficiently, under a second for all the models considered here. We showed that our approach can achieve up to 5% higher accuracy as compared to the prior integer-only approach of [120]. Finally, we directly implemented the low-precision quantized models in hardware by extending TVM to support INT4 and

INT4/8 inference. We verified all the results, by matching the activation of each layer with our PyTorch framework (up to machine precision), including the verification of the final accuracy of the model. The framework, the TVM implementation, and the quantized models have been open sourced [112].

Chapter 8

TRAttack: Trust Region Based Adversarial Attack on Neural Networks

8.1 Introduction

Deep Neural Networks (DNNs) have achieved impressive results in many research areas, such as classification, object detection, and natural language processing. However, recent studies have shown that DNNs are often not robust to adversarial perturbation of the input data [225, 89]. This has become a major challenge for DNN deployment, and significant research has been performed to address this. These efforts can be broadly classified into three categories: (i) research into finding strategies to defend against adversarial inputs (which has so far been largely unsuccessful); (ii) new attack methods that are stronger and can break the proposed defense mechanisms; and (iii) using attack methods as form of implicit adversarial regularization for training neural networks [253, 211, 251]. Our interest here is mainly focused on finding more effective attack methods that could be used in the latter two directions. Such adversarial attack methods can be broadly classified into two categories: white-box attacks, where the model architecture is known; and black-box attacks, where the adversary can only perform a finite number of queries and observe the model behaviour. In practice, white-box attacks are often not feasible, but recent work has shown that some adversarial attacks can actually transfer from one model to the other [164]. Therefore, precise knowledge of the target DNN may actually not be essential. Another important finding in this direction is the existence of an *adversarial patch*, i.e., a small set of pixels which, if added to an image, can fool the network. This has raised important security concerns for applications such as autonomous driving, where addition of such an adversarial patch to traffic signs could fool the system [29].

Relatedly, finding more *efficient* attack methods is important for evaluating defense strategies, and this is the main focus of our paper. For instance, the seminal work of [36]

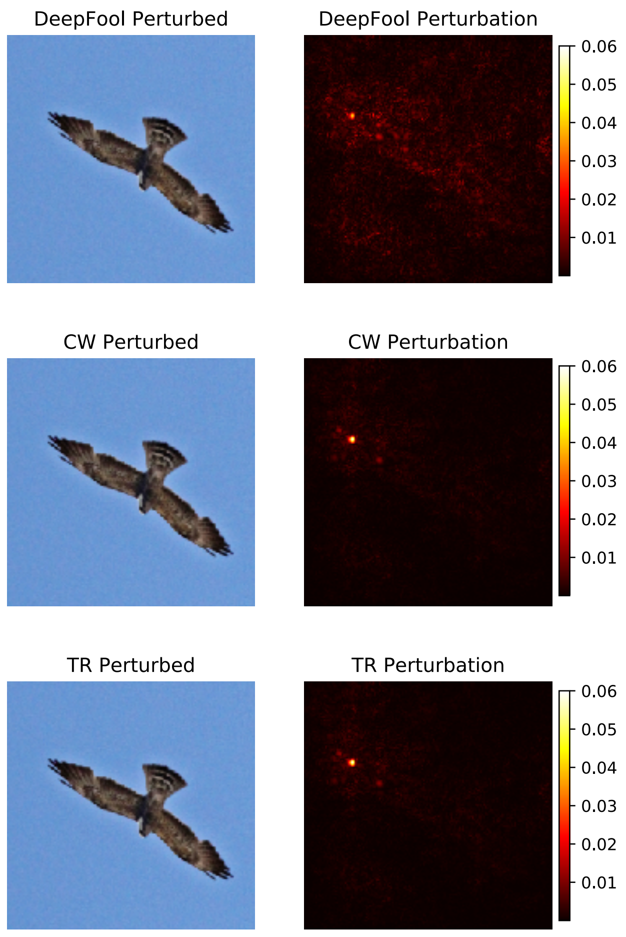

Figure 8.1: An example of DeepFool, CW, and our TR attack on AlexNet, with L_2 norm. Both CW and TR perturbations are smaller in magnitude and more targeted than DeepFool's (2× smaller here). TR attack obtains similar perturbation as CW, but **15**× faster. In the case of the VGG-16 network, we achieve an even higher speedup of **37.5**× (please see Figure 8.4 for timings).

introduced a new type of optimization based attack, commonly referred to as CW (Carlini-Wagner) attack, which illustrated that defensive distillation [175] can be broken with its stronger attack. The latter approach had shown significant robustness to the fast gradient sign attack method [89], but not when tested against the stronger CW attack. Three metrics for an *efficient attack* are the speed with which such a perturbation can be computed, the magnitude or norm of the perturbation that needs to be added to the input to fool the network, and the transferability of the attack to other networks. Ideally (from the perspective of the attacker), a stronger attack with a smaller perturbation magnitude is desired, so that it could be undetectable (e.g. an adversarial patch that is harder to detect by humans).

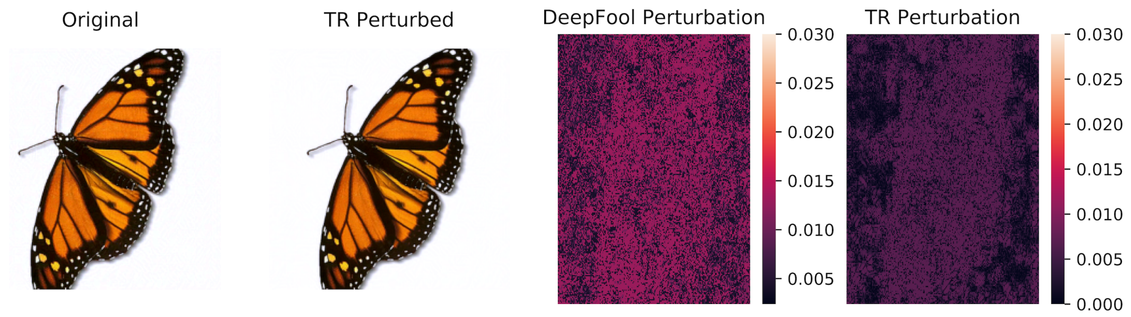

Figure 8.2: An example of DeepFool and TR attack on VGG-16, with L_∞ norm. The first pattern is the original image. The second pattern is the image after TR attack. The final two patterns are the perturbations generated by DeepFool and TR. The TR perturbation is smaller than DeepFool's (1.9× smaller). Also, the TR perturbation is more concentrated around the butterfly.

In this work, we propose a novel trust region based attack method. Introduced in [221, 57], trust region (TR) methods form a family of numerical optimization methods for solving non-convex optimization problems [171]. The basic TR optimization method works by first defining a region, commonly referred to as *trust region*, around the current point in the optimization landscape, in which a (quadratic) model approximation is used to find a descent/ascent direction. The idea of using this confined region is due to the model approximation error. In particular, the trust region method is designed to improve upon vanilla first-order and second-order methods, especially in the presence of non-convexity.

We first consider a first-order TR method, which uses gradient information for attacking the target DNN model and adaptively adjusts the trust region. The main advantage of first-order attacks is their computational efficiency and ease of implementation. We show that our first-order TR method significantly reduces the over-estimation problem (i.e. requiring very large perturbation to fool the network), resulting in up to 3.9× reduction in the perturbation magnitude, as compared to DeepFool [163]. Furthermore, we show TR is significantly faster than the CW attack (up to 37×), while achieving similar attack performance. We then propose an adaptive TR method, where we adaptively choose the TR radius based on the model approximation to further speed up the attack process. Finally, we present the formulation for how our basic TR method could be extended to a second-order TR method, which could be useful for cases with significant non-linear decision boundaries, e.g., CNNs with Swish activation function [188]. In more detail, our main contributions are the following:

• We cast the adversarial attack problem into the optimization framework of TR methods. This enables several new attack schemes, which are easy to implement and are significantly more effective than existing attack methods (up to 3.9×, when compared to DeepFool). Our method requires a similar perturbation magnitude, as compared to CW, but it can compute the perturbation significantly faster (up to 37×), as it does not require extensive hyper-parameter tuning.

• Our TR-based attack methods can adaptively choose the perturbation magnitude in every iteration. This removes the need for expensive hyper-parameter tuning, which is a major issue with the existing optimization based methods.

• Our method can easily be extended to second-order TR attacks, which could be useful for non-linear activation functions. With fewer iterations, our second-order attack method outperforms the first-order attack method.

Limitations. We believe that it is important for every work to state its limitations (in general, but in particular in this area). We paid special attention to repeat the experiments multiple times, and we considered multiple different DNNs on different datasets to make sure the results are general. One important limitation of our approach is that a naïve implementation of our second-order method requires computation of Hessian matvec backpropogation, which is very expensive for DNNs. Although the second-order TR attack achieves better results, as compared to the first-order TR attack, this additional computational cost could limit its usefulness in certain applications. Moreover, our method achieves similar results as CW attack, but significantly faster. However, if we ignore the strength of the attack, then the DeepFool attack is faster than our method (and CW's for that matter). Although such comparison may not be fair, as our attack is stronger. However, this may be an important point for certain applications where maximum speed is needed.

8.2 Background

In this section, we review related work on adversarial attacks. Consider $\mathbf{x} \in \mathbb{R}^n$ as input image, and $\mathbf{y} \in \mathbb{R}^c$ the corresponding label. Suppose $\mathbf{M}(\mathbf{x};\theta) = \hat{\mathbf{y}}$ is the DNN's prediction probabilities, with θ the model parameters and $\hat{\mathbf{y}} \in \mathbb{R}^c$ the vector of probabilities. We denote the loss function of a DNN as $\mathcal{L}(\mathbf{x}, \theta, \mathbf{y})$. Then, an adversarial attack is aimed to find a (small) perturbation, $\Delta\mathbf{x}$, such that:

$$argmax(\mathbf{M}(\mathbf{x} + \Delta\mathbf{x};\theta)) = argmax(\hat{\mathbf{y}}) \neq argmax(\mathbf{y}).$$

There is no closed form solution to analytically compute such a perturbation. However, several different approaches have been proposed by solving auxiliary optimization or analytical approximations to solve for the perturbation. For instance, the Fast Gradient Sign Method (FGSM) [89] is a simple adversarial attack scheme that works by directly maximizing the loss function $\mathcal{L}(\mathbf{x}, \theta, \mathbf{y})$. It was subsequently extended to an iterative FGSM [130], which performs multiple gradient ascend steps to compute the adversarial perturbation, and is often more effective than FGSM in attacking the network. Another interesting work in this direction is DeepFool, which uses an approximate analytical method. DeepFool assumes that the neural network behaves as an affine multiclass classifier, which allows one to find a closed form solution. DeepFool is based on "projecting" perturbed inputs to cross the decision boundary (schematically shown in Figure 8.3), so that its classification is changed, and this was shown to outperform FGSM. However, the landscape of the decision boundary of a neural network is not linear. This is the case even for ReLU networks with the softmax layer. Even before the softmax layer, the landscape is *piece-wise linear*, but this cannot be approximated with a simple affine transformation. Therefore, if we use the local information, we can overestimate/underestimate the adversarial perturbation needed to fool the network.

The seminal work of [36] introduced the so-called CW attack, a more sophisticated way to directly solve for the $\Delta\mathbf{x}$ perturbation. Here, the problem is cast as an optimization problem, where we seek to minimize the distance between the original image and the perturbed one, subject to the constraint that the perturbed input would be misclassified by the neural network. This work also clearly showed that defensive distillation, which at the time was believed to be a robust method to defend against adversaries, is not robust to stronger attacks. One major disadvantage of the CW attack is that it is very sensitive to hyper-parameter tuning. This is an important problem in applications where speed is important, as finding a good/optimal adversarial perturbation for a given input is very time consuming. Addressing this issue, without sacrificing attack strength, is a goal of our work.

On another direction, adversarial training has been used as a defense method against adversarial attacks [211]. In particular, by using adversarial examples during training, one can obtain models that are more robust to attacks (but still not foolproof). This adversarial training was further extended to ensemble adversarial training [228], with the goal of making the model more robust to black box attacks. Other approaches have also been proposed to detect/defend against adversarial attacks [175, 158]. However, it has recently been shown that, with a stronger attack method, defense schemes such as distillation or obfuscated gradients can be broken [34, 36, 9].

A final important application of adversarial attacks is to train neural networks to obtain improved generalization, even in non-adversarial environments. Multiple recent works have shown that adversarial training (specifically, training with mixed adversarial and clean data) can be used to train a neural network from scratch in order to achieve a better final *generalization* performance [211, 205, 253, 251]. In particular, the work of [251] empirically showed that using adversarial training would result in finding areas with "flatter" curvature. This property has recently been argued to be an important parameter for generalization performance [124]. Here, the speed with which adversarial perturbations can be computed

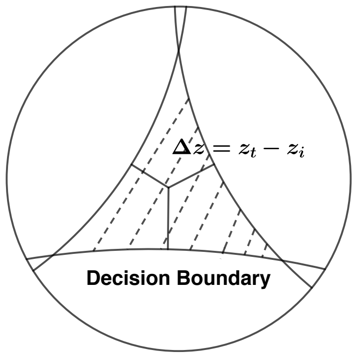

Figure 8.3: Schematic illustration of the decision boundaries for a classification problem. Points mapped to the hashed region, are classified with the same label.

is very important since it appears in the inner loop of the training process, and the training needs to be performed for many epochs.

8.3 Trust Region Adversarial Attack

Let us denote the output of the DNN before the softmax function to be $\mathbf{z} = \mathbf{Z}(\mathbf{x};\theta) \in \mathbb{R}^c$. Therefore, we will have:

$$\mathbf{M}(\mathbf{x};\theta) = softmax\big(\mathbf{Z}(\mathbf{x};\theta)\big) = \hat{\mathbf{y}}.$$

Denote $\mathbf{y}_t$ to be the true label of $\mathbf{x}$, and $\mathbf{z}_t = \arg\max \mathbf{z}$ to be the prediction output of $\mathbf{M}(\mathbf{x};\theta)$. For clarification, note that $\mathbf{z}_t$ is only the same as $\mathbf{y}_t$ if the neural network makes the correct classification. An adversarial attack seeks to find $\Delta\mathbf{x}$ that fools the DNN, that is:

$$\arg\min_{\|\Delta\mathbf{x}\|_p} \arg\max \mathbf{Z}(\mathbf{x}+\Delta\mathbf{x};\theta) \neq \mathbf{y}_t, \tag{8.1}$$

where $\|\cdot\|_p$ denotes the L_p norm of a vector. It is often computationally infeasible to solve (8.1) exactly. Therefore, a common approach is to approximately solve (8.1) [36, 89, 225]. To do so, the problem can be formulated as follows:

$$\max_{\|\Delta\mathbf{x}\|_p \leq \epsilon} \mathcal{J}(\mathbf{x}+\Delta\mathbf{x},\theta,\mathbf{y}), \tag{8.2}$$

where ϵ constrains the perturbation magnitude, and $\mathcal{J}$ can be either the loss function ($\mathcal{L}$) or more generally another kernel [36]. In the case of DeepFool (DF) attack, this problem is solved by approximating the decision boundary by a linear affine transformation. For such a decision boundary, the perturbation magnitude could be analytically computed by just evaluating the gradient at the current point. However, for neural networks this approximation could be very inaccurate, that is it could lead to over/under-estimation of the perturbation

Algorithm 7 Trust Region Attack

```
1: **Input**:
       - Image $\mathbf{x}^0$, label $\mathbf{y}$
       - initial radius $\epsilon^0$, threshold $\sigma_1$ and $\sigma_2$, radius adjustment rate $\eta$
2: $\Delta\mathbf{x} = 0,\ j = 0$
3: Using scheme to choose the attacking index $i$
4: **while** $\arg\max \mathbf{Z}(\mathbf{x}^j) = \arg\max \mathbf{y}$ **do**
5:     $\Delta\mathbf{x}_{tmp} = \arg\min_{\|\Delta\mathbf{x}^j\| \le \epsilon^j} m^j(\Delta\mathbf{x}^j) = \arg\min_{\|\Delta\mathbf{x}^j\| \le \epsilon^j} \langle \Delta\mathbf{x}^j, \mathbf{g}^j_{t,i}\rangle + \frac{1}{2}\langle \Delta\mathbf{x}^j, \mathbf{H}^j_{t,i}\Delta\mathbf{x}^j\rangle$
6:     $\mathbf{x}^{j+1} = clip(\mathbf{x}^j + \Delta\mathbf{x}_{tmp}, \min(\mathbf{x}), \max(\mathbf{x}))$
7:     $\rho = \frac{(\mathbf{z}_t^{j+1} - \mathbf{z}_i^{j+1}) - (\mathbf{z}_t^{j+1} - \mathbf{z}_i^{j+1})}{m^j(\Delta\mathbf{x}^j)}$
8:     **if** $\rho > \sigma_1$ **then**
9:         $\epsilon^{j+1} = \min\{\eta\epsilon^j, \epsilon_{\max}\}$
10:    **else if** $\rho < \sigma_2$ **then**
11:        $\epsilon^{j+1} = \min\{\epsilon^j/\eta, \epsilon_{\min}\}$
12:    **else**
13:        $\epsilon^{j+1} = \epsilon^j$
14:    **end if**
15:    $j = j + 1$
16: **end while**
17: **Output**: Adversarial Image
```

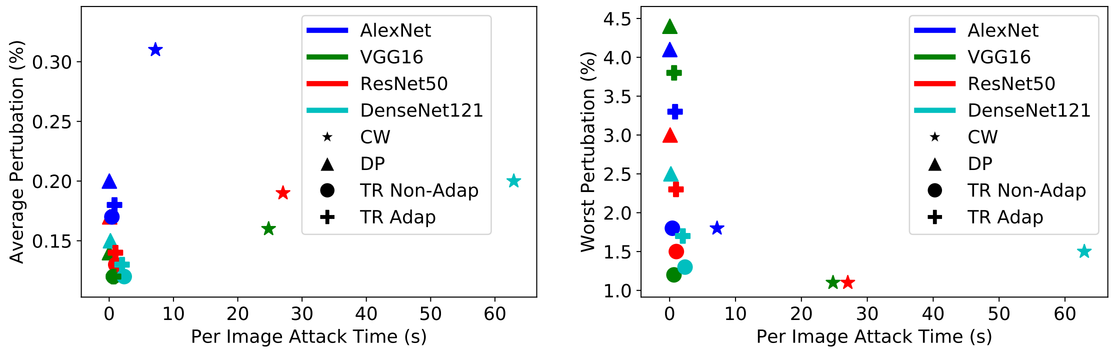

Figure 8.4: The two subfigures show, for various neural networks, the time to compute the adversarial attack (x-axis) and the perturbation needed by that attack method to fool an image (y-axis), corresponding to ImageNet results in Table 8.3. On the left, the y-axis is plotted for average perturbation; and on the right, for the worst case perturbation. An attack that achieves smaller perturbation in shorter time is preferred. Different colors represent different models, and different markers illustrate the different attack methods. Observe that our TR and TR Adap methods achieve similar perturbations as CW but with significantly less time (up to **37.5**×).

along a sub-optimal direction. The smallest direction would be orthogonal to the decision boundary, and this cannot be computed by a simple affine transformation, as the decision boundary is non-linear (see Fig. 8.3 for illustration). This is obvious for non-linear activation functions, but even in the case of ReLU, the model behaves like a piece-wise linear function (before the softmax layer, and actually non-linear afterwards). This approach cannot correctly find the orthogonal direction, even if we ignore the non-linearity of the softmax layer.

To address this limitation, we instead use TR methods, which are well-known for solving non-convex optimization problems [221]. The problem of finding adversarial perturbation using TR is defined as follows:

$$\max_{\|\Delta \mathbf{x}^j\|_p \le \epsilon^j} m^j(\Delta \mathbf{x}^j) = \langle \Delta \mathbf{x}^j, \mathbf{g}^j_{t,i} \rangle + \frac{1}{2}\langle \Delta \mathbf{x}^j, \mathbf{H}^j_{t,i} \Delta \mathbf{x}^j \rangle, \tag{8.3}$$

where ϵ^j is the TR radius at j^{th} iteration, m^j is the approximation of the kernel function of $f(\mathbf{x}^{j-1}) = (\mathbf{z}_t^{j-1} - \mathbf{z}_i^{j-1})$ with $\mathbf{g}^j_{t,i}$ and $\mathbf{H}^j_{t,i}$ denoting the corresponding gradient and Hessian, and $\mathbf{x}^j = \mathbf{x} + \sum_{i=1}^{j-1} \Delta \mathbf{x}^i$. Note that the subscript means the index of maximal $\mathbf{z}$, i.e. $\arg\max \mathbf{z}$. The main idea of the TR method is to iteratively select the trusted radius ϵ^j to find the adversarial perturbation within this region such that the probability of an incorrect class becomes maximum. TR adjusts this radius by measuring the approximation of the local model $m^j(\mathbf{s}^j)$ to the actual function value $f(\mathbf{x}^{j+1}) - f(\mathbf{x}^j)$. In particular, we increase the trusted radius if the approximation of the function is accurate (measured by $\rho = \frac{f(\mathbf{x}^{j+1}) - f(\mathbf{x}^j)}{m^j(\mathbf{s}^j)} > \sigma_1$ with a typical value of $\sigma_1 = 0.9$). In such a case, the trusted radius is increased for the next iterations by a factor of $\eta > 1$ ($\epsilon^{j+1} = \eta \epsilon^t$). However, when the local model $m^j(\mathbf{s}^j)$ is a poor approximation of $f(\mathbf{x}^{j+1}) - f(\mathbf{x}^j)$, i.e., $\rho = \frac{f(\mathbf{x}^{j+1}) - f(\mathbf{x}^j)}{m^j(\mathbf{s}^j)} < \sigma_2$ (with a typical $\sigma_2 = 0.5$), we decrease the trusted radius for the next iteration $\epsilon^{j+1} = \epsilon^t/\eta$. Otherwise, we keep the same ϵ^j for ϵ^{j+1}. Typically, a threshold is also used for lower and upper bounds of ϵ^j. Using this approach, the TR attack can iteratively find an adversarial perturbation to fool the network. See Alg. 7 for details.

Note that for cases where all the activations of the DNN are ReLU, the Hessian is zero almost everywhere [251, Theorem 1], and we actually do not need the Hessian. This means the landscape of $\mathbf{z}_t - \mathbf{z}_i$ is piece-wise linear, i.e., we could omit $\mathbf{H}^j_{t,i}$ in (8.3). However, for non-linear activation functions, we need to keep the Hessian term (since when the NN has smooth activation functions, the Hessian is not zero). For these cases, the problem of finding the adversarial perturbation becomes a Quadratic Constrained Quadratic Programming (QCQP) problem. It is quadratic constraint due to the fact that the norm of the perturbation is limited by the TR radius, η^j, and the quadratic programming arises from the non-zero Hessian term. We use Lanczos algorithm to solve the QCQP problem. In this approach, the solution is iteratively found in a Krylov subspace formed by the Hessian operator.

Table 8.1: Average perturbations / worst case perturbations are reported of different models on Cifar-10 for best class attack.. Lower values are better. The first set of rows show L_2 attack and the second shows L_∞ attack.

		DeepFool	CW	TR Non-Adap	TR Adap
Model	Accuracy	ρ_2	ρ_2	ρ_2	ρ_2
AlexLike	85.78	1.67% / 11.5%	1.47% / 9.70%	1.49% / 9.13%	1.49% / 9.09%
AlexLike-S	86.53	1.74% / 11.0%	1.57% / 8.59%	1.57% / 9.48%	1.57% / 9.46%
ResNet	92.10	0.80% / 5.60%	0.62% / 3.12%	0.66% / 3.97%	0.66% / 3.96%
WResNet	94.77	0.89% / 5.79%	0.66% / 4.51%	0.73% / 4.43%	0.72% / 4.34%
		DeepFool	FGSM	TR Non-Adap	TR Adap
Model	Accuracy	ρ_∞	ρ_∞	ρ_∞	ρ_∞
AlexLike	85.78	1.15% / 6.85%	1.40% / 16.44%	1.05% / 5.45%	1.03% / 5.45%
AlexLike-S	86.53	1.18% / 6.01%	1.45% / 14.88%	1.09% / 4.76%	1.07% / 4.73%
ResNet	92.10	0.60% / 3.98%	0.85% / 4.35%	0.56% / 3.18%	0.50% / 3.35%
WResNet	94.77	0.66% / 3.34%	0.85% / 3.30%	0.56% / 2.67%	0.54% / 2.69%

8.4 Performance of the Method

To test the efficacy of the TR attack method and to compare its performance with other approaches, we perform multiple experiments using different models on Cifar-10 [129] and ImageNet [65] datasets. In particular, we compare to DeepFool [163], iterative FGSM [89, 130], and the Carlini-Wagner (CW) attack [36].

As mentioned above, the original TR method adaptively selects the perturbation magnitude. Here, to test how effective the adaptive method performs, we also experiment with a case where we set the TR radius to be a fixed small value and compare the results with the original adaptive version. We refer to the fixed radius version as "TR Non-Adap" and the adaptive version as "TR Adap". Furthermore, the metric that we use for performance of the attack is the relative perturbation, defined as follows:

$$\rho_p = \frac{\|\Delta \mathbf{x}\|_p}{\|\mathbf{x}\|_p}, \tag{8.4}$$

where $\Delta\mathbf{x}$ is the perturbation needed to fool the testing example. The perturbation is chosen such that the accuracy of the model is reduced to less than 0.1%. We report both the average perturbation as well as the highest perturbation required to fool a testing image. To clarify this, the highest perturbation is computed after all of testing images (50K in ImageNet and 10K in Cifar-10) and then finding the the highest perturbation magnitude that was needed to fool a correctly classified example. We refer to this case as *worst case* perturbation. Ideally we would like this worst case perturbation to be bounded and close to the average cases.

Table 8.2: Average perturbations / worst case perturbations are reported of different models on Cifar-10 for hardest class attack. Lower values are better. The first set of rows show L_2 attack and the second shows L_∞ attack.

	DeepFool	TR Non-Adap	TR Adap
Model	ρ_2	ρ_2	ρ_2
AlexLike	4.36% /18.9%	2.47% /13.4%	2.47% /13.4%
AlexLike-S	4.70% /17.7%	2.63% /14.4%	2.62% /14.2%
ResNet	1.71% /8.01%	0.99% /4.76%	0.99% /4.90%
WResNet	1.80% /8.74%	1.05% /6.23%	1.08% /6.23%
Model	ρ_∞	ρ_∞	ρ_∞
AlexLike	2.96% /12.6%	1.92% /9.99%	1.86% /10.0%
AlexLike-S	3.12% /12.2%	1.98% /8.19%	1.92% /8.17%
ResNet	1.34% /9.65%	0.77% /4.70%	0.85% /5.44%
WResNet	1.35% /6.49%	0.81% /3.77%	0.89% /3.90%

Table 8.3: Average perturbations / worst case perturbations are reported of different models on ImageNet for best class attack. Lower values are better. The first set of rows show L_2 attack and the second shows L_∞ attack.

		DeepFool	CW	TR Non-Adap	TR Adap
Model	Accuracy	ρ_2	ρ_2	ρ_2	ρ_2
AlexNet	56.5	0.20% / 4.1%	0.31% / 1.8%	0.17% / 2.5%	0.18% / 3.3%
VGG16	71.6	0.14% / 4.4%	0.16% / 1.1%	0.12% / 1.2%	0.12% / 3.8%
ResNet50	76.1	0.17% / 3.0%	0.19% / 1.1%	0.13% / 1.5%	0.14% / 2.3%
DenseNet121	74.4	0.15% / 2.5%	0.20% / 1.5%	0.12% / 1.3%	0.13% / 1.7%
		DeepFool	FGSM	TR Non-Adap	TR Adap
Model	Accuracy	ρ_∞	ρ_∞	ρ_∞	ρ_∞
AlexNet	56.5	0.14% / 4.3%	0.16% / 4.7%	0.13% / 1.4%	0.13% / 3.6%
VGG16	71.5	0.11% / 4.0%	0.18% / 5.1%	0.10% / 1.4%	0.10% / 3.4%
ResNet50	76.1	0.13% / 3.2%	0.18% / 3.7%	0.11% / 1.3%	0.11% / 2.7%
DenseNet121	74.4	0.11% / 2.3%	0.15% / 4.1%	0.10% / 1.1%	0.10% / 1.8%

8.4.1 Setup

We consider multiple different neural networks including variants of (wide) residual networks [104, 259], AlexNet, VGG16 [217], and DenseNet from [113]. We also test with custom smaller/shallower convolutional networks such as a simple CNN [251, p. C1] (refer as AlexLike with ReLU and AlexLike-S with Swish activation). To test the second-order attack method we run experiments with AlexNet-S (by replacing all ReLUs with Swish function activation function [189]), along with a simple MLP ($3072 \to 1024 \to 512 \to 512 \to 256 \to 10$) with Swish activation function.

By definition, an adversarial attack is considered successful if it is able to change the classification of the input image. Here we perform two types of attacks. The first one is where we compute the smallest perturbation needed to change the target label. We refer to this as *best class* attack. This means we attack the class with:

$$\underset{j}{argmin} \frac{z_t - z_j}{\|\nabla_{\mathbf{x}}(z_t - z_j)\|}.$$

Intuitively, this corresponds to perturbing the input to cross the closest decision boundary (Figure 8.3). On the other hand, we also consider perturbing the input to the class whose decision boundary is farthest away:

$$\underset{j}{argmax} \frac{z_t - z_j}{\|\nabla_{\mathbf{x}}(z_t - z_j)\|}.$$

Furthermore, we report two perturbation metrics of *average perturbation*, computed as:

$$\rho_p = \frac{1}{N}\sum_{i=1}^{N} \frac{\|\Delta \mathbf{x}_i\|_p}{\|\mathbf{x}_i\|_p},$$

along with worst perturbation, computed as:

$$\rho_p = \max\{\frac{\|\Delta \mathbf{x}_i\|_p}{\|\mathbf{x}_i\|_p}\}_{i=1}^{N}.$$

For comparison, we also consider the following attack methods:

- Iterative FGSM from [89, 130], where the following formula is used to compute adversarial perturbation, after which the perturbation is clipped in range $(\min(\mathbf{x}), \max(x))$:
$$\mathbf{x}^{j+1} = \mathbf{x}^j + \epsilon \; sign(\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}^j, \theta, \mathbf{y})),$$
- DeepFool (DF) from [163]. We follow the same implementation as [163]. For the hardest class test, the target class is set the same as our TR method.
- CW attack from [36]. We use the open source code from [191][1].

Finally, we measure the time to fool an input image by averaging the attack time over all the testing examples. The measurements are performed on a Titan Xp GPU with an Intel E5-2640 CPU.

[1]https://github.com/bethgelab/foolbox

Table 8.4: Average perturbations / worst case perturbations are reported of different models on ImageNet for hardest class attack (on the top 100 prediction classes). Lower values are better. The first set of rows show L_2 attack and the second shows L_∞ attack.

	DeepFool	TR Non-Adap	TR Adap
Model	ρ_2	ρ_2	ρ_2
AlexNet	0.74% /8.7%	0.39% /5.0%	0.39% /5.0%
VGG16	0.45% /5.4%	0.27% /3.6%	0.27% /3.8%
ResNet50	0.52% /5.8%	0.31% /4.2%	0.31% /4.2%
DenseNet	0.48% /5.7%	0.29% /3.8%	0.29% /3.8%
Model	ρ_∞	ρ_∞	ρ_∞
AlexNet	0.53% /9.9%	0.31% /7.5%	0.33% /9.1%
VGG16	0.36% /11.6%	0.25% /5.1%	0.26% /6.8%
ResNet50	0.43% /6.6%	0.28% /3.7%	0.30% /4.6%
DenseNet	0.38% /6.4%	0.24% /4.5%	0.27% /5.7%

Table 8.5: second-order and first-order comparison on MLP and AlexNet with Swish activation function on Cifar-10. The corresponding baseline accuracy without adversarial perturbation is 62.4% and 76.6%, respectively. As expected, the second-order TR attack achieves better results as compared to first-order with fixed iterations. However, the second-order attack is significantly more expensive, due to the overhead of solving QCQP problem.

Iter		1	2	3	4	5	6	7	8	9	10
MLP	TR First	47.63	33.7	22.24	13.76	8.13	4.59	2.41	1.31	0.63	0.27
MLP	TR Second	47.84	33.37	21.49	13.3	7.39	4.16	2.17	1.09	0.49	0.20
AlexNet	TR First	51.51	28.17	12.45	5.53	2.61	1.33	0.82	0.66	0.51	0.46
AlexNet	TR Second	50.96	26.97	10.73	4.11	1.79	0.91	0.67	0.54	0.47	0.44

8.4.2 Cifar-10

We first compare different attacks of various neural network models on Cifar-10 dataset, as reported in Table 8.1. Here, we compute the average and worst case perturbation for best class attack. For L_2 attack, we can see that TR Non-Adap can achieve comparable perturbation as CW, with both TR and CW requiring smaller perturbation than DeepFool. An important advantage of the TR attack is its speed, as compared to CW attack, which is illustrated in Figure G.1 (please see appendix). Here we plot the time spent to fool one input image versus average perturbation for all L_2 attack methods on different models. It can be clearly seen that, with similar perturbations, the time to get the adversarial examples is: TR $<$ CW. Note that DeepFool is also very fast but requires much larger perturbations than TR attack and CW. Also note that the TR Adap method achieves similar results, with slightly slower speed and slightly larger perturbation. This is because the adaptive method has not been tuned any way, whereas for the non adaptive version we manually tuned ϵ. TR Adap does not require tuning, as it automatically adjust the TR radius. The slight performance degradation is due to the relaxed σ_1 and σ_2 parameters, which could be made more conservative as a trade-off for speed. But we did not tune these parameters beyond the default, to give a realistic performance for the non-tuned version.

Another important test is to measure the perturbation needed to fool the network to the hardest target class. This is important in that flipping a pedestrian to a cyclist may be easier than flipping it to become a car. In Table 8.2, we report the hardest class attack on Cifar-10. Note that our methods are roughly 1.5 times better than DeepFool in all cases. Particularly, For L_2 attack on WResNet, our worst case is 3.9 times better than DeepFool in terms of perturbation magnitude.

8.4.3 ImageNet Result

We observe similar trends on ImageNet. We report different attacks on various models on ImageNet in Table 8.3. Note that TR and CW require significantly smaller perturbation for the worst case as compared to DeepFool. However, TR is significantly faster than CW. The timing results are shown in Figure 8.4. For instance in the case of VGG-16, TR attack is $\mathbf{37.5}\times$ faster than CW which is significant. An example perturbation with AlexNet is shown in Figure 8.1 (for which TR is $\mathbf{15}\times$ faster). As one can see, CW and TR perturbations are smaller than DeepFool ($2\times$ in this case), and more targeted around the object. For L_∞ methods, our TR Non-Adap and TR Adap are consistently better than FGSM and DeepFool in both average and worst cases. Particularly, for worst cases, TR is roughly two times better than the other methods. An example perturbation of DeepFool and TR Non-Adap with L_∞ on VGG16 is shown in Figure 8.2. It can be clearly seen that, TR perturbation is much smaller than DeepFool ($1.9\times$ in this case), and more targeted around the objective.

8.4.4 second-order method

As mentioned in Section 8.3, the ReLU activation function does not require Hessian computation. However, for non-linear activation functions including Hessian information is beneficial, although it may be very expensive. To test this hypothesis, we consider two models with Swish activation function. We fix the TR radius (set to be 1 for all cases) of our first and second-order methods, and gradually increase the iterations. Table 8.5 shows the results for MLP and AlexNet models. It can be seen that second-order TR out-performs the first-order TR method in all iterations. Particularly, for two and three iterations on AlexNet, TRS can drop the model accuracy 1.2% more as compared to the first-order TR variant. However, the second order based model is more expensive than the first-order model, mainly due to the overhead associated with solving the QCQP problem. There is no closed form solution for this problem because the problem is non-convex and the Hessian can contain negative spectrum. Developing a computationally efficient method for this is an interesting next direction.

8.5 Conclusions

We have considered various TR based methods for adversarial attacks on neural networks. We presented the formulation for the TR method along with results for our first/second-order attacks. We considered multiple models on Cifar-10 and ImageNet datasets, including variants of residual and densely connected networks. Our method requires significantly smaller perturbation (up to $3.9\times$), as compared to DeepFool. Furthermore, we achieve similar results (in terms of average/worst perturbation magnitude to fool the network), as compared to the CW attack, but with significant speed up of up to $37.5\times$. For all the models considered, our attack method can bring down the model accuracy to less than 0.1% with relative small perturbation (in L_2/L_∞ norms) of the input image. Meanwhile, we also tested the second-order TR attack by backpropogating the Hessian information through the neural network, showing that it can find a stronger attack direction, as compared to the first-order variant.

Chapter 9

Conclusion

9.1 Summary

In this dissertation, we presented variants of the usage of second-order methods including solving non-convex optimization, analyzing the training procedure and design ingredients of deep learning architectures, measuring the sensitivity of low-precision neural networks, and adversarial attacking deep learning models. In order to apply second-order methods to large-scale optimization and/or machine learning problems, we incorporated sampling gradient and/or Hessian algorithms for non-convex optimization, and we combined randomized numerical linear algebra algorithms to efficiently compute Hessian-based metrics for deep learning (neural networks) analysis and applications.

In Part I, we considered optimization algorithms for non-convex problems. Specifically, in Chapter 2, we proposed *inexact* variants of trust region and adaptive cubic regularization methods, which, to increase efficiency, incorporate various approximations. In particular, in addition to *inexact sub-problem solves*, both the *gradient and Hessian are suitably estimated*. Using certain conditions on such approximations, we showed that our proposed inexact methods achieve similar *optimal worst-case iteration complexities* as the exact counterparts. In the context of finite-sum problems, we then explored randomized sub-sampling methods as ways to construct the gradient and Hessian approximations and examine the empirical performance of our algorithms on some model problems. We empirically demonstrated that our proposed algorithms are *practically implementable* in that failure to precisely fine-tune the associated hyper-parameters is unlikely to result in unwanted behaviors, e.g., divergence or stagnation. In Chapter 3, We considered variants of the Newton-CG algorithm for non-convex problems proposed in [199] in which only inexact estimates of the gradient and the Hessian information are available. Under certain conditions on the inexactness measures, we derived iteration complexity for achieving ϵ-approximate second-order optimality that matches best-known lower bounds. Our inexactness condition on the gradient was adaptive, allowing for crude accuracy in regions with large gradient. We described two variants of our approach, one in which the step-size along the computed search direction is chosen adaptively and another in which the step-size is pre-defined. We evaluated the performance of our proposed algorithms empirically on several machine learning models. In Chapter 4, we introduced AdaHessian, a new stochastic optimization algorithm. AdaHessian directly incorporated approximate curvature information from the loss function, and it included several novel performance-improving features, including: (i) a fast Hutchinson based method to approximate the curvature matrix with low computational overhead; (ii) a spatial averaging to reduce the variance of the second derivative; and (iii) a root-mean-square exponential moving average to smooth out variations of the second-derivative across different iterations. We performed extensive tests on NLP, CV, and recommendation system tasks, and AdaHessian achieved state-of-the-art results.

In Part II, we explored the usage of second-order methods for analyzing neural networks. In Chapter 5, we studied large batch size training through the lens of the Hessian operator and robust optimization. In particular, we performed a Hessian-based study to analyze exactly how the landscape of the loss function changes when training with a large batch size. We computed the true Hessian spectrum, without approximation, by back-propagating the second derivative. Extensive experiments on multiple networks showed that saddle-points are not the cause for the generalization gap of large batch size training, and the results consistently showed that large batch converges to points with noticeably higher Hessian spectrum. Furthermore, we showed that robust training allows one to favor flat areas, as points with a large Hessian spectrum show poor robustness to adversarial perturbation. We further studied this relationship, and provided empirical and theoretical proof that the inner loop for robust training is a saddle-free optimization problem *almost everywhere*. In Chapter 6, We presented PyHessian, a new scalable framework that enables fast computation of Hessian (i.e., second-order derivative) information for deep neural networks. PyHessian enabled fast computations of the top Hessian eigenvalues, the Hessian trace, and the full Hessian eigenvalue/spectral density, and it supported distributed-memory execution on cloud/supercomputer systems and is available as open source [111]. This general framework can be used to analyze neural network models, including the topology of the loss landscape (i.e., curvature information) to gain insight into the behavior of different models/optimizers. To illustrate this, we analyzed the effect of residual connections and Batch Normalization layers on the trainability of neural networks.

In Part III, we studied the application of second-order methods for neural network-related applications. In Chapter 7, we presented HAWQ-V3, a novel mixed-precision integer-only quantization framework using Hessian trace as the sensitivity metric. We proposed the integer-only inference where the entire computational graph is performed only with integer multiplication, addition, and bit shifting, without any floating point operations or even integer division. A novel hardware-aware mixed-precision quantization method is presented where the bit-precision is calculated by solving an integer linear programming problem that balances the trade-off between Hessian-based model perturbation and other constraints, e.g., memory footprint and latency. We also had direct hardware deployment for 4-bit uniform/mixed-precision quantization. To illustrate the benefit of our framework, we did extensive experiments on different neural networks. In Chapter 8, we presented a new family of trust region based adversarial attacks, with the goal of computing adversarial perturbations efficiently. We proposed several attacks based on variants of the trust region optimization method. We tested the proposed methods on different datasets using several different models.

Hessian-based analysis/computation/optimization is widely used in scientific computing. However, as machine learning problems grow both in data size and model size, the usage of the second-order for large-scale machine learning problems is limited due to the high computation cost of the Hessian matrix. In this dissertation, we presented some efficient second-order based algorithms and demonstrated both theoretical and empirical advantages of these algorithms. We hope this dissertation takes an important step towards more efforts and developments of second-order methods.

9.2 Future Directions

Beyond to what has been done in this dissertation, there are promising directions as future works.

• For second-order optimization algorithms, usually, an accurate Hessian approximation is needed to achieve a fast theoretical and/or practical convergence rate. This limits the application of traditional Newton-type of algorithms for recent large-scale machine learning problems, particularly for deep learning problems, since the model size and the dataset increase to the order of trillion [30]. Therefore, it is almost impossible to get an accurate gradient/Hessian approximation of such models and tasks. First-order based methods, such as SGD and Adam, work well for such large-scale problems since they are robust to mini-batch training. Designing and developing effective second-order methods for mini-batch training will be a promising direction.

• The storage cost and the per-iteration computational cost of second-order optimization methods are generally much higher than those of first-order methods. Those require both new algorithm design and system design. On the algorithm design side, more efficient Newton and quasi-Newton algorithms need to be proposed. For instance, a low-rank approximation of Hessian can save the space cost of storing the Hessian matrix as well as reduce the computational cost. However, how to find such an effective approximation is still an open problem. On the system design side, the current popular frameworks, e.g., TensorFlow [1] and PyTorch [178], only optimize first-order related computations. This leads Hessian-related computations to be slow. Designing an optimized system framework for Hessian-based computation would be super beneficial for this community.

• Current module components for deep learning models are mainly designed for achieving high testing accuracy. However, this causes the final architecture to be vulnerable to adversarial attacks (or even random perturbations) and/or to be hard to compress (e.g., quantization and pruning). One way to address this is to use the Hessian-aware module component design. As shown in [251, 111, 250], Hessian-related metric can be used to determine the robustness and the sensitivity of a trained model. However, how to directly incorporate Hessian-related metric in model designing is still under-explored. One interesting direction is that using Hessian-related metric as a regularization term (e.g., treat Hessian spectrum as a penalty term) in Neural Architecture Search [270].

Bibliography

[1] Martıén Abadi et al. “Tensorflow: A system for large-scale machine learning”. In: *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 2016, pp. 265–283.

[2] Naman Agarwal, Brian Bullins, and Elad Hazan. “Second order stochastic optimization in linear time”. In: *arXiv preprint arXiv:1602.03943* (2016).

[3] Naman Agarwal et al. “Finding Approximate Local Minima Faster than Gradient Descent”. In: *arXiv preprint arXiv:1611.01146* (2016).

[4] Amirali Aghazadeh et al. “BEAR: Sketching BFGS Algorithm for Ultra-High Dimensional Feature Selection in Sublinear Memory”. In: *arXiv preprint arXiv:2010.13829* (2020).

[5] Shun-Ichi Amari. “Natural gradient works efficiently in learning”. In: *Neural computation* 10.2 (1998), pp. 251–276.

[6] Martin Anthony and Peter L Bartlett. *Neural network learning: Theoretical foundations.* cambridge university press, 2009.

[7] Krste Asanovic and Nelson Morgan. *Experimental determination of precision requirements for back-propagation training of artificial neural networks.* International Computer Science Institute, 1991.

[8] U.M. Ascher and C. Greif. *A First Course on Numerical Methods.* Computational Science and Engineering. Siam, 2011. ISBN: 9780898719987.

[9] Anish Athalye, Nicholas Carlini, and David Wagner. “Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples”. In: *arXiv preprint arXiv:1802.00420* (2018).

[10] Haim Avron and Sivan Toledo. “Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix”. In: *Journal of the ACM (JACM)* 58.2 (2011), p. 8.

[11] Mart van Baalen et al. “Bayesian Bits: Unifying Quantization and Pruning”. In: *arXiv preprint arXiv:2005.07093* (2020).

[12] Zhaojun Bai, Gark Fahey, and Gene Golub. “Some large-scale matrix computation problems”. In: *Journal of Computational and Applied Mathematics* 74.1-2 (1996), pp. 71–89.

[13] Afonso S Bandeira, Katya Scheinberg, and Luiés N Vicente. “Convergence of trust-region methods based on probabilistic models”. In: *SIAM Journal on Optimization* 24.3 (2014), pp. 1238–1264.

[14] Ron Banner, Yury Nahshan, and Daniel Soudry. “Post training 4-bit quantization of convolutional networks for rapid-deployment”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 7950–7958.

[15] A. Beck. *First-Order Methods in Optimization*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, 2017. ISBN: 9781611974997.

[16] Sue Becker and Yann Le Cun. “Improving the convergence of back-propagation learning with second order methods”. In: *Proceedings of the 1988 connectionist models summer school*. 1988, pp. 29–37.

[17] Costas Bekas, Effrosyni Kokiopoulou, and Yousef Saad. “An estimator for the diagonal of a matrix”. In: *Applied numerical mathematics* 57.11-12 (2007), pp. 1214–1229.

[18] Albert S Berahas, Raghu Bollapragada, and Jorge Nocedal. “An Investigation of Newton-Sketch and Subsampled Newton Methods”. In: *arXiv preprint arXiv:1705.06211* (2017).

[19] Dimitri P. Bertsekas. *Nonlinear programming*. Athena scientific, 1999.

[20] Arjun Nitin Bhagoji, Daniel Cullina, and Prateek Mittal. “Dimensionality reduction as a defense against evasion attacks on machine learning classifiers”. In: *arXiv preprint arXiv:1704.02654* (2017).

[21] Jose Blanchet et al. “Convergence Rate Analysis of a Stochastic Trust-Region Method via Supermartingales”. In: *INFORMS journal on optimization* 1.2 (2019), pp. 92–119.

[22] Jose Blanchet et al. “Convergence rate analysis of a stochastic trust-region method via supermartingales”. In: *INFORMS Journal on Optimization* 1.2 (2019), pp. 92–119.

[23] Raghu Bollapragada, Richard H Byrd, and Jorge Nocedal. “Exact and inexact subsampled Newton methods for optimization”. In: *IMA Journal of Numerical Analysis* 39.2 (2018), pp. 545–578.

[24] Raghu Bollapragada, Richard H Byrd, and Jorge Nocedal. “Exact and inexact subsampled Newton methods for optimization”. In: *IMA Journal of Numerical Analysis* 39.2 (2019), pp. 545–578.

[25] Raghu Bollapragada et al. “A progressive batching L-BFGS method for machine learning”. In: *arXiv preprint arXiv:1802.05374* (2018).

[26] Léon Bottou, Frank E Curtis, and Jorge Nocedal. “Optimization methods for large-scale machine learning”. In: *SIAM Review* 60.2 (2018), pp. 223–311.

[27] Léon Bottou et al. "Comparison of classifier methods: a case study in handwritten digit recognition". In: *Computer Vision & Image Processing., Proceedings of the 12th IAPR International.* Vol. 2. 1994, pp. 77–82.

[28] Stephen Boyd and Lieven Vandenberghe. *Convex optimization.* Cambridge university press, 2004.

[29] Tom B Brown et al. "Adversarial patch". In: *arXiv preprint arXiv:1712.09665* (2017).

[30] Tom B Brown et al. "Language models are few-shot learners". In: *arXiv preprint arXiv:2005.14165* (2020).

[31] Richard H Byrd et al. "A limited memory algorithm for bound constrained optimization". In: *SIAM Journal on scientific computing* 16.5 (1995), pp. 1190–1208.

[32] Richard H Byrd et al. "On the use of stochastic Hessian information in optimization methods for machine learning". In: *SIAM Journal on Optimization* 21.3 (2011), pp. 977–995.

[33] Yaohui Cai et al. "ZeroQ: A novel zero shot quantization framework". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2020, pp. 13169–13178.

[34] Nicholas Carlini and David Wagner. "Adversarial examples are not easily detected: Bypassing ten detection methods". In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security.* ACM. 2017, pp. 3–14.

[35] Nicholas Carlini and David Wagner. "Towards evaluating the robustness of Neural Networks". In: *Security and Privacy (SP).* IEEE. 2017, pp. 39–57.

[36] Nicholas Carlini and David Wagner. "Towards evaluating the robustness of neural networks". In: *2017 IEEE Symposium on Security and Privacy (SP).* IEEE. 2017, pp. 39–57.

[37] Yair Carmon and John C Duchi. "Gradient Descent Efficiently Finds the Cubic-Regularized Non-Convex Newton Step". In: *arXiv preprint arXiv:1612.00547* (2016).

[38] Yair Carmon et al. "Accelerated methods for nonconvex optimization". In: *SIAM Journal on Optimization* 28.2 (2018), pp. 1751–1772.

[39] Coralia Cartis, Nicholas IM Gould, and Philippe L Toint. "Adaptive cubic regularisation methods for unconstrained optimization. Part I: motivation, convergence and numerical results". In: *Mathematical Programming* 127.2 (2011), pp. 245–295.

[40] Coralia Cartis, Nicholas IM Gould, and Philippe L Toint. "Adaptive cubic regularisation methods for unconstrained optimization. Part II: worst-case function-and derivative-evaluation complexity". In: *Mathematical programming* 130.2 (2011), pp. 295–319.

[41] Coralia Cartis, Nicholas IM Gould, and Philippe L Toint. "Complexity bounds for second-order optimality in unconstrained optimization". In: *Journal of Complexity* 28.1 (2012), pp. 93–108.

[42] Coralia Cartis, Nicholas IM Gould, and Philippe L Toint. "On the complexity of steepest descent, Newton's and regularized Newton's methods for nonconvex unconstrained optimization problems". In: *Siam Journal on Optimization* 20.6 (2010), pp. 2833–2852.

[43] Coralia Cartis, Nicholas IM Gould, and Philippe L Toint. *Optimal Newton-type methods for nonconvex smooth optimization problems*. Tech. rep. ERGO technical report 11-009, School of Mathematics, University of Edinburgh, 2011.

[44] Coralia Cartis and Katya Scheinberg. "Global convergence rate analysis of unconstrained optimization methods based on probabilistic models". In: *Mathematical Programming* 169.2 (2018), pp. 337–375.

[45] Chih-Chung Chang and Chih-Jen Lin. "LIBSVM: A library for support vector machines". In: *ACM Transactions on Intelligent Systems and Technology* 2 (3 2011). Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`, 27:1–27:27.

[46] Pratik Chaudhari et al. "Entropy-SGD: Biasing gradient descent into wide valleys". In: *arXiv preprint arXiv:1611.01838* (2016).

[47] Pratik Chaudhari et al. "Entropy-sgd: Biasing gradient descent into wide valleys". In: *Journal of Statistical Mechanics: Theory and Experiment* 2019.12 (2019), p. 124018.

[48] Chao Chen et al. "Fast Evaluation and Approximation of the Gauss-Newton Hessian Matrix for the Multilayer Perceptron". In: *arXiv preprint arXiv:1910.12184* (2019).

[49] Ruobing Chen, Matt Menickelly, and Katya Scheinberg. "Stochastic optimization using a trust-region method and random models". In: *arXiv preprint arXiv:1504.04231* (2015).

[50] Tianqi Chen et al. "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems". In: *arXiv preprint arXiv:1512.01274* (2015).

[51] Tianqi Chen et al. "TVM: An automated end-to-end optimizing compiler for deep learning". In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 2018, pp. 578–594.

[52] Xi Chen et al. "On adaptive cubic regularized Newton's methods for convex optimization via random sampling". In: *arXiv preprint arXiv:1802.05426* (2018).

[53] Sharan Chetlur et al. "cuDNN: Efficient primitives for deep learning". In: *arXiv preprint arXiv:1410.0759* (2014).

[54] Ting-Wu Chin et al. "One Weight Bitwidth to Rule Them All". In: *arXiv preprint arXiv:2008.09916* (2020).

[55] Jungwook Choi et al. "Pact: Parameterized clipping activation for quantized neural networks". In: *arXiv preprint arXiv:1805.06085* (2018).

[56] Anna Choromanska et al. "The loss surfaces of multilayer networks". In: *Artificial Intelligence and Statistics*. 2015, pp. 192–204.

[57] Andrew R Conn, Nicholas IM Gould, and Philippe L Toint. *Trust region methods*. Series on Optimization. SIAM, 2000.

[58] Andrew R Conn, Katya Scheinberg, and Luiés N Vicente. "Global convergence of general derivative-free trust-region algorithms to first-and second-order critical points". In: *SIAM Journal on Optimization* 20.1 (2009), pp. 387–415.

[59] F. E. Curtis et al. "Trust-region Newton-CG with strong second- order complexity guarantees for nonconvex optimization". In: *SIAM Journal on Optimization* (2021). to appear. URL: https://arxiv.org/abs/1912.04365.

[60] Frank E Curtis, Daniel P Robinson, and Mohammadreza Samadi. "A Trust Region Algorithm with a Worst-Case Iteration Complexity of $\mathcal{O}(\epsilon^{-3/2})$for Nonconvex Optimization". In: *COR@ L Technical Report 14T-009, Lehigh University,, Bethlehem, PA, USA* (2014).

[61] Zihang Dai et al. "Transformer-XL: Attentive Language Models beyond a Fixed-Length Context". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019, pp. 2978–2988.

[62] Yann N Dauphin et al. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". In: *Advances in neural information processing systems*. 2014, pp. 2933–2941.

[63] Olivier Delalleau and Yoshua Bengio. "Shallow vs. deep sum-product networks". In: *Advances in Neural Information Processing Systems*. 2011, pp. 666–674.

[64] Ron S Dembo, Stanley C Eisenstat, and Trond Steihaug. "Inexact Newton methods". In: *SIAM Journal on Numerical analysis* 19.2 (1982), pp. 400–408.

[65] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. Ieee. 2009, pp. 248–255.

[66] Guillaume Desjardins et al. "Natural neural networks". In: *Advances in Neural Information Processing Systems*. 2015, pp. 2071–2079.

[67] Laurent Dinh et al. "Sharp minima can generalize for deep nets". In: *arXiv preprint arXiv:1703.04933* (2017).

[68] Yinpeng Dong et al. "Learning accurate low-bit deep neural networks with stochastic quantization". In: *British Machine Vision Conference* (2017).

[69] Zhen Dong et al. "HAWQ-V2: Hessian Aware trace-Weighted Quantization of Neural Networks". In: *NuerIPS'19 workshop on Beyond First-Order Optimization Methods in Machine Learning.* (2019).

[70] Zhen Dong et al. "Hawq: Hessian aware quantization of neural networks with mixed-precision". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 293–302.

[71] Petros Drineas, Ravi Kannan, and Michael W Mahoney. "Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix". In: *SIAM Journal on computing* 36.1 (2006), pp. 158–183.

[72] Petros Drineas and Michael W Mahoney. "Lectures on Randomized Numerical Linear Algebra". In: *The Mathematics of Data*. IAS/Park City Mathematics Series. AMS/IAS/SIAM, 2018, pp. 1–48.

[73] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[74] Marat Dukhan. *NNPACK*. 2016.

[75] Stanley C Eisenstat and Homer F Walker. "Choosing the forcing terms in an inexact Newton method". In: *SIAM Journal on Scientific Computing* 17.1 (1996), pp. 16–32.

[76] Ahmed T Elthakeb et al. "Gradient-Based Deep Quantization of Neural Networks through Sinusoidal Adaptive Regularization". In: *arXiv preprint arXiv:2003.00146* (2020).

[77] Murat A Erdogdu and Andrea Montanari. "Convergence rates of sub-sampled Newton methods". In: *arXiv preprint arXiv:1508.02810* (2015).

[78] Mahyar Fazlyab et al. "Efficient and accurate estimation of Lipschitz constants for deep neural networks". In: *Advances in Neural Information Processing Systems*. 2019, pp. 11427–11438.

[79] Reuben Feinman et al. "Detecting adversarial samples from artifacts". In: *arXiv preprint arXiv:1703.00410* (2017).

[80] Kenji Fukumizu and Shun-ichi Amari. "Local minima and plateaus in hierarchical structures of multilayer perceptrons". In: *Neural Networks* 13.3 (2000), pp. 317–327.

[81] Rong Ge et al. "Escaping from saddle points—online stochastic gradient for tensor decomposition". In: *Conference on Learning Theory*. 2015, pp. 797–842.

[82] Amir Gholami et al. "Integrated Model, Batch and Domain Parallelism in Training Neural Networks". In: *ACM Symposium on Parallelism in Algorithms and Architectures(SPAA'18)* (2018). [PDF].

[83] Amir Gholami et al. "SqueezeNext: Hardware-Aware Neural Network Design". In: *Workshop paper in CVPR* (2018).

[84] Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. "An investigation into neural net optimization via Hessian eigenvalue density". In: *arXiv preprint arXiv:1901.10159* (2019).

[85] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 315–323.

[86] Gene H Golub and Gérard Meurant. *Matrices, moments and quadrature with applications*. Princeton University Press, 2009.

[87] Gene H Golub and John H Welsch. “Calculation of Gauss quadrature rules”. In: *Mathematics of computation* 23.106 (1969), pp. 221–230.

[88] Zhitao Gong, Wenlu Wang, and Wei-Shinn Ku. “Adversarial and clean data are not twins”. In: *arXiv preprint arXiv:1704.04960* (2017).

[89] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and harnessing adversarial examples (2014)”. In: *arXiv preprint arXiv:1412.6572* (2014).

[90] Priya Goyal et al. “Accurate, large minibatch SGD: training imagenet in 1 hour”. In: *arXiv preprint arXiv:1706.02677* (2017).

[91] Serge Gratton, Amos S Lawless, and Nancy K Nichols. “Approximate Gauss–Newton methods for nonlinear least squares problems”. In: *SIAM Journal on Optimization* 18.1 (2007), pp. 106–132.

[92] Serge Gratton et al. “Complexity and global rates of trust-region methods based on probabilistic models”. In: *IMA Journal of Numerical Analysis* 38.3 (2017), pp. 1579–1597.

[93] Serge Gratton et al. “Complexity and global rates of trust-region methods based on probabilistic models”. In: *IMA Journal of Numerical Analysis* 38.3 (2018), pp. 1579–1597.

[94] Robert M. Gray and David L. Neuhoff. “Quantization”. In: *IEEE transactions on information theory* 44.6 (1998), pp. 2325–2383.

[95] Andreas Griewank. “Some bounds on the complexity of gradients, Jacobians, and Hessians”. In: *Complexity in numerical optimization*. World Scientific, 1993, pp. 128–162.

[96] Kathrin Grosse et al. “On the (statistical) detection of adversarial examples”. In: *arXiv preprint arXiv:1702.06280* (2017).

[97] Antonio Gulli and Sujit Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[98] Vineet Gupta, Tomer Koren, and Yoram Singer. “Shampoo: Preconditioned stochastic tensor optimization”. In: *arXiv preprint arXiv:1802.09568* (2018).

[99] Vipul Gupta et al. “Oversketched newton: Fast convex optimization for serverless systems”. In: *arXiv preprint arXiv:1903.08857* (2019).

[100] Song Han and B Dally. “Efficient methods and hardware for deep learning”. In: *University Lecture* (2017).

[101] Song Han, Huizi Mao, and William J Dally. “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding”. In: *International Conference on Learning Representations* (2016).

[102] Song Han et al. “Learning both weights and connections for efficient neural network”. In: *Advances in neural information processing systems*. 2015, pp. 1135–1143.

[103] Babak Hassibi and David G Stork. “Second order derivatives for network pruning: Optimal brain surgeon”. In: *Advances in neural information processing systems*. 1993, pp. 164–171.

[104] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[105] Xi He et al. “Large scale distributed Hessian-free optimization for deep neural network”. In: *arXiv preprint arXiv:1606.00511* (2016).

[106] Christopher J Hillar and Lek-Heng Lim. “Most tensor problems are NP-hard”. In: *Journal of the ACM (JACM)* 60.6 (2013), p. 45.

[107] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the knowledge in a neural network”. In: *Workshop paper in NIPS* (2014).

[108] Sepp Hochreiter and Jürgen Schmidhuber. “Flat minima”. In: *Neural Computation* 9.1 (1997), pp. 1–42.

[109] Andrew Howard et al. “Searching for MobileNetV3”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 1314–1324.

[110] *https://github.com/amirgholami/ADAHESSIAN.git*. May 2020.

[111] *https://github.com/amirgholami/PyHessian.git*. Sept. 2019.

[112] *https://github.com/Zhen-Dong/HAWQ.git*. Oct. 2020.

[113] Gao Huang et al. “Densely connected convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[114] Itay Hubara et al. “Binarized neural networks”. In: *Advances in neural information processing systems*. 2016, pp. 4107–4115.

[115] Itay Hubara et al. “Improving Post Training Neural Quantization: Layer-wise Calibration and Integer Programming”. In: *arXiv preprint arXiv:2006.10518* (2020).

[116] Forrest N Iandola et al. “SqueezeBERT: What can computer vision teach NLP about efficient neural networks?” In: *arXiv preprint arXiv:2006.11316* (2020).

[117] Forrest N Iandola et al. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and¡ 0.5 MB model size”. In: *arXiv preprint arXiv:1602.07360* (2016).

[118] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *International Conference on Machine Learning*. 2015, pp. 448–456.

[119] Benoit Jacob et al. *gemmlowp: a small self-contained low-precision GEMM library.(2017)*. 2017.

[120] Benoit Jacob et al. “Quantization and training of neural networks for efficient integer-arithmetic-only inference”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2704–2713.

[121] Animesh Jain et al. “Efficient execution of quantized deep learning models: A compiler approach”. In: *arXiv preprint arXiv:2006.10226* (2020).

[122] Yangqing Jia et al. “Caffe: Convolutional architecture for fast feature embedding”. In: *Proceedings of the 22nd ACM international conference on Multimedia*. 2014, pp. 675–678.

[123] Chi Jin et al. “How to escape saddle points efficiently”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 1724–1732.

[124] Nitish Shirish Keskar et al. “On large-batch training for deep learning: Generalization gap and sharp minima”. In: *arXiv preprint arXiv:1609.04836* (2016).

[125] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *International Conference on Learning Representations* (2015).

[126] Ryan Kiros. “Training neural networks with stochastic Hessian-free optimization”. In: *arXiv preprint arXiv:1301.3641* (2013).

[127] Jonas Moritz Kohler and Aurelien Lucchi. “Sub-sampled Cubic Regularization for Non-convex Optimization”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 1895–1904.

[128] Raghuraman Krishnamoorthi. “Quantizing deep convolutional networks for efficient inference: A whitepaper”. In: *arXiv preprint arXiv:1806.08342* (2018).

[129] Alex Krizhevsky and Geoffrey Hinton. *Learning multiple layers of features from tiny images*. Tech. rep. Citeseer, 2009.

[130] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. “Adversarial examples in the physical world”. In: *arXiv preprint arXiv:1607.02533* (2016).

[131] Sudhir Kylasa et al. “GPU Accelerated Sub-Sampled Newton’s Method for Convex Classification Problems”. In: *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM. 2019, pp. 702–710.

[132] G. Lan. *First-order and Stochastic Optimization Methods for Machine Learning*. Springer Series in the Data Sciences. Springer International Publishing, 2020. ISBN: 9783030395674.

[133] Jeffrey Larson and Stephen C Billups. “Stochastic derivative-free optimization using a trust region framework”. In: *Computational Optimization and Applications* 64.3 (2016), pp. 619–645.

[134] Nicolas Le Roux and Yoshua Bengio. “Deep belief networks are compact universal approximators”. In: *Neural computation* 22.8 (2010), pp. 2192–2207.

[135] Yann LeCun, John S Denker, and Sara A Solla. “Optimal brain damage”. In: *Advances in neural information processing systems*. 1990, pp. 598–605.

[136] Yann LeCun, Ido Kanter, and Sara A Solla. “Second order properties of error surfaces: Learning time and generalization”. In: *Advances in neural information processing systems*. 1991, pp. 918–924.

[137] Yann A LeCun et al. “Efficient backprop”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[138] Jason D Lee et al. “First-order methods almost always avoid saddle points”. In: *arXiv preprint arXiv:1710.07406* (2017).

[139] Kfir Y Levy. “The power of normalization: Faster evasion of saddle points”. In: *arXiv preprint arXiv:1611.04831* (2016).

[140] Hao Li et al. “Pruning filters for efficient convnets”. In: *arXiv preprint arXiv:1608.08710* (2016).

[141] Hao Li et al. “Visualizing the loss landscape of neural nets”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 6389–6399.

[142] Lin Lin, Yousef Saad, and Chao Yang. “Approximating spectral densities of large matrices”. In: *SIAM review* 58.1 (2016), pp. 34–65.

[143] Z. Lin, H. Li, and C. Fang. *Accelerated Optimization for Machine Learning: First-Order Algorithms*. Springer Singapore, 2020.

[144] Congcong Liu and Huaming Wu. “Channel pruning based on mean gradient for accelerating convolutional neural networks”. In: *Signal Processing* 156 (2019), pp. 84–91.

[145] Dong C Liu and Jorge Nocedal. “On the limited memory BFGS method for large scale optimization”. In: *Mathematical programming* 45.1-3 (1989), pp. 503–528.

[146] Ilya Loshchilov and Frank Hutter. “Decoupled Weight Decay Regularization”. In: *International Conference on Learning Representations*. 2019.

[147] Ilya Loshchilov and Frank Hutter. “Sgdr: Stochastic gradient descent with warm restarts”. In: *International Conference on Learning Representations*. 2017.

[148] Linjian Ma et al. “Inefficiency of K-FAC for Large Batch Size Training”. In: *Thirty-Fourth AAAI Conference on Artificial Intelligence*. 2020.

[149] Xindian Ma et al. “A tensorized transformer for language modeling”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 2229–2239.

[150] Aleksander Madry et al. “Towards deep learning models resistant to adversarial attacks”. In: *International Conference on Learning Representations* (2018). URL: https://openreview.net/forum?id=rJzIBfZAb.

[151] Michael W Mahoney. *Randomized algorithms for matrices and data*. Foundations and Trends in Machine Learning. Boston: NOW Publishers, 2011.

[152] Huizi Mao et al. “Exploring the regularity of sparse structure in convolutional neural networks”. In: *Workshop paper in CVPR* (2017).

[153] James Martens. “Deep learning via Hessian-free optimization”. In: *International Conference on Machine Learning (ICML).* 2010.

[154] James Martens and Roger Grosse. “Optimizing neural networks with Kronecker-factored approximate curvature”. In: *International conference on machine learning.* 2015, pp. 2408–2417.

[155] James Martens and Ilya Sutskever. “Training deep and recurrent networks with Hessian-free optimization”. In: *Neural networks: Tricks of the trade* (2012), pp. 479–535.

[156] H Brendan McMahan and Matthew Streeter. “Adaptive bound optimization for online convex optimization”. In: *arXiv preprint arXiv:1002.4908* (2010).

[157] Stephen Merity et al. “Pointer sentinel mixture models”. In: *International Conference on Learning Representations.* 2017.

[158] Jan Hendrik Metzen et al. “On detecting adversarial perturbations”. In: *arXiv preprint arXiv:1702.04267* (2017).

[159] Tomá**v**s Mikolov et al. “Empirical evaluation and combination of advanced language modeling techniques”. In: *Twelfth Annual Conference of the International Speech Communication Association.* 2011.

[160] Asit Mishra and Debbie Marr. “Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy”. In: *arXiv preprint arXiv:1711.05852* (2017).

[161] Pavlo Molchanov et al. “Pruning convolutional neural networks for resource efficient inference”. In: *arXiv preprint arXiv:1611.06440* (2016).

[162] Guido F Montufar et al. “On the number of linear regions of deep neural networks”. In: *Advances in neural information processing systems.* 2014, pp. 2924–2932.

[163] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. “Deepfool: a simple and accurate method to fool deep neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2016, pp. 2574–2582.

[164] Seyed-Mohsen Moosavi-Dezfooli et al. “Universal Adversarial Perturbations”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE. 2017, pp. 86–94.

[165] Katta G Murty and Santosh N Kabadi. “Some NP-complete problems in quadratic and nonlinear programming”. In: *Mathematical Programming* 39.2 (1987), pp. 117–129.

[166] Maxim Naumov et al. “Deep learning recommendation model for personalization and recommendation systems”. In: *arXiv preprint arXiv:1906.00091* (2019).

[167] Maxim Naumov et al. "On periodic functions as regularizers for quantization of neural networks". In: *arXiv preprint arXiv:1811.09862* (2018).

[168] Yurii Nesterov. "A method for unconstrained convex minimization problem with the rate of convergence O (1/kˆ 2)". In: *Doklady an ussr*. Vol. 269. 1983, pp. 543–547.

[169] Yurii Nesterov and Boris T Polyak. "Cubic regularization of Newton method and its global performance". In: *Mathematical Programming* 108.1 (2006), pp. 177–205.

[170] Jorge Nocedal. "Updating quasi-Newton matrices with limited storage". In: *Mathematics of computation* 35.151 (1980), pp. 773–782.

[171] Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer Science & Business Media, 2006.

[172] NVIDIA. *Cutlass Library*. 2020. URL: https://github.com/NVIDIA/cutlass.

[173] Myle Ott et al. "fairseq: A Fast, Extensible Toolkit for Sequence Modeling". In: *Proceedings of NAACL-HLT 2019: Demonstrations*. 2019.

[174] Myle Ott et al. "Scaling Neural Machine Translation". In: *Proceedings of the Third Conference on Machine Translation: Research Papers*. 2018, pp. 1–9.

[175] Nicolas Papernot et al. "Distillation as a defense to adversarial perturbations against deep neural networks". In: *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2016, pp. 582–597.

[176] Kishore Papineni et al. "BLEU: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2002, pp. 311–318.

[177] Eunhyeok Park, Sungjoo Yoo, and Peter Vajda. "Value-aware quantization for training and inference of neural networks". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 580–595.

[178] Adam Paszke et al. "Automatic differentiation in PyTorch". In: (2017).

[179] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[180] Barak A Pearlmutter. "Fast exact multiplication by the Hessian". In: *Neural computation* 6.1 (1994), pp. 147–160.

[181] Jeffrey Pennington and Yasaman Bahri. "Geometry of neural network loss surfaces via random matrix theory". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 2798–2806.

[182] Jason Phang, Thibault Févry, and Samuel R Bowman. "Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks". In: *arXiv preprint arXiv:1811.01088* (2018).

[183] Mert Pilanci and Martin J Wainwright. “Newton sketch: A near linear-time optimization algorithm with linear-quadratic convergence”. In: *SIAM Journal on Optimization* 27.1 (2017), pp. 205–245.

[184] Mert Pilanci and Martin J. Wainwright. “Newton Sketch: a linear-time optimization algorithm with linear-quadratic convergence”. In: *arXiv preprint arXiv:1505.02250* (2015).

[185] Antonio Polino, Razvan Pascanu, and Dan Alistarh. “Model compression via distillation and quantization”. In: *arXiv preprint arXiv:1802.05668* (2018).

[186] R Gerhard Pratt, Changsoo Shin, and GJ Hick. “Gauss–Newton and full Newton methods in frequency–space seismic waveform inversion”. In: *Geophysical Journal International* 133.2 (1998), pp. 341–362.

[187] *PyTorchCV Library*. 2020. URL: https://pypi.org/project/pytorchcv/.

[188] Prajit Ramachandran, Barret Zoph, and Quoc V Le. “Searching for activation functions”. In: *arXiv preprint arXiv:1710.05941* (2018).

[189] Prajit Ramachandran, Barret Zoph, and Quoc V Le. “Swish: a self-gated activation function”. In: *arXiv preprint arXiv:1710.05941* (2017).

[190] Mohammad Rastegari et al. “Xnor-net: Imagenet classification using binary convolutional neural networks”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 525–542.

[191] Jonas Rauber, Wieland Brendel, and Matthias Bethge. “Foolbox v0. 8.0: A python toolbox to benchmark the robustness of machine learning models”. In: *arXiv preprint arXiv:1707.04131* (2017).

[192] Sashank J Reddi et al. “A generic approach for escaping saddle points”. In: *arXiv preprint arXiv:1709.01434* (2017).

[193] Jeffrey Regier, Michael I Jordan, and Jon McAuliffe. “Fast Black-box Variational Inference through Stochastic Trust-Region Optimization”. In: *arXiv preprint arXiv:1706.02375* (2017).

[194] Jorma Rissanen. “Modeling by shortest data description”. In: *Automatica* 14.5 (1978), pp. 465–471.

[195] Herbert Robbins and Sutton Monro. “A stochastic approximation method”. In: *The annals of mathematical statistics* (1951), pp. 400–407.

[196] Fred Roosta and Michael W Mahoney. “Sub-sampled Newton methods”. In: *Mathematical Programming* 174.1-2 (2019), pp. 293–326.

[197] Farbod Roosta-Khorasani and Michael W Mahoney. “Sub-sampled Newton methods I: globally convergent algorithms”. In: *arXiv preprint arXiv:1601.04737* (2016).

[198] J.S. Roy and S.A. Mitchell. “PuLP is an LP modeler written in Python”. In: (2020). URL: https://github.com/coin-or/pulp.

[199] Clément W Royer, Michael O'Neill, and Stephen J Wright. "A Newton-CG Algorithm with Complexity Guarantees for Smooth Unconstrained Optimization". In: *arXiv preprint arXiv:1803.02924* (2018).

[200] Clément W Royer and Stephen J Wright. "Complexity analysis of second-order line-search algorithms for smooth nonconvex optimization". In: *SIAM Journal on Optimization* 28.2 (2018), pp. 1448–1477.

[201] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[202] Levent Sagun, Leon Bottou, and Yann LeCun. "Eigenvalues of the Hessian in deep learning: Singularity and beyond". In: *arXiv preprint arXiv:1611.07476* (2016).

[203] Levent Sagun et al. "Empirical analysis of the hessian of over-parametrized neural networks". In: *arXiv preprint arXiv:1706.04454* (2017).

[204] Mark Sandler et al. "Mobilenetv2: Inverted residuals and linear bottlenecks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2018, pp. 4510–4520.

[205] Swami Sankaranarayanan et al. "Regularizing deep networks using efficient layerwise adversarial training". In: *arXiv preprint arXiv:1705.07819* (2017).

[206] Shibani Santurkar et al. "How does batch normalization help optimization?" In: *Advances in Neural Information Processing Systems.* 2018, pp. 2483–2493.

[207] Andrew M Saxe, James L McClelland, and Surya Ganguli. "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks". In: *arXiv preprint arXiv:1312.6120* (2013).

[208] Tom Schaul, Sixin Zhang, and Yann LeCun. "No more pesky learning rates". In: *International Conference on Machine Learning.* 2013, pp. 343–351.

[209] Nicol N Schraudolph, Jin Yu, and Simon Günter. "A stochastic quasi-Newton method for online convex optimization". In: *Artificial intelligence and statistics.* 2007, pp. 436–443.

[210] Frank Seide and Amit Agarwal. "CNTK: Microsoft's open-source deep-learning toolkit". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* 2016, pp. 2135–2135.

[211] Uri Shaham, Yutaro Yamada, and Sahand Negahban. "Understanding adversarial training: Increasing local stability of neural nets through robust optimization". In: *arXiv preprint arXiv:1511.05432* (2015).

[212] Sara Shashaani, Fatemeh S Hashemi, and Raghu Pasupathy. "ASTRO-DF: A class of adaptive sampling trust-region algorithms for derivative-free stochastic optimization". In: *SIAM Journal on Optimization* 28.4 (2018), pp. 3145–3176.

[213] Noam Shazeer and Mitchell Stern. “Adafactor: Adaptive Learning Rates with Sublinear Memory Cost”. In: *International Conference on Machine Learning*. 2018, pp. 4596–4604.

[214] Sheng Shen et al. “Q-BERT: Hessian based ultra low precision quantization of bert”. In: *Thirty-Fourth AAAI Conference on Artificial Intelligence*. 2020.

[215] Sheng Shen et al. “Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT.” In: *AAAI*. 2020, pp. 8815–8821.

[216] Jonathan Richard Shewchuk et al. *An introduction to the conjugate gradient method without the agonizing pain*. 1994.

[217] K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *International Conference on Learning Representations*. 2015.

[218] Bharat Singh et al. “Layer-specific adaptive learning rates for deep networks”. In: *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2015, pp. 364–368.

[219] Samuel L Smith, Pieter-Jan Kindermans, and Quoc V Le. “Don’t decay the learning rate, increase the batch size”. In: *arXiv preprint arXiv:1711.00489* (2017).

[220] Samuel L Smith and Quoc V Le. “A Bayesian perspective on generalization and stochastic gradient descent”. In: *Second workshop on Bayesian Deep Learning (NIPS 2017)* (2017).

[221] Trond Steihaug. “The conjugate gradient method and trust regions in large scale optimization”. In: *SIAM Journal on Numerical Analysis* 20.3 (1983), pp. 626–637.

[222] Xu Sun et al. “meprop: Sparsified back propagation for accelerated deep learning with reduced overfitting”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3299–3308.

[223] Ilya Sutskever et al. “On the importance of initialization and momentum in deep learning”. In: *International conference on machine learning*. 2013, pp. 1139–1147.

[224] Grzegorz Swirszcz, Wojciech Marian Czarnecki, and Razvan Pascanu. “Local minima in training of deep networks”. In: *arXiv preprint arXiv:1611.06310* (2016).

[225] Christian Szegedy et al. “Intriguing properties of neural networks”. In: *arXiv preprint arXiv:1312.6199* (2013).

[226] Mingxing Tan and Quoc V Le. “Efficientnet: Rethinking model scaling for convolutional neural networks”. In: *arXiv preprint arXiv:1905.11946* (2019).

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

[228] Florian Tramèr et al. “Ensemble adversarial training: Attacks and defenses”. In: *arXiv preprint arXiv:1705.07204* (2017).

[229] Nilesh Tripuraneni et al. "Stochastic cubic regularization for fast nonconvex optimization". In: *Advances in neural information processing systems*. 2018, pp. 2899–2908.

[230] Shashanka Ubaru, Jie Chen, and Yousef Saad. "Fast Estimation of tr(f(A)) via Stochastic Lanczos Quadrature". In: *SIAM Journal on Matrix Analysis and Applications* 38.4 (2017), pp. 1075–1099.

[231] Nicolas Vasilache et al. "Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions". In: *arXiv preprint arXiv:1802.04730* (2018).

[232] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[233] Oriol Vinyals and Daniel Povey. "Krylov Subspace Descent for Deep Learning". In: *AISTATS*. 2012, pp. 1261–1268.

[234] Alex Wang et al. "Glue: A multi-task benchmark and analysis platform for natural language understanding". In: *arXiv preprint arXiv:1804.07461* (2018).

[235] Kuan Wang et al. "HAQ: Hardware-Aware Automated Quantization". In: *In Proceedings of the IEEE conference on computer vision and pattern recognition* (2019).

[236] Qiang Wang et al. "Learning Deep Transformer Models for Machine Translation". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019, pp. 1810–1822.

[237] Ruoxi Wang et al. "Deep & cross network for ad click predictions". In: *Proceedings of the ADKDD'17*. 2017, pp. 1–7.

[238] Shusen Wang et al. "GIANT: Globally improved approximate Newton method for distributed optimization". In: *Advances in Neural Information Processing Systems*. 2018, pp. 2332–2342.

[239] Simon Wiesler, Jinyu Li, and Jian Xue. "Investigations on Hessian-free optimization for cross-entropy training of deep neural networks". In: *INTERSPEECH*. 2013, pp. 3317–3321.

[240] Stephen J. Wright and Benjamin Recht. *Optimization for Data Analysis*. Cambridge University Press, 2021.

[241] Bichen Wu et al. "FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 10734–10742.

[242] Bichen Wu et al. "Mixed Precision Quantization of ConvNets via Differentiable Neural Architecture Search". In: *arXiv preprint arXiv:1812.00090* (2018).

[243] Peng Xu, Fred Roosta, and Michael W. Mahoney. "Second-Order Optimization for Non-Convex Machine Learning: An Empirical Study". In: *Proceedings of the 2020 SIAM International Conference on Data Mining*. to appear. SIAM. 2020.

[244] Peng Xu, Farbod Roosta-Khorasan, and Michael W Mahoney. "Second-order optimization for non-convex machine learning: An empirical study". In: *arXiv preprint arXiv:1708.07827* (2017).

[245] Peng Xu, Farbod Roosta-Khorasani, and Michael W Mahoney. "Newton-type methods for non-convex optimization under inexact Hessian information". In: *arXiv preprint arXiv:1708.07164* (2017).

[246] Peng Xu, Farbod Roosta-Khorasani, and Michael W. Mahoney. "Newton-Type Methods for Non-Convex Optimization Under Inexact Hessian Information". In: *arXiv preprint arXiv:1708.07164* (2017).

[247] Peng Xu et al. "Sub-sampled Newton methods with non-uniform sampling". In: *Advances in Neural Information Processing Systems*. 2016, pp. 3000–3008.

[248] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. "Designing energy-efficient convolutional neural networks using energy-aware pruning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5687–5695.

[249] Zhewei Yao et al. "ADAHESSIAN: An adaptive second order optimizer for machine learning". In: *arXiv preprint arXiv:2006.00719* (2020).

[250] Zhewei Yao et al. "HAWQV3: Dyadic Neural Network Quantization". In: *arXiv preprint arXiv:2011.10680* (2020).

[251] Zhewei Yao et al. "Hessian-based Analysis of Large Batch Training and Robustness to Adversaries". In: *Advances in Neural Information Processing Systems* (2018).

[252] Zhewei Yao et al. "Inexact non-convex Newton-type methods". In: *arXiv preprint arXiv:1802.06925* (2018).

[253] Zhewei Yao et al. "Large batch size training of neural networks with adversarial training and second-order information". In: *arXiv preprint arXiv:1810.01021* (2018).

[254] Zhewei Yao et al. "PyHessian: Neural Networks Through the Lens of the Hessian". In: *arXiv preprint arXiv:1912.07145* (2019).

[255] Zhewei Yao et al. "Trust region based adversarial attack on neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11350–11359.

[256] Hongxu Yin et al. "Dreaming to distill: Data-free knowledge transfer via DeepInversion". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 8715–8724.

[257] Yang You, Igor Gitman, and Boris Ginsburg. "Scaling SGD batch size to 32K for ImageNet training". In: *arXiv preprint arXiv:1708.03888* (2017).

[258] Yang You et al. "Large batch optimization for deep learning: Training bert in 76 minutes". In: *International Conference on Learning Representations*. 2019.

[259] Sergey Zagoruyko and Nikos Komodakis. “Wide residual networks”. In: *arXiv preprint arXiv:1605.07146* (2016).

[260] Matthew D Zeiler. “Adadelta: an adaptive learning rate method”. In: *arXiv preprint arXiv:1212.5701* (2012).

[261] Chiyuan Zhang et al. “Understanding deep learning requires rethinking generalization”. In: *arXiv preprint arXiv:1611.03530* (2016).

[262] Dongqing Zhang et al. “LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks”. In: *The European Conference on Computer Vision (ECCV)*. 2018.

[263] Jingzhao Zhang et al. “Why ADAM Beats SGD for Attention Models”. In: *arXiv preprint arXiv:1912.03194* (2019).

[264] Ruizhi Zhang et al. “Robustness and Tractability for Non-convex M-estimators”. In: *arXiv preprint arXiv:1906.02272* (2019).

[265] Sixin Zhang, Anna E Choromanska, and Yann LeCun. “Deep learning with elastic averaging SGD”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 685–693.

[266] X-Y Zhang et al. “Sobol sensitivity analysis: a tool to guide the development and evaluation of systems pharmacology models”. In: *CPT: pharmacometrics & systems pharmacology* 4.2 (2015), pp. 69–79.

[267] Aojun Zhou et al. “Incremental network quantization: Towards lossless cnns with low-precision weights”. In: *International Conference on Learning Representations* (2017).

[268] Shuchang Zhou et al. “Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients”. In: *arXiv preprint arXiv:1606.06160* (2016).

[269] Yiren Zhou et al. “Adaptive quantization for deep neural network”. In: *arXiv preprint arXiv:1712.01048* (2017).

[270] Barret Zoph and Quoc V Le. “Neural architecture search with reinforcement learning”. In: *arXiv preprint arXiv:1611.01578* (2016).

Part IV

Appendix

Appendix A

Appendix for Chapter 2

A.1 Proof of Theorem 1

We first give the following two standard lemmas regarding Cauchy and Eigen points [57], which establish Condition 2.

Lemma 2 (Cauchy Points)**.** *[57, Corollary 6.3.2] Suppose that* $\boldsymbol{s}_t^C = \arg\min_{\|\alpha \mathbf{g}_k\| \le \Delta_t} m_t(-\alpha \mathbf{g}_t)$*. We have*

$$-m_t(\boldsymbol{s}_t^C) \ge \frac{1}{2}\|\mathbf{g}_t\| \min\left\{\frac{\|\mathbf{g}_t\|}{1+\|\mathbf{H}_t\|}, \Delta_t\right\}. \tag{A.1}$$

Lemma 3 (Eigen points)**.** *[57, Theorem 6.6.1] When* $\lambda_{\min}(\mathbf{H}_t)$ *is negative, suppose* $\boldsymbol{u}_t$ *satisfies*

$$\langle \mathbf{g}_t, \boldsymbol{u}_t \rangle \le 0, \qquad \text{and} \qquad \langle \boldsymbol{u}_t, \mathbf{H}_t \boldsymbol{u}_t \rangle \le -\nu |\lambda_{\min}(\mathbf{H}_t)| \|\boldsymbol{u}_t\|^2, \tag{A.2}$$

and let $\boldsymbol{s}_t^E = \arg\min_{\|\boldsymbol{s}_t\| \le \Delta_t} m_t(\alpha \boldsymbol{u}_t)$*. We have*

$$-m_t(\boldsymbol{s}_t^E) \ge \frac{\nu}{2} |\lambda_{\min}(\mathbf{H}_t)| \Delta_t^2. \tag{A.3}$$

The above two lemmas show the descent that can be obtained by Cauchy and Eigen Points. The following lemma bounds the difference between the actual decrement, i.e., $F(\mathbf{x}_t + \mathbf{s}_t) - F(\mathbf{x}_t)$, and the one predicted by $m(\mathbf{s}_t)$.

Lemma 4. *Under Assumptions 1 and 2, we have*

$$F(\boldsymbol{x}_t + \boldsymbol{s}_t) - F(\boldsymbol{x}_t) - m_t(\boldsymbol{s}_t) \le \begin{cases} \delta_g \Delta_t + \frac{1}{2}\delta_H \Delta_t^2 + \frac{1}{2} L_F \Delta_t^3, & \|\mathbf{g}_t\| \ge \epsilon_g, \\ \\ \langle \boldsymbol{s}_t, \nabla F(\boldsymbol{x}_t) \rangle + \frac{1}{2}\delta_H \Delta_t^2 + \frac{1}{2} L_F \Delta_t^3, & \|\mathbf{g}_t\| < \epsilon_g. \end{cases} \tag{A.4}$$

Proof. When $\|\mathbf{g}_t\| \geq \epsilon_g$, using taylor expansion of $F(\mathbf{x}_t)$ at point $\mathbf{x}_t$,

$$\begin{aligned}
F(\mathbf{x}_t + \mathbf{s}_t) - F(\mathbf{x}_t) - m_t(\mathbf{s}_t) &= \langle \mathbf{s}_t, \nabla F(\mathbf{x}_t) - \mathbf{g}_t \rangle + \frac{1}{2}\langle \mathbf{s}_t, (\nabla^2 F(\mathbf{x}_t + \tau \mathbf{s}_t) - \mathbf{H}_t)\mathbf{s}_t \rangle \\
&\leq \langle \mathbf{s}_t, \nabla F(\mathbf{x}_t) - \mathbf{g}_t \rangle + |\frac{1}{2}\langle \mathbf{s}_t, (\mathbf{H}_t - \nabla^2 F(\mathbf{x}_t + \tau \mathbf{s}_t))\mathbf{s}_t \rangle| \\
&\leq \langle \mathbf{s}_t, \nabla F(\mathbf{x}_t) - \mathbf{g}_t \rangle + |\frac{1}{2}\langle \mathbf{s}_t, (\mathbf{H}_t - \nabla^2 F(\mathbf{x}_t))\mathbf{s}_t \rangle| \\
&\qquad + |\frac{1}{2}\langle \mathbf{s}_t, (\nabla^2 F(\mathbf{x}_t + \tau \mathbf{s}_t) - \nabla^2 F(\mathbf{x}_t))\mathbf{s}_t \rangle| \\
&\leq \langle \mathbf{s}_t, \nabla F(\mathbf{x}_t) - \mathbf{g}_t \rangle + \frac{1}{2}\delta_H \|\mathbf{s}_t\|^2 + \frac{1}{2}L_F \|\mathbf{s}_t\|^3 \\
&\leq \delta_g \Delta_t + \frac{1}{2}\delta_H \Delta_t^2 + \frac{1}{2}L_F \Delta_t^3,
\end{aligned}$$

where $\tau \in [0, 1]$. Similarly, When $\|\mathbf{g}_t\| < \epsilon_g$,

$$F(\mathbf{x}_t + \mathbf{s}_t) - F(\mathbf{x}_t) - m_t(\mathbf{s}_t) \leq \langle \mathbf{s}_t, \nabla F(\mathbf{x}_t) \rangle + \frac{1}{2}\delta_H \Delta_t^2 + \frac{1}{2}L_F \Delta_t^3.$$

∎

By combining Lemmas 2 and 4, Lemma 5 guarantees that, in case $\|\mathbf{g}_t\| \geq \epsilon_g$, the iteration is successful and the update is accepted.

Lemma 5. *Suppose Assumptions 1 and 2, as well as Conditions 1 and 2 hold. Further, suppose at iteration t, we have $\|\mathbf{g}_t\| \geq \epsilon_g$ and*

$$\Delta_t \leq \min\left\{ \frac{\|\mathbf{g}_t\|}{1 + K_H}, \sqrt{\frac{(1-\eta)\|\mathbf{g}_t\|}{12 L_F}}, \frac{(1-\eta)\|\mathbf{g}_t\|}{3} \right\}.$$

Then the iteration t is successful, i.e. $\Delta_{t+1} = \gamma \Delta_t$.

Proof. First, since $\|\mathbf{g}_t\| \geq \epsilon_g$ and $\Delta_t \leq \|\mathbf{g}_t\|/(1 + K_H)$, by Condition 2, we have $\mathbf{s}_t = \mathbf{s}_t^C$ and

$$-m_t(\mathbf{s}_t) \geq \frac{1}{2}\|\mathbf{g}_t\| \min\left\{ \frac{\|\mathbf{g}_t\|}{1 + \|\mathbf{H}_t\|}, \Delta_t \right\} = \frac{1}{2}\|\mathbf{g}_t\|\Delta_t.$$

Now according to Lemma 4, we have

$$\begin{aligned}
1 - \rho_t &= \frac{F(\mathbf{x}_t + \mathbf{s}_t) - F(\mathbf{x}_t) - m_t(\mathbf{s}_t)}{-m_t(\mathbf{s}_t)} \leq \frac{\delta_g \Delta_t + \frac{1}{2}\delta_H \Delta_t^2 + \frac{1}{2}L_F \Delta_t^3}{\frac{1}{2}\|\mathbf{g}_t\|\Delta_t} \\
&= 2\frac{\delta_g}{\|\mathbf{g}_t\|} + \frac{\delta_H}{\|\mathbf{g}_t\|}\Delta_t + \frac{L_F}{\|\mathbf{g}_t\|}\Delta_t^2 \leq \frac{1-\eta}{2} + \frac{\delta_H}{\|\mathbf{g}_t\|}\Delta_t + \frac{L_F}{\|\mathbf{g}_t\|}\Delta_t^2.
\end{aligned}$$

Let

$$r(t) = \frac{L_F}{\|\mathbf{g}_t\|}t^2 + \frac{\delta_H}{\|\mathbf{g}_t\|}t - \frac{1-\eta}{2}.$$

It is not hard to see that $-\delta_H + \sqrt{\delta_H^2 + 2L_F(1-\eta)\|\mathbf{g}_t\|}/(2L_F)$ is the positive root of $r(t)$. Then by the fact that $-y + \sqrt{y^2 + 2L_F(1-\eta\|\mathbf{g}\|_t)}/(2L_F)$ is monotonically decreasing for $y \geq 0$ and Condition 1 ($\delta_H < 1$), it follows

$$\frac{-\delta_H + \sqrt{\delta_H^2 + 2L_F(1-\eta)\|\mathbf{g}_t\|}}{2L_F} \geq \frac{-1 + \sqrt{1 + 2L_F(1-\eta)\|\mathbf{g}_t\|}}{2L_F}.$$

Now, we consider two cases. If $2L_F(1-\eta)\|\mathbf{g}_t\| \leq 1$, it is not hard to show that

$$-1 + \sqrt{1 + 2L_F(1-\eta)\|\mathbf{g}_t\|} \geq \frac{2L_F(1-\eta)\|\mathbf{g}_t\|}{3}.$$

Otherwise, if $2L_F(1-\eta)\|\mathbf{g}_t\| > 1$, then it can be shown that

$$-1 + \sqrt{1 + 2L_F(1-\eta)\|\mathbf{g}_t\|} \geq \sqrt{\frac{L_F(1-\eta)\|\mathbf{g}_t\|}{3}}.$$

By assumption $\Delta_t \leq \min\left\{\sqrt{(1-\eta)\|\mathbf{g}_t\|/(12L_F)}, (1-\eta)\|\mathbf{g}_t\|/3\right\}$, so

$$\Delta_t \leq -1 + \sqrt{1 + 2L_F(1-\eta)\|\mathbf{g}_t\|}/(2L_F),$$

and $r(\Delta_t) \leq 0$. Therefore, it follows,

$$1 - \rho_t \leq \frac{1-\eta}{2} + \frac{\delta_H}{\|\mathbf{g}_t\|}\Delta_t + \frac{L_F}{\|\mathbf{g}_t\|}\Delta_t^2 \leq (1-\eta) + r(\Delta_t) \leq 1-\eta,$$

which implies that the iteration t is successful. ∎

Remark 3. *It can be easily seen that if $\delta_g \leq 3\Delta_t/4$, the above lemma still holds. Indeed,*

$$\delta_g \leq \frac{3}{4}\Delta_t \leq \frac{3}{4}\min\left\{\frac{\|\mathbf{g}_t\|}{1+K_H}, \sqrt{\frac{(1-\eta)\|\mathbf{g}_t\|}{12L_F}}, \frac{(1-\eta)\|\mathbf{g}_t\|}{3}\right\} \leq \frac{(1-\eta)\|\mathbf{g}_t\|}{4}.$$

Although $\delta_g \leq 3\Delta_t/4$ can be looser than what Condition 1 requires, it nonetheless can be used in practice as a rough bound for gradient approximations.

Now we consider that case when $\|\mathbf{g}_t\| \leq \epsilon_g$. As alluded to earlier in this section, in this case we have to rely on the negative curvature of Hessian since dealing with the first-order term in (A.4) is particularly challenging when $\|\mathbf{g}_t\| < \epsilon_g$. Hence, by solely considering the negative eigenvectors of Hessian, we drop the first-order term $\langle \mathbf{s}_t, \nabla F(\mathbf{x}_t)\rangle$ in the quadratic model. Lemma 6 gives the corresponding details.

Lemma 6. *Suppose Assumptions 1 and 2, as well as Conditions 1 and 2 hold. Further, suppose at iteration t, we have $\|\mathbf{g}_t\| < \epsilon_g, \lambda_{\min}(H_t) < -\epsilon_H$ and*

$$\Delta_t \leq \left(\frac{1-\eta}{2}\right)\left(\frac{\nu|\lambda_{\min}(\mathbf{H}_t)|}{L_F+1}\right).$$

Then t^{th} is successful, i.e. $\Delta_{t+1} = \gamma\Delta_t$.

Proof. Here, by Condition 2, we have $\mathbf{s}_t = \mathbf{s}_t^E$, which by (A.3) implies $-m_t(\mathbf{s}_t) \geq \nu|\lambda_{\min}(\mathbf{H}_t)|\Delta_t^2/2$. Hence, recalling (A.4), we have

$$F(\mathbf{x}_t+\mathbf{s}_t) - F(\mathbf{x}_t) - m_t(\mathbf{s}_t) \leq \langle \mathbf{s}_t, \nabla F(\mathbf{x}_t)\rangle + \frac{1}{2}\delta_H\|\mathbf{s}_t\|^2 + \frac{1}{2}L_F\|\mathbf{s}_t\|^3.$$

Since either $\mathbf{s}_t$ or $-\mathbf{s}_t$ could be a searching direction, at least one of

$$\langle \mathbf{s}_t, \nabla F(\mathbf{x}_t)\rangle \leq 0 \qquad \text{or} \qquad \langle -\mathbf{s}_t, \nabla F(\mathbf{x}_t)\rangle \leq 0,$$

is true. Without loss of generality, assume $\langle \mathbf{s}_t, \nabla F(\mathbf{x}_t)\rangle \leq 0$. Hence,

$$F(\mathbf{x}_t+\mathbf{s}_t) - F(\mathbf{x}_t) - m_t(\mathbf{s}_t) \leq \frac{1}{2}\delta_H\|\mathbf{s}_t\|^2 + \frac{1}{2}L_F\|\mathbf{s}_t\|^3.$$

Next, suppose $\Delta_t \leq (1-\eta)\nu\epsilon_H/2$, which from (2.8) implies that $\delta_H \leq (1-\eta)\nu\epsilon_H/2$. We have

$$\begin{aligned}1-\rho_t &= \frac{F(\mathbf{x}_t+\mathbf{s}_t) - F(\mathbf{x}_t) - m_t(\mathbf{s}_t)}{-m_t(\mathbf{s}_t)} \leq \frac{\delta_H\Delta_t^2/2 + L_F\Delta_t^3/2}{\nu|\lambda_{\min}(\mathbf{H}_t)|\Delta_t^2/2} = \frac{\delta_H + L_F\Delta_t}{\nu|\lambda_{\min}(\mathbf{H}_t)|}\\ &\leq \frac{(1-\eta)\nu\epsilon_H/2 + L_F(1-\eta)\nu|\lambda_{\min}(H_t)|/(2(L_F+1))}{\nu|\lambda_{\min}(\mathbf{H}_t)|} < 1-\eta.\end{aligned}$$

Now, consider $\Delta_t \geq (1-\eta)\nu\epsilon_H/2$, which from (2.8) implies that $\delta_H \leq \Delta_t$. Similarly, we have

$$\begin{aligned}1-\rho_t &= \frac{F(\mathbf{x}_t+\mathbf{s}_t) - F(\mathbf{x}_t) - m_t(\mathbf{s}_t)}{-m_t(\mathbf{s}_t)}\\ &\leq \frac{\delta_H\Delta_t^2/2 + L_F\Delta_t^3/2}{\nu|\lambda_{\min}(\mathbf{H}_t)|\Delta_t^2/2}\\ &= \frac{(L_F+1)\Delta_t}{\nu|\lambda_{\min}(\mathbf{H}_t)|} < (1-\eta)/2 < 1-\eta.\end{aligned}$$

Hence, in both cases, we have $\rho_t \geq \eta$ and the iteration is successful. ∎

Based on Lemmas 5 and 6, the following lemma helps to get the lower bound of Δ_t, whose proof could be found in [246].

Lemma 7. *Under Assumptions 1 and 2 and Conditions 1 and 2, for Algorithm 1 and for all t, we have*

$$\Delta_t \geq \frac{1}{\gamma} \min\left\{\frac{\epsilon_g}{1+K_H}, \sqrt{\frac{(1-\eta)\epsilon_g}{12L_H}}, \frac{(1-\eta)\epsilon_g}{3}, \frac{(1-\eta)\nu\epsilon_H}{2(L_F+1)}\right\}.$$

As a consequence, we now can give the upper bound on the number of successful iterations.

Lemma 8 (Successful iterations)**.** *Let $\mathscr{T}_{succ}$ denote the set of all the successful iterations before Algorithm 1 stops. Under Assumptions 1 and 2 and Conditions 1 and 2, the number of successful iterations is upper bounded by*

$$|\mathscr{T}_{succ}| \leq \frac{F(\boldsymbol{x}_0) - F(\boldsymbol{x}^*)}{C\epsilon_H \min\{\epsilon_g^2, \epsilon_H^2\}},$$

where C is a constant depending on L_F, K_H, η, ν.

Proof. Suppose Algorithm 1 doesn't terminate at iteration t. Then either $\|\mathbf{g}_t\| \geq \epsilon_g$ or $\lambda_{\min}(\mathbf{H}_t) \leq -\epsilon_H$. If $\|\mathbf{g}_t\| \geq \epsilon_g$, according to (A.1), we have

$$-m_t(\mathbf{s}_t) \geq \frac{1}{2}\|\mathbf{g}_t\| \min\left\{\frac{\|\mathbf{g}_t\|}{1+\|\mathbf{H}_t\|}, \Delta_t\right\} \geq \frac{1}{2}\epsilon_g \min\left\{\frac{\epsilon_g}{1+K_H}, C_0\epsilon_g, C_1\epsilon_H\right\} \geq C_2\epsilon_g \min\left\{\epsilon_g, \epsilon_H\right\}.$$

Similarly, in the second case $\lambda_{\min}(\mathbf{H}_t) \leq -\epsilon_H$, from (A.3),

$$-m_t(\mathbf{s}_t) \geq \frac{1}{2}\nu\|\lambda_{\min}(\mathbf{H}_t)\|\Delta_t^2 \geq C_3\epsilon_H \min\{\epsilon_g^2, \epsilon_H^2\}.$$

Since $F(\mathbf{x}_t)$ is monotonically decreasing as t increases, we have

$$\begin{aligned} F(\mathbf{x}_0) - F(\mathbf{x}^*) &\geq \sum_{t=0}^{\infty} F(\mathbf{x}_t) - F(\mathbf{x}_{t+1}) \geq \sum_{t\in\mathscr{T}_{\text{succ}}} F(\mathbf{x}_t) - F(\mathbf{x}_{t+1}) \\ &\geq \eta \sum_{t\in\mathscr{T}_{\text{succ}}} \min\left\{C_2\epsilon_g \min\left\{\epsilon_g, \epsilon_H\right\}, C_3\epsilon_H \min\{\epsilon_g^2, \epsilon_H^2\}\right\} \\ &\geq |\mathscr{T}_{\text{succ}}|\, C\epsilon_H \min\{\epsilon_g^2, \epsilon_H^2\}. \end{aligned}$$

Since one of the above cases must happen for a successful iteration, it follows,

$$|\mathscr{T}_{\text{succ}}| \leq \frac{F(\mathbf{x}_0) - F(\mathbf{x}^*)}{C\epsilon_H \min\{\epsilon_g^2, \epsilon_H^2\}}.$$

∎

Using the above lemma, the proof of following theorem could be found in [246]. In this section, we give the proofs of some lemmas mentioned in the main text.

A.1.1 Proof of Lemma 12

Proof. When $\|\mathbf{g}_t\| \geq \epsilon_g$, using Taylor expansion of $F(\mathbf{x})$ at point $\mathbf{x}_t$,

$$\begin{aligned}
F(\mathbf{x}_t + \mathbf{s}_t) - F(\mathbf{x}_t) - m_t(\mathbf{s}_t) &= \langle \mathbf{s}_t, \nabla F(\mathbf{x}_t) - \mathbf{g}_t \rangle + \frac{1}{2}\langle \mathbf{s}_t, (\nabla^2 F(\mathbf{x}_t + \tau \mathbf{s}_t) - \mathbf{H}_t)\mathbf{s}_t \rangle - \frac{\sigma_t}{3}\|\mathbf{s}_t\|^3 \\
&\leq \langle \mathbf{s}_t, \nabla F(\mathbf{x}_t) - \mathbf{g}_t \rangle + |\frac{1}{2}\langle \mathbf{s}_t, (\mathbf{H}_t - \nabla^2 F(\mathbf{x}_t + \tau \mathbf{s}_t)\mathbf{s}_t \rangle| - \frac{\sigma_t}{3}\|\mathbf{s}_t\|^3 \\
&\leq \langle \mathbf{s}_t, \nabla F(\mathbf{x}_t) - \mathbf{g}_t \rangle + |\frac{1}{2}\langle \mathbf{s}_t, (\mathbf{H}_t - \nabla^2 F(\mathbf{x}_t))\mathbf{s}_t \rangle| \\
&+ |\frac{1}{2}\langle \mathbf{s}_t, (\nabla^2 F(\mathbf{x}_t + \tau \mathbf{s}_t) - \nabla^2 F(\mathbf{x}_t))\mathbf{s}_t \rangle| - \frac{\sigma_t}{3}\|\mathbf{s}_t\|^3 \\
&\leq \langle \mathbf{s}_t, \nabla F(\mathbf{x}_t) - \mathbf{g}_t \rangle + \frac{1}{2}\delta_H\|\mathbf{s}_t\|^2 + (\frac{L_F}{2} - \frac{\sigma_t}{3})\|\mathbf{s}_t\|^3, \\
&\leq \delta_g\|\mathbf{s}_t\| + \frac{1}{2}\delta_H\|\mathbf{s}_t\|^2 + (\frac{L_F}{2} - \frac{\sigma_t}{3})\|\mathbf{s}_t\|^3,
\end{aligned}$$

where $\tau \in [0, 1]$. Similarly, when $\|\mathbf{g}_t\| < \epsilon_g$,

$$F(\mathbf{x}_t + \mathbf{s}_t) - F(\mathbf{x}_t) - m_t(\mathbf{s}_t) \leq \langle \mathbf{s}_t, \nabla F(\mathbf{x}_t) \rangle + \frac{1}{2}\delta_H\|\mathbf{s}_t\|^2 + (\frac{L_F}{2} - \frac{\sigma_t}{3})\|\mathbf{s}_t\|^3.$$

∎

A.2 Proof of Theorem 2

First let's denote $\mathscr{T}_{\text{succ}}$ as the set of all the successful iteration and $\mathscr{T}_{\text{fail}}$ as the set of all the failure iterations. Now we will upper bound the iteration complexity $T := |\mathscr{T}_{\text{succ}}| + |\mathscr{T}_{\text{fail}}|$. First we present the following lemma that gives an upper bound of $|\mathscr{T}_{\text{fail}}|$.

Lemma 9. *In Algorithm 2, suppose we have $\sigma_t \leq C$, where C is some constant, for all the iteration t before it stops. Then we have $|\mathscr{T}_{fail}| \leq |\mathscr{T}_{succ}| + \mathcal{O}(1)$.*

Proof. Since $\sigma_t \leq C$, then $\sigma_T = \sigma_0 \gamma^{|\mathscr{T}_{\text{succ}}| - |\mathscr{T}_{\text{fail}}|} \leq C$. Then, we immediately obtain

$$|\mathscr{T}_{\text{fail}}| \leq \log(C/\sigma_0)/\log\gamma + |\mathscr{T}_{\text{fail}}| = |\mathscr{T}_{\text{succ}}| + \mathcal{O}(1).$$

Now, for the rest of the analysis, we first show that there is a uniform upper bound for all σ_t and, subsequently, we obtain a bound on the number of all the successful iterations.

We now present Lemma 10, which is very similar to [246, Lemma 6], but is slightly more refined. The main difference lies in the quantity K_t defined in Lemma 10. In [246, Lemma 6] a simple global upper bound of this quantity is used. However, here, we retain its local nature, which is found to be crucial in proving Lemma 13. This is a subtle distinction that arises as a result of using gradient approximations here, compared with exact gradients in [246].

Lemma 10 (Cauchy Point)**.** *When $\|\mathbf{g}_t\| \geq \epsilon_g$, let*

$$\boldsymbol{s}_t^C = \arg\min_{\alpha \geq 0} m_t(-\alpha \mathbf{g}_t).$$

Then, we have

$$\|\boldsymbol{s}_t^C\| = \frac{1}{2\sigma_t}(\sqrt{K_t^2 + 4\sigma_t\|\mathbf{g}_t\|} - K_t), \tag{A.5a}$$

$$-m_t(\boldsymbol{s}_t^C) \geq \max\left\{\frac{1}{12}\|\boldsymbol{s}_t^C\|^2(\sqrt{K_t^2 + 4\sigma_t\|\mathbf{g}_t\|} - K_t), \frac{\|\mathbf{g}_t\|}{2\sqrt{3}}\min\left\{\frac{\|\mathbf{g}_t\|}{|K_t|}, \frac{\|\mathbf{g}_t\|}{\sqrt{\sigma_t\|\mathbf{g}_t\|}}\right\}\right\}, \tag{A.5b}$$

where $K_t = \langle \mathbf{H}_t\mathbf{g}_t, \mathbf{g}_t\rangle / \|\mathbf{g}_t\|^2$.

Proof. The proof is organized as follows. we will first use the definition of Cauchy Point to get an expression in terms of $\mathbf{s}_t^C$. Subsequently, we use that fact that $m_t(\mathbf{s}_t^C) \leq m_t(\alpha\mathbf{g}_t)$, $\forall \alpha \geq 0$ to bound $m_t(\mathbf{s}_t^C)$ by leveraging the quadratic form of $m_t(\alpha\mathbf{g}_t)$ in terms of α. First, we have

$$\langle \mathbf{g}_t, \mathbf{s}_t^C\rangle + \langle \mathbf{s}_t^C, \mathbf{H}_t\mathbf{s}_t^C\rangle + \sigma_t\|\mathbf{s}_t^C\|^3 = 0.$$

Since $\mathbf{s}_t^C = -\alpha\mathbf{g}_t$ for some $\alpha > 0$,

$$-\alpha\|\mathbf{g}_t\|^2 + \alpha^2\langle \mathbf{g}_t, \mathbf{H}_t\mathbf{g}_t\rangle + \sigma_t\alpha^3\|\mathbf{g}_t\|^3 = 0.$$

We can find explicit formula for such α by finding the roots of the quadratic function

$$r(\alpha) = -\|\mathbf{g}_t\|^2 + \alpha\langle \mathbf{g}_t, \mathbf{H}_t\mathbf{g}_t\rangle + \sigma_t\alpha^2\|\mathbf{g}_t\|^3.$$

We have

$$\alpha = \frac{-\langle \mathbf{g}_t, \mathbf{H}_t\mathbf{g}_t\rangle + \sqrt{\langle \mathbf{g}_t, \mathbf{H}_t\mathbf{g}_t\rangle^2 + 4\sigma_t\|\mathbf{g}_t\|^5}}{2\sigma_t\|\mathbf{g}_t\|^3},$$

and

$$2\alpha\sigma_t\|\mathbf{g}_t\| = \sqrt{K_t^2 + 4\sigma_t\|\mathbf{g}_t\|} - K_t.$$

Hence, it follows that

$$\|\mathbf{s}_t^C\| = \alpha\|\mathbf{g}_t\| = \frac{1}{2\sigma_t}(\sqrt{K_t^2 + 4\sigma_t\|\mathbf{g}_t\|} - K_t).$$

Now, from [41, Lemma 2.1], we get

$$-m_t(\mathbf{s}_t^C) \geq \frac{1}{6}\sigma_t\|\mathbf{s}_t^C\|^3 = \frac{1}{6}\sigma_t\|\mathbf{s}_t^C\|^2\alpha\|\mathbf{g}_t\| = \frac{1}{12}\|\mathbf{s}_t^C\|^2(\sqrt{K_t^2 + 4\sigma_t\|\mathbf{g}_t\|} - K_t).$$

Alternatively, we have

$$\begin{aligned} m_t(\mathbf{s}_t^C) \leq m_t(-\alpha\mathbf{g}_t) &= -\alpha\|\mathbf{g}_t\|^2 + \frac{1}{2}\alpha^2\langle \mathbf{g}_t, \mathbf{H}_t\mathbf{g}_t\rangle + \frac{\alpha^3}{3}\sigma_t\|\mathbf{g}_t\|^3 \\ &= \frac{\alpha\|\mathbf{g}_t\|^2}{6}(-6 + 3\alpha K_t + 2\alpha^2\sigma_t\|\mathbf{g}_t\|). \end{aligned}$$

Consider the quadratic part,

$$r(\alpha) = -6 + 3\alpha K_t + 2\alpha^2 \sigma_t \|\mathbf{g}_t\|.$$

We have $r(\alpha) \le 0$ for $\alpha \in [0, \bar{\alpha}]$, where

$$\bar{\alpha} = \frac{-3K_t + \sqrt{9K_t^2 + 48\sigma_t\|\mathbf{g}_t\|}}{4\sigma_t\|\mathbf{g}_t\|}.$$

We can express $\bar{\alpha}$ as

$$\bar{\alpha} = \frac{12}{3K_t + \sqrt{9K_t^2 + 48\sigma_t\|\mathbf{g}_t\|}}.$$

Note that,

$$\sqrt{9K_t^2 + 48\sigma_t\|\mathbf{g}_t\|} \le 3|K_t| + 4\sqrt{3\sigma_t\|\mathbf{g}_t\|} \le 8\sqrt{3}\max\{|K_t|, \sqrt{\sigma_t\|\mathbf{g}_t\|}\}.$$

Also,

$$3K_t \le 2\sqrt{3}\max\{|K_t|, \sqrt{\sigma_t\|\mathbf{g}_t\|}\} \le 4\sqrt{3}\max\{|K_t|, \sqrt{\sigma_t\|\mathbf{g}_t\|}\}.$$

Hence, defining

$$\alpha_0 = \frac{1}{\sqrt{3}\max\{|K_t|, \sqrt{\sigma_t\|\mathbf{g}_t\|}\}},$$

it is clear that $0 \le \alpha_0 \le \bar{\alpha}$. With α_0, we have

$$r(\alpha_0) \le 2/3 + 3/\sqrt{3} - 6 \le -3.$$

So finally, we get

$$\begin{aligned} m_t(\mathbf{s}_t^C) \le \frac{-3\|\mathbf{g}_t\|^2}{6\sqrt{3}} \frac{1}{\max\{|K_t|, \sqrt{\sigma_t\|\mathbf{g}_t\|}\}} &= \frac{-\|\mathbf{g}_t\|^2}{2\sqrt{3}} \min\left\{\frac{1}{|K_t|}, \frac{1}{\sqrt{\sigma_t\|\mathbf{g}_t\|}}\right\} \\ &= \frac{-\|\mathbf{g}_t\|}{2\sqrt{3}} \min\left\{\frac{\|\mathbf{g}_t\|}{|K_t|}, \frac{\|\mathbf{g}_t\|}{\sqrt{\sigma_t\|\mathbf{g}_t\|}}\right\}. \end{aligned}$$

∎

When $\mathbf{H}_t$ has a negative eigenvalue, Eigen Point has the following properties.

Lemma 11 (Eigen Point)**.** *Suppose* $\lambda_{\min}(\mathbf{H}_t) < 0$ *and for some* $\nu \in (0, 1]$*, let*

$$\boldsymbol{s}_t^E = \arg\min_{\alpha \in R} m_t(\alpha \boldsymbol{u}_t),$$

where $\boldsymbol{u}_t$ *is the approximate most negative eigenvector defined as*

$$\langle \boldsymbol{u}_t, \mathbf{H}_t \boldsymbol{u}_t \rangle \le \nu \lambda_{\min}(\mathbf{H}_t)\|\boldsymbol{u}_t\|^2 \le 0.$$

We have

$$\left\| \boldsymbol{s}_t^E \right\| \geq \frac{\nu \left|\lambda_{\min}(\mathbf{H}_t)\right|}{\sigma_t}, \tag{A.6a}$$

$$-m_t(\boldsymbol{s}_t^E) \geq \frac{\nu|\lambda_{\min}(\mathbf{H}_t)|}{6}\|\boldsymbol{s}_t^E\|^2. \tag{A.6b}$$

Proof. Again, we know that

$$\langle \mathbf{g}_t, \mathbf{s}_t^E\rangle + \langle \mathbf{s}_t^E, \mathbf{H}_t\mathbf{s}_t^E\rangle + \sigma_t\|\mathbf{s}_t^E\|^3 = 0.$$

Meanwhile, since $-\mathbf{s}_t$ would keep the last two term as the same value, w.l.o.g, we could assume $\langle \mathbf{g}_t, \mathbf{s}_t^E\rangle \leq 0$, which means

$$\langle \mathbf{s}_t^E, \mathbf{H}_t\mathbf{s}_t^E\rangle + \sigma_t\|\mathbf{s}_t^E\|^3 \geq 0.$$

Now, from [41, Lemma 2.1]

$$-m_t(\mathbf{s}_t) \geq \frac{1}{6}\sigma_t\|\mathbf{s}_t\|^3 \geq -\frac{1}{6}\langle \mathbf{s}_t^E, \mathbf{H}_t\mathbf{s}_t^E\rangle \geq \frac{1}{6}\nu|\lambda_{\min}(H_t)|\|\mathbf{s}_t^E\|^2.$$

∎

The following lemma gives a bound on the difference between the decrease of the objective function and the value of the quadratic model $m(\mathbf{s}_t)$. This lemma can be easily obtained by the smoothness assumption of the objective function; the detailed proof is included in Appendix A.1.1.

Lemma 12. *Under Assumptions 1 and 2, we have*

$$F(\boldsymbol{x}_t + \boldsymbol{s}_t) - F(\boldsymbol{x}_t) - m_t(\boldsymbol{s}_t) \leq \begin{cases} \delta_g\|\boldsymbol{s}_t\| + \frac{1}{2}\delta_H\|\boldsymbol{s}_t\|^2 + (\frac{L_F}{2} - \frac{\sigma_t}{3})\|\boldsymbol{s}_t\|^3, & \|\mathbf{g}_t\| \geq \epsilon_g, \\ \langle \boldsymbol{s}_t, \nabla F(\boldsymbol{x}_t)\rangle + \frac{1}{2}\delta_H\|\boldsymbol{s}_t\|^2 + (\frac{L_F}{2} - \frac{\sigma_t}{3})\|\boldsymbol{s}_t\|^3, & \|\mathbf{g}_t\| < \epsilon_g. \end{cases} \tag{A.7}$$

Based on the above lemmas, the following lemma shows that iteration t is successful when $\|\mathbf{g}_t\| \geq \epsilon_g$.

Lemma 13. *Suppose Assumptions 1 and 2 and Condition 3 hold. Further, suppose at iteration t, we have $\|\mathbf{g}_t\| \geq \epsilon_g$ and $\sigma_t \geq 2L_F$. Then the iteration t is successful, i.e. $\sigma_{t+1} = \sigma_t/\gamma$.*

Proof. Using Lemma 12, we get

$$\begin{aligned} F(\mathbf{x}_t + \mathbf{s}_t^C) - F(\mathbf{x}_t) - m_t(\mathbf{s}_t^C) &\leq \delta_g\|\mathbf{s}_t^C\| + \frac{1}{2}\delta_H\|\mathbf{s}_t^C\|^2 + (\frac{L_F}{2} - \frac{\sigma_t}{3})\|\mathbf{s}_t^C\|^3 \\ &\leq \delta_g\|\mathbf{s}_t^C\| + \frac{1}{2}\delta_H\|\mathbf{s}_t^C\|^2, \end{aligned}$$

since $\sigma_t \geq 2L_F$. We divide it to two cases.

First, if $K_t = \langle \mathbf{H}_t\mathbf{g}_t, \mathbf{g}_t\rangle / \|\mathbf{g}_t\|^2 \leq 0$, then from (A.5a), it follows

$$\|\mathbf{s}_t^C\| \geq \frac{1}{2\sigma_t}\sqrt{4\sigma_t\|\mathbf{g}_t\|} = \sqrt{\|\mathbf{g}_t\|/\sigma_t}.$$

Using [41, Lemma 2.1], we get

$$\begin{aligned}1-\rho_t &= \frac{F(\mathbf{x}_t+\mathbf{s}_t)-F(\mathbf{x}_t)-m_t(\mathbf{s}_t)}{-m_t(\mathbf{s}_t)} \leq \frac{\delta_g\left\|\mathbf{s}_t^C\right\| + \frac{1}{2}\delta_H\left\|\mathbf{s}_t^C\right\|^2}{\frac{\sigma_t\|\mathbf{s}_t^C\|^3}{6}} = \frac{\delta_g + \frac{1}{2}\delta_H\left\|\mathbf{s}_t^C\right\|}{\frac{\sigma_t\|\mathbf{s}_t^C\|^2}{6}} \\ &\leq \frac{6\delta_g}{\|\mathbf{g}_t\|} + \frac{3\delta_H}{\sqrt{2\epsilon_g L_F}} \leq \frac{1-\eta}{2} + \frac{1-\eta}{2} = 1-\eta,\end{aligned}$$

where the last inequality follows from the condition on δ_g and δ_H.

For the second case where $K_t > 0$, from (A.5a) in Lemma 10, it follows that

$$\|\mathbf{s}_t^C\| = \frac{\sqrt{K_t^2 + 4\sigma_t\|\mathbf{g}_t\|} - K_t}{2\sigma_t} = \frac{2\|\mathbf{g}_t\|}{\sqrt{K_t^2 + 4\sigma_t\|\mathbf{g}_t\|} + K_t}.$$

Now we consider two cases: **(a)** $K_t^2 \geq \sigma_t\|\mathbf{g}_t\|$ and (b) $K_t^2 \leq \sigma_t\|\mathbf{g}_t\|$.

(a) When $K_H^2 \geq K_t^2 \geq \sigma_t\|\mathbf{g}_t\|$, from above equality we have

$$\|\mathbf{s}_t^C\| \leq \frac{\|\mathbf{g}_t\|}{K_t}.$$

Meanwhile, since $K_t^2 \geq \sigma_t\|\mathbf{g}_t\|$, from Lemma 10, we have

$$-m_t(\mathbf{s}_t^C) \geq \frac{\|\mathbf{g}_t\|}{2\sqrt{3}}\min\left\{\frac{\|\mathbf{g}_t\|}{|K_t|}, \frac{\|\mathbf{g}_t\|}{\sqrt{\sigma_t\|\mathbf{g}_t\|}}\right\} = \frac{\|\mathbf{g}_t\|^2}{2\sqrt{3}K_t}.$$

Combine above inequality together, it follows

$$\begin{aligned}1-\rho_t &= \frac{F(\mathbf{x}_t+\mathbf{s}_t)-F(\mathbf{x}_t)-m_t(\mathbf{s}_t)}{-m_t(\mathbf{s}_t)} \leq \frac{\delta_g\left\|\mathbf{s}_t^C\right\| + \frac{1}{2}\delta_H\left\|\mathbf{s}_t^C\right\|^2}{\frac{\|\mathbf{g}_t\|^2}{2\sqrt{3}K_t}} \leq \frac{\delta_g\frac{\|\mathbf{g}_t\|}{K_t} + \frac{1}{2}\delta_H(\frac{\|\mathbf{g}_t\|}{K_t})^2}{\frac{\|\mathbf{g}_t\|^2}{2\sqrt{3}K_t}} \\ &= \frac{2\sqrt{3}\delta_g}{\|\mathbf{g}_t\|} + \frac{\sqrt{3}\delta_H}{K_t} \leq \frac{2\sqrt{3}\delta_g}{\|\mathbf{g}_t\|} + \frac{\sqrt{3}\delta_H}{\sqrt{2L_F\epsilon_g}} \leq \frac{1-\eta}{2} + \frac{1-\eta}{2} = 1-\eta.\end{aligned}$$

(b) When $K_t^2 \leq \sigma_t\|\mathbf{g}_t\|$, we have

$$\|\mathbf{s}_t^C\| \leq \frac{\|\mathbf{g}_t\|}{\sqrt{\|\mathbf{g}_t\sigma_t\|}},$$

and

$$-m_t(\mathbf{s}_t^C) \geq \frac{\|\mathbf{g}_t\|}{2\sqrt{3}} \min\left\{\frac{\|\mathbf{g}_t\|}{|K_t|}, \frac{\|\mathbf{g}_t\|}{\sqrt{\sigma_t\|\mathbf{g}_t\|}}\right\} \geq \frac{\|\mathbf{g}_t\|^{3/2}}{2\sqrt{3}\sqrt{\sigma_t}}.$$

Then,

$$\begin{aligned}1-\rho_t &= \frac{F(\mathbf{x}_t+\mathbf{s}_t)-F(\mathbf{x}_t)-m_t(\mathbf{s}_t)}{-m_t(\mathbf{s}_t)} \leq \frac{\delta_g\|\mathbf{s}_t^C\| + \frac{1}{2}\delta_H\|\mathbf{s}_t^C\|^2}{\frac{\|\mathbf{g}_t\|^{3/2}}{2\sqrt{3}\sqrt{\sigma_t}}} = \frac{2\sqrt{3}\delta_g}{\|\mathbf{g}_t\|} + \frac{\sqrt{3}\delta_H}{\sqrt{\sigma_t\epsilon_g}}\\ &\leq \frac{2\sqrt{3}\delta_g}{\|\mathbf{g}_t\|} + \frac{\sqrt{3}\delta_H}{\sqrt{2L_F\epsilon_g}} \leq \frac{1-\eta}{2} + \frac{1-\eta}{2} = 1-\eta.\end{aligned}$$

From the above, it follows that t^{th} iteration is successful, i.e. $\sigma_{t+1} = \sigma_t/\gamma$, when $\|\mathbf{g}_t\| \geq \epsilon_g$.
■

The following lemma, whose proof is similar to [246, Lemma 9], helps bound $F(\mathbf{x}_t+\mathbf{s}_t) - F(\mathbf{x}_t) - m_t(\mathbf{s}_t)$ when Hessian has negative eigenvalues.

Lemma 14. *[246, Lemma 9] Suppose Assumptions 1 and 2 and Condition 3 hold and $\sigma_t \geq 2L_F$. Then, if $\lambda_{\min}(\mathbf{H}_t) < -\epsilon_H$, we have*

$$\frac{\delta_H}{2}\|\boldsymbol{s}_t\|^2 + \left(\frac{L_F}{2} - \frac{\sigma_t}{3}\right)\|\boldsymbol{s}_t\|^3 \leq \frac{\delta_H}{2}\|\boldsymbol{s}_t^E\|^2.$$

Proof. If $\|\mathbf{s}_t\| \leq \mathbf{s}_e^E$, then based on the condition of $\sigma_t \geq 2L_F$, we have

$$\frac{1}{2}\delta_H\|\mathbf{s}_t\|^2 + (\frac{1}{2}L_F - \frac{\sigma_t}{3})\|\mathbf{s}_t\|^3 \leq \frac{1}{2}\delta_H\|\mathbf{s}_t\|^2 \leq \frac{\delta_H}{2}\|\mathbf{s}_t^E\|^2.$$

When $\|\mathbf{s}_t\| > \|\mathbf{s}_e^E\|$, since $L_f \leq \sigma_t/2$,

$$\begin{aligned}\frac{1}{2}\delta_H\|\mathbf{s}_t\|^2 + (\frac{1}{2}L_F - \frac{\sigma_t}{3})\|\mathbf{s}_t\|^3 &\leq \frac{1}{2}\delta_H\|\mathbf{s}_t\|^2 - \frac{\sigma_t}{12}\|\mathbf{s}_t\|^3\\ &\leq \frac{1}{2}\delta_H\|\mathbf{s}_t\|^2 - \frac{\sigma_t}{12}\|\mathbf{s}_t^E\|\|\mathbf{s}_t\|^2\\ &\leq \frac{1}{2}\delta_H\|\mathbf{s}_t\|^2 - \frac{\nu|\lambda_{\min}(\mathbf{H}_t)|}{12}\|\mathbf{s}_t\|^2\\ &\leq \left((1-\eta)\nu|\lambda_{\min}(\mathbf{H}_t)| - \nu|\lambda_{\min}(\mathbf{H}_t)|\right)\|\mathbf{s}_t\|^2/12\\ &\leq 0 \leq \frac{\delta_H}{2}\|\mathbf{s}_t^E\|^2,\end{aligned}$$

where the third and fourth inequalities follow from (A.6a) and (2.11), respectively. ■

Then, the following lemma shows Eigen Points also yields a descent similarly as in Lemma 6.

Lemma 15. *Suppose Assumptions 1 and 2 and Conditions 3 and 4 hold. Further, suppose at iteration t, we have $\lambda_{\min}(\mathbf{H}_t) < -\epsilon_H$, $\|\mathbf{g}_t\| \le \epsilon_g$ and $\sigma_t \ge 2L_F$. Then iteration t is successful, i.e. $\sigma_{t+1} = \sigma_t/\gamma$.*

Proof. If $\lambda_{\min}(\mathbf{H}_t) < -\epsilon_H$ and $\|\mathbf{g}_t\| \le \epsilon_g$, recall that our sub-problem is

$$m_t(\mathbf{s}) = \frac{1}{2}\langle \mathbf{s}, \mathbf{H}_t\mathbf{s}\rangle + \frac{\sigma_t}{3}\|\mathbf{s}\|^3,$$

and we pick the Eigen Point direction, i.e., $\mathbf{s}_t = \mathbf{s}_t^E$. Now, it is clear that if $\mathbf{s}_t$ is an approximate solution of the above problem, then so is $-\mathbf{s}_t$. Similar to Lemma 6, without loss of generality, assume $\langle \mathbf{s}_t, \nabla F(\mathbf{x}_t)\rangle \le 0$. Then according to (A.7)

$$F(\mathbf{x}_t + \mathbf{s}_t) - F(\mathbf{x}_t) - m_t(\mathbf{s}_t) \le \frac{\delta_H}{2}\|\mathbf{s}_t\|^2 + \frac{(L_F - \sigma_t/3)}{2}\|\mathbf{s}_t\|^3.$$

Therefore, according to (A.6b) and Lemma 14,

$$\begin{aligned} 1 - \rho_t = \frac{F(\mathbf{x}_t + \mathbf{s}_t) - F(\mathbf{x}_t) - m_t(\mathbf{s}_t)}{-m_t(\mathbf{s}_t)} &\le \frac{\delta_H\|\mathbf{s}_t\|^2 + (L_F - \sigma_t/3)\|\mathbf{s}_t\|^3}{-2m_t(\mathbf{s}_t)} \\ \le \frac{\delta_H\|\mathbf{s}_t\|^2/2}{\nu|\lambda_{\min}(\mathbf{H}_t)|\|\mathbf{s}_t\|^2/6} &= \frac{3\delta_H}{\nu\|\lambda_{\min}(\mathbf{H}_t)\|} \le 1 - \eta, \end{aligned}$$

which means the iteration t is successful.

With the help of the above lemmas, we can now show an upper bound for σ_t, as in Lemma 16.

Lemma 16. *Under Assumptions 1 and 2 and Conditions 3 and 4, we have $\sigma_t \le \max\{2\gamma L_F, \sigma_0\}$ for all t.*

Proof. We consider two cases. First, suppose $\sigma_0 \le 2\gamma L_F$. We prove the claim in this case by contradiction. Suppose that t^{th} iteration is the first unsuccessful iteration such that $\sigma_{t+1} = \gamma\sigma_t \ge 2\gamma L_F$, which implies that $\sigma_t \ge 2L_F$. However, according to Lemmas 13 and 15, respectively, if $\|\mathbf{g}_t\| \ge \epsilon_g$ or $\lambda_{\min}(\mathbf{H}_t) \le -\epsilon_H$, then the iteration is successful and we must have $\sigma_{t+1} = \sigma_t/\gamma \le \sigma_t$, which is a contradiction. Second, consider the case where $\sigma_0 > 2\gamma L_F$. According to Lemmas 13 and 15, any iteration t with $\sigma_t \ge 2L_F$ is successful, which implies that $\sigma_t \le \sigma_0, ; \forall t$.

Now we upper bound the number of all successful iterations $|\mathscr{T}_{\text{succ}}|$, which is shown in Lemma 17. The proof is similar to [246, Lemma 13].

Lemma 17 (Successful iterations)**.** *Under Assumptions 1 and 2 and Conditions 3 and 4, the number of successful iterations is upper bounded by*

$$|\mathscr{T}_{succ}| \le \left(\frac{F(x_0) - F(x^*)}{C}\right)\max\{\epsilon_g^{-2}, \epsilon_H^{-3}\}.$$

A.3 Proof of Theorem 3

As a result of the stricter condition imposed by Condition 6, we need to refine some lemmas in Section 2.2.2.1. First, we need use the following result which gives conditions for a successful iteration when $\|\mathbf{g}_t\| \geq \epsilon_g$.

Lemma 18. *Suppose Assumptions 1 and 2 and Condition 5 hold. If $\sigma_t \geq 2L_F$ and $\|\mathbf{g}_t\| > \epsilon_g$, then*

$$\delta_g\|\boldsymbol{s}_t\| + \frac{1}{2}\delta_H\|\boldsymbol{s}_t\|^2 + \left(\frac{1}{2}L_F - \frac{\sigma_t}{3}\right)\|\boldsymbol{s}_t\|^3 \leq \delta_g\|\boldsymbol{s}_t^C\| + \frac{1}{2}\delta_H\|\boldsymbol{s}_t^C\|^2. \tag{A.8}$$

Proof. We consider the following two cases:

i. If $\|\mathbf{s}_t\| \leq \|\mathbf{s}_t^C\|$, then from the assumption of σ_t, it immediately follows that

$$\delta_g\|\mathbf{s}_t\| + \frac{1}{2}\delta_H\|\mathbf{s}_t\|^2 + \left(\frac{1}{2}L_F - \frac{\sigma_t}{3}\right)\|\mathbf{s}_t\|^3 \leq \delta_g\|\mathbf{s}_t\| + \frac{1}{2}\delta_H\|\mathbf{s}_t\|^2 \leq \delta_g\|\mathbf{s}_t^C\| + \frac{1}{2}\delta_H\|\mathbf{s}_t^C\|^2.$$

ii. If $\|\mathbf{s}_t\| \geq \left\|\mathbf{s}_t^C\right\|$, first, since $L_F \leq \sigma_t/2$,

$$\delta_g\|\mathbf{s}_t\| + \frac{1}{2}\delta_H\|\mathbf{s}_t\|^2 + \left(\frac{1}{2}L_F - \frac{\sigma_t}{3}\right)\|\mathbf{s}_t\|^3 \leq \delta_g\|\mathbf{s}_t\| + \frac{1}{2}\delta_H\|\mathbf{s}_t\|^2 - \frac{\sigma_t}{12}\|\mathbf{s}_t\|^3.$$

Now let's define function $r(x) = \delta_g + \delta_H x/2 - \sigma_t x^2/12$. The derivative of $r(x)$ is given by $r'(x) = \delta_H/2 - \sigma_t x/6$. For any $x \geq \left\|\mathbf{s}_t^C\right\|$, according to (A.5a), we have

$$r'(x) \leq \frac{1}{2}\delta_H - \frac{1}{6}\sigma_t\left\|\mathbf{s}_t^C\right\| \leq \frac{1}{2}\delta_H - \frac{\sqrt{K_H^2 + 4\sigma_t\|\mathbf{g}_t\|} - K_H}{12} \leq 0.$$

Therefore,

$$\begin{aligned}
r\left(\|\mathbf{s}_t\|\right) &\leq r\left(\left\|\mathbf{s}_t^C\right\|\right) = \delta_g + \frac{1}{2}\delta_H\left\|\mathbf{s}_t^C\right\| - \frac{1}{12}\sigma_t\left\|\mathbf{s}_t^C\right\|^2 \\
&\leq \delta_g + \left(\frac{1}{2}\delta_H - \frac{\sqrt{K_H^2 + 4\sigma_t\|\mathbf{g}_t\|} - K_H}{24}\right)\left\|\mathbf{s}_t^C\right\| \\
&\leq \delta_g - \frac{\sqrt{K_H^2 + 4\sigma_t\|\mathbf{g}_t\|} - K_H}{48}\left\|\mathbf{s}_t^C\right\| \\
&\leq \frac{\sqrt{K_H^2 + 8L_F\|\mathbf{g}_t\|} - K_H}{192L_F} - \frac{\sqrt{K_H^2 + 4\sigma_t\|\mathbf{g}_t\|} - K_H}{96\sigma_t} \\
&\leq 0.
\end{aligned}$$

The last inequality follows from the fact that $p(x) := \left(\sqrt{a^2 + x} - a\right)^2/x$ is an increasing function over $\mathbb{R}_+$. Then, we have

$$\delta_g\|\mathbf{s}_t\| + \frac{1}{2}\delta_H\|\mathbf{s}_t\|^2 + \left(\frac{1}{2}L_F - \frac{\sigma_t}{3}\right)\|\mathbf{s}_t\|^3 = \|\mathbf{s}_t\|r(\|\mathbf{s}_t\|) \leq 0,$$

which completes the proof.

With the help of the above lemma, we show that iteration t is successful when $\|\mathbf{g}_t\| \geq \epsilon_g$.

Lemma 19. *Suppose Assumptions 1 and 2 and Conditions 5 and 6 hold. Further, suppose at iteration t, we have $\|\mathbf{g}_t\| > \epsilon_g$ and $\sigma_t \geq 2L_F$. Then, the iteration t is successful, i.e. $\sigma_{t+1} = \sigma_t/\gamma$.*

Proof. First, since $\|\mathbf{g}_t\| \geq \epsilon_g$, by Lemma 12 and Lemma 18, we have

$$F(\mathbf{x}_t + \mathbf{s}_t) - F(\mathbf{x}_t) - m_t(\mathbf{s}_t) \leq \delta_g \left\|\mathbf{s}_t^C\right\| + \frac{1}{2}\epsilon_H \left\|\mathbf{s}_t^C\right\|^2.$$

Now from Condition 6 and (A.5a), we get

$$-m_t(\mathbf{s}_t) \geq -m_t(\mathbf{s}_t^C) \geq \frac{1}{12}\|\mathbf{s}_t^C\|^2 \left(\sqrt{K_H^2 + 4\sigma_t\|\mathbf{g}_t\|} - K_H\right).$$

Consider the approximation quality ρ_t,

$$\begin{aligned}
1 - \rho_t = \frac{F(\mathbf{x}_t + \mathbf{s}_t) - F(\mathbf{x}_t) - m_t(\mathbf{s}_t)}{-m_t(\mathbf{s}_t)} &\leq \frac{\delta_g \left\|\mathbf{s}_t^C\right\| + \frac{1}{2}\delta_H \left\|\mathbf{s}_t^C\right\|^2}{\frac{1}{12}\|\mathbf{s}_t^C\|^2(\sqrt{K_H^2 + 4\sigma_t\|\mathbf{g}_t\|} - K_H)} \\
&= \frac{12\delta_g}{\|\mathbf{s}_t^C\|(\sqrt{K_H^2 + 4\sigma_t\|\mathbf{g}_t\|} - K_H)} + \frac{6\delta_H}{\sqrt{K_H^2 + 4\sigma_t\|\mathbf{g}_t\|} - K_H} \\
&\leq \frac{24\sigma_t\delta_g}{(\sqrt{K_H^2 + 4\sigma_t\|\mathbf{g}_t\|} - K_H)^2} + \frac{6\delta_H}{\sqrt{K_H^2 + 4\sigma_t\|\mathbf{g}_t\|} - K_H} \\
&\leq \frac{48L_F\delta_g}{(\sqrt{K_H^2 + 8L_F\|\mathbf{g}_t\|} - K_H)^2} + \frac{6\delta_H}{\sqrt{K_H^2 + 8L_F\|\mathbf{g}_t\|} - K_H},
\end{aligned}$$

where the second inequality follows from (A.3) and the last inequality follows from $\sigma_t \geq 2L_F$ as well as the fact that function $r(x) := x/\left(\sqrt{a^2+x} - a\right)^2$ is monotonically decreasing over $\mathbb{R}_+$. Now, Since $\delta_H \leq (1-\eta)\left(\sqrt{K_H^2 + 4L_F\|\mathbf{g}_t\|} - K_H\right)/24$, we get
$6\delta_H/\left(\sqrt{K_H^2 + 8L_F\|\mathbf{g}_t\|} - K_H\right) \leq (1-\eta)/4$.
Similarly, since $\delta_g \leq (1-\eta)\left(\sqrt{K_H^2 + 8L_F\|\mathbf{g}_t\|} - K_H\right)^2/(192L_F)$,
we get $48L_F\delta_g/\left(\sqrt{K_H^2 + 8L_F\|\mathbf{g}_t\|} - K_H\right)^2 \leq (1-\eta)/4$. Therefore, $1 - \rho_t \leq 1 - \eta$, which means the iteration is successful. ∎

Now, as in Lemma 16, we have

Lemma 20. *Under Assumptions 1 and 2 and Conditions 5 and 6, we have $\sigma_t \leq 2\gamma L_F$ for all t.*

We are now in position to prove the optimal complexity of Algorithm 2 under Condition 6. Recall that Lemma 16 still holds. Hence, we only need to prove a tighter bound for $|\mathcal{T}_{\text{succ}}|$. In particular, we separate $\mathscr{T}_{\text{succ}}$ into the following three subsets:

$$\mathscr{T}^1_{\text{succ}} \triangleq \{t \in \mathscr{T}_{\text{succ}} \mid \|\mathbf{g}_{t+1}\| \geq \epsilon_g\}\,, \tag{A.9}$$
$$\mathscr{T}^2_{\text{succ}} \triangleq \{t \in \mathscr{T}_{\text{succ}} \mid \|\mathbf{g}_{t+1}\| \leq \epsilon_g \text{ and } \lambda_{\min}(\mathbf{H}_{t+1}) \leq -\epsilon_H\}\,, \tag{A.10}$$
$$\mathscr{T}^3_{\text{succ}} \triangleq \{t \in \mathscr{T}_{\text{succ}} \mid \|\mathbf{g}_{t+1}\| \leq \epsilon_g \text{ and } \lambda_{\min}(\mathbf{H}_{t+1}) \geq -\epsilon_H\}\,. \tag{A.11}$$

Clearly, $\mathscr{T}_{\text{succ}} = \mathscr{T}^1_{\text{succ}} \bigcup \mathscr{T}^2_{\text{succ}} \bigcup \mathscr{T}^3_{\text{succ}}$, and, trivially, $|\mathscr{T}^3_{\text{succ}}| = 1$.

First, let us bound $\mathscr{T}^2_{\text{succ}}$.

Lemma 21. *Under Assumptions 1 and 2 and Conditions 5 and 6, we have the following upper bound,*

$$\left|\mathscr{T}^2_{succ}\right| \leq C\epsilon_H^{-3}.$$

Proof. Since $F(\mathbf{x}_t)$ is monotonically decreasing, then

$$\begin{aligned}
F(\mathbf{x}_0) - F_{\min} &\geq \sum_{t=0}^{T-1} F(\mathbf{x}_t) - F(\mathbf{x}_{t+1}) = F(\mathbf{x}_0) - F(\mathbf{x}_1) + \sum_{t=0}^{T-1} F(\mathbf{x}_t) - F(\mathbf{x}_{t+1}) \\
&\geq F(\mathbf{x}_0) - F(\mathbf{x}_1) + \sum_{t\in\mathscr{T}^2_{\text{succ}}} F(\mathbf{x}_t) - F(\mathbf{x}_{t+1}) \\
&\geq F(\mathbf{x}_0) - F(\mathbf{x}_1) + \sum_{t\in\mathscr{T}^2_{\text{succ}}} \eta m_{t+1}(\mathbf{s}_{t+1}) \\
&\geq F(\mathbf{x}_0) - F(\mathbf{x}_1) + \eta \sum_{t\in\mathscr{T}^2_{\text{succ}}} \frac{\nu^3\epsilon_H^3}{24\gamma^2 L_F^2},
\end{aligned}$$

where the last inequality follows from (A.6b). Hence,

$$\left|\mathscr{T}^2_{\text{succ}}\right| \leq \frac{(F(\mathbf{x}_1) - F_{\min})24\gamma^2 L_F^2}{\eta\nu^3}\epsilon_H^{-3} = \mathcal{O}(\epsilon_H^{-3}).$$

∎

Intuitively, we could see that we need each update to yield sufficient descent in order to bound $\mathscr{T}^1_{\text{succ}}$. Equivalently, we need each $\mathbf{s}_t$ to be bounded below to get sufficient decrease; see the following lemma.

Lemma 22. *Suppose Assumptions 1 and 2 and Conditions 5 and 6 hold. If iteration t is successful and $\|\mathbf{g}_t\| \geq \epsilon_g$, then*

$$\|\boldsymbol{s}_t\| \geq \kappa_g \left[\left(1 - \zeta - \frac{\zeta}{1-2\zeta}\right)\|\mathbf{g}_{t+1}\| - \frac{5}{2}\delta_g\right],$$

where

$$\kappa_g := \min\left\{\left(\frac{L_F}{2} + 2\gamma L_F + \epsilon_0 + \zeta K_F\right)^{-1}, \left(\frac{L_F}{2} + 2\gamma L_F + \frac{\zeta}{1-2\zeta}K_F + \zeta K_F\right)^{-1}\right\}.$$

Proof. Using Condition 6, we get

$$\|\mathbf{g}_{t+1}\| \le \|\mathbf{g}_{t+1} - \nabla m_t(\mathbf{s}_t)\| + \|\nabla m_t(\mathbf{s}_t)\| \le \|\mathbf{g}_{t+1} - \nabla m_t(\mathbf{s}_t)\| + \theta_t\|\mathbf{g}_t\|. \tag{A.12}$$

Noting that $\nabla m_t(\mathbf{s}_t) = \mathbf{g}_t + \mathbf{H}_t\mathbf{s}_t + \sigma_t\|\mathbf{s}_t\|\mathbf{s}_t$, by Assumptions 1 and 2, we have

$$\begin{aligned}\|\mathbf{g}_{t+1} - \nabla m_t(\mathbf{s}_t)\| &\le \|\mathbf{g}_{t+1} - \mathbf{g}_t - \mathbf{H}_t\mathbf{s}_t\| + \sigma_t\|\mathbf{s}_t\|^2\\ &\le \|\int_0^1 (\nabla^2 F(\mathbf{x}_t + \tau\mathbf{s}_t) - \nabla^2 F(\mathbf{x}_t))\mathbf{s}_t d\tau + (\nabla^2 F(\mathbf{x}_t) - \mathbf{H}_t)\mathbf{s}_t\|\\ &\quad + \|\mathbf{g}_t - \nabla F(\mathbf{x}_t)\| + \|\mathbf{g}_{t+1} - \nabla F(\mathbf{x}_t + \tau\mathbf{s}_t)\| + \sigma_t\|\mathbf{s}_t\|^2\\ &\le \left(\frac{L_F}{2} + 2\gamma L_F\right)\|\mathbf{s}_t\|^2 + \delta_H\|\mathbf{s}_t\| + 2\delta_g. \end{aligned}\tag{A.13}$$

Also according to Assumption 2, we get

$$\begin{aligned}\|\mathbf{g}_t\| \le \|\mathbf{g}_t - \nabla F(\mathbf{x}_t)\| + \|\nabla F(\mathbf{x}_t))\| &\le \delta_g + K_H\|\mathbf{s}_t\| + \|\nabla F(\mathbf{x}_t + \mathbf{s}_t)\|\\ &\le 2\delta_g + K_H\|\mathbf{s}_t\| + \|\mathbf{g}_{t+1}\|. \end{aligned}\tag{A.14}$$

By combining (A.12)–(A.14) and using definition of θ_t in (2.13), we get

$$\begin{aligned}\|\mathbf{g}_{t+1}\| &\le \left(\frac{L_F}{2} + 2\gamma L_F\right)\|\mathbf{s}_t\|^2 + (\delta_H + \theta_t K_F)\|\mathbf{s}_t\| + 2(1+\theta_t)\delta_g + \theta_t\|\mathbf{g}_{t+1}\|\\ &\le \left(\frac{L_F}{2} + 2\gamma L_F\right)\|\mathbf{s}_t\|^2 + (\delta_H + \theta_t K_F)\|\mathbf{s}_t\| + \frac{5}{2}\delta_g + \zeta\|\mathbf{g}_{t+1}\|,\end{aligned}$$

which implies

$$(1-\zeta)\|\mathbf{g}_{t+1}\| - \frac{5}{2}\delta_g \le \left(\frac{L_F}{2} + 2\gamma L_F\right)\|\mathbf{s}_t\|^2 + (\delta_H + \theta_t K_F)\|\mathbf{s}_t\|. \tag{A.15}$$

Now, consider two cases:

i. If $\|\mathbf{s}_t\| \ge 1$, then

$$(\delta_H + \theta_t K_F)\|\mathbf{s}_t\| \le (\epsilon_H + \zeta K_F)\|\mathbf{s}_t\|^2.$$

It follows,

$$(1-\zeta)\|\mathbf{g}_{t+1}\| - 5/2\delta_g \le \left(\frac{L_F}{2} + 2\gamma L_F + \epsilon_H + \zeta K_F\right)\|\mathbf{s}_t\|^2,$$

i.e.

$$\|\mathbf{s}_t^2\| \ge \frac{(1-\zeta)\|\mathbf{g}_{t+1}\| - \frac{5}{2}\delta_g}{L_F/2 + 2\gamma L_F + \epsilon_H + \zeta K_F}.$$

ii. If $\|\mathbf{s}_t\| \leq 1$, then

$$\begin{aligned}\delta_H &\leq \zeta\epsilon_g \leq \zeta\|\mathbf{g}_t\| \\ &\leq \zeta\Big(\|\mathbf{g}_{t+1}\| + \|\nabla F(\mathbf{x}_t+\mathbf{s}_t) - \mathbf{g}_{t+1}\| + \|\nabla F(\mathbf{x}_t) - \nabla F(\mathbf{x}_t+\mathbf{s}_t)\| + \|\mathbf{g}_t - \nabla F(\mathbf{x}_t)\|\Big) \\ &\leq \zeta\left(2\delta_g + K_F\|\mathbf{s}_t\| + \|\mathbf{g}_{t+1}\|\right) \\ &\leq \zeta\left(2\delta_H + K_F\|\mathbf{s}_t\| + \|\mathbf{g}_{t+1}\|\right),\end{aligned}$$

where the third inequality is from triangular inequality and the last inequality follows from $\delta_g \leq \delta_H$ in (2.12c) in Condition 6. Therefore we have

$$\delta_H\|\mathbf{s}_t\| \leq \frac{\zeta}{1-2\zeta}\left(K_F\|\mathbf{s}_t\| + \|\mathbf{g}_{t+1}\|\right)\|\mathbf{s}_t\| \leq \frac{\zeta}{1-2\zeta}\left(K_F\|\mathbf{s}_t\|^2 + \|\mathbf{g}_{t+1}\|\right).$$

Then, using $\theta_t \leq \zeta$ in (2.13),

$$(\delta_H + \theta_t K_F)\|\mathbf{s}_t\| \leq \left(\frac{\zeta}{1-2\zeta} + \zeta\right)K_F\|\mathbf{s}_t\|^2 + \frac{\zeta}{1-2\zeta}\|\mathbf{g}_{t+1}\|.$$

Substituting this into (A.15), we have

$$\left(1 - \zeta - \frac{\zeta}{1-2\zeta}\right)\|\mathbf{g}_{t+1}\| - \frac{5}{2}\delta_g \leq \left(\frac{L_F}{2} + 2\gamma L_F + \frac{\zeta}{1-2\zeta}K_F + \zeta K_F\right)\|\mathbf{s}_t\|^2,$$

i.e.

$$\|\mathbf{s}_t\|^2 \geq \left(\left(1 - \zeta - \frac{\zeta}{1-2\zeta}\right)\|\mathbf{g}_{t+1}\| - \frac{5}{2}\delta_g\right)\left(\frac{L_F}{2} + 2\gamma L_F + \frac{\zeta}{1-2\zeta}K_F + \zeta K_F\right)^{-1}.$$

The two cases complete the proof.

Now, based on Lemma 22, it is easy to bound $|\mathscr{T}^1_{\text{succ}}|$.

Lemma 23. *Given the same setting as Lemma 22, the success iterations $\mathscr{T}^1_{succ}$ is bounded by*

$$\left|\mathscr{T}^1_{succ}\right| \leq C\max\left\{\epsilon_g^{-1.5}, \epsilon_H^{-3}\right\}.$$

Proof. First, according to (2.12a) in Condition 5, we have

$$\begin{aligned}\delta_g &\leq \frac{1-\eta}{192L_F}\left(\sqrt{K_H^2 + 8L_F\max\{\min\{\|\mathbf{g}_t\|, \|\mathbf{g}_{t+1}\|\}, \epsilon_g\}} - K_H\right)^2 \\ &\leq \frac{(1-\eta)8L_F\max\{\min\{\|\mathbf{g}_t\|, \|\mathbf{g}_{t+1}\|\}, \epsilon_g\}}{192L_F} \\ &\leq \frac{\max\{\min\{\|\mathbf{g}_t\|, \|\mathbf{g}_{t+1}\|\}, \epsilon_g\}}{24}.\end{aligned}$$

If $\|\mathbf{g}_{t+1}\| \geq \epsilon_g$ and $\|\mathbf{g}_t\| \geq \epsilon_g$, according to Lemma 22 and substituting $\zeta = 1/4$, we have

$$\|\mathbf{s}_t\|^2 \geq \kappa_g \left[(1 - 1/4 - \frac{1/4}{1-2/4})\epsilon_g - 5/2\frac{1}{24}\epsilon_g \right] = \frac{1}{8}\kappa_g\epsilon_g.$$

Now consider any $t \in \mathscr{T}^1_{\text{succ}}$. Since $\|\mathbf{g}_t\| \geq \epsilon_g$, then we have

$$-m_t(\mathbf{s}_t) \geq \frac{\sigma_t}{6}\|\mathbf{s}_t\|^3 \geq \frac{\sigma_{\min}}{6}(\frac{\kappa_g\epsilon_g}{8})^{3/2} \geq c_g\epsilon_g^{3/2},$$

where $c_g \triangleq \kappa_g^{3/2}\sigma_{\min}/200$. Otherwise, we must have $\lambda_{\min}(\mathbf{H}_t) \leq -\epsilon_H$, and by (A.6b), we have

$$-m_t(\mathbf{s}_t) \geq \frac{\nu^3\epsilon_H^3}{24\gamma^2 L_F^2} = c_H\epsilon_H^3,$$

where $c_H \triangleq \frac{\nu^3}{24\gamma^2 L_F^2}$. Therefore,

$$-m_t(\mathbf{s}_t) \geq \min\{c_g\epsilon_g^{3/2}, c_H\epsilon_H^3\}.$$

Since $F(\mathbf{x}_t)$ is monotonically decreasing and $F(\mathbf{x})$ is lower bounded by $F_{\min}$, it follows that

$$\begin{aligned} F(\mathbf{x}_0) - F_{\min} &\geq \sum_{t=0}^{T-1} F(\mathbf{x}_t) - F(\mathbf{x}_{t+1}) \geq \sum_{t\in\mathscr{T}^1_{\text{succ}}} F(\mathbf{x}_t) - F(\mathbf{x}_{t+1}) \geq \sum_{t\in\mathscr{T}^1_{\text{succ}}} -\eta m_t(\mathbf{s}_t) \\ &\geq \sum_{t\in\mathscr{T}^1_{\text{succ}}} \min\{c_g\epsilon_g^{3/2}, c_H\epsilon_H^3\} = \left|\mathscr{T}^1_{\text{succ}}\right| \min\{c_g\epsilon_g^{3/2}, c_H\epsilon_H^3\}. \end{aligned}$$

Therefore

$$\left|\mathscr{T}^1_{\text{succ}}\right| \leq \max\left\{\frac{F(\mathbf{x}_0) - F_{\min}}{c_g}\epsilon_g^{-3/2}, \frac{F(\mathbf{x}_0) - F_{\min}}{c_H}\epsilon_H^{-3}\right\},$$

which completes the proof. ∎

Since $\mathscr{T}_{\text{succ}} = \mathscr{T}^1_{\text{succ}} \bigcup \mathscr{T}^2_{\text{succ}} \bigcup \mathscr{T}^3_{\text{succ}}$, we can get a bound of the total number of successful iterations.

Lemma 24. *Given Assumptions 1 and 2 as well as Conditions 5 and 6, the number of successful iterations $|\mathscr{T}_{succ}|$ is bounded by*

$$|\mathscr{T}_{succ}| \leq C \max\left\{\epsilon_g^{-1.5}, \epsilon_H^{-3}\right\}.$$

Proof. It immediately follows from Lemma 21 and Lemma 23.

Appendix B

Appendix for Chapter 3

B.1 Key Ingredients of the Newton-CG Method

We present the two major components from [199] that are also used in our inexact variant of the Newton-CG algorithm. The first ingredient, Procedure 8 (referred to in some places as "Capped CG"), is a version of the conjugate gradient [216] algorithm that is used to solve a damped Newton system of the form $\bar{\mathbf{H}}\mathbf{d} = \mathbf{g}$, where $\bar{\mathbf{H}} = \mathbf{H} + 2\epsilon\mathbf{I}$ for some positive parameter ϵ. Procedure 8 is modified to detect indefiniteness in the matrix $\mathbf{H}$ and, when this occurs, to return a direction along which the curvature of $\mathbf{H}$ at most $-\epsilon$. The second ingredient, Procedure 9 (referred to as the "Minimum Eigenvalue Oracle" or "MEO"), checks whether a direction of negative curvature (less than $-\epsilon$ for a given positive argument ϵ) exists for the given matrix $\mathbf{H}$ We now discuss each of these procedures in more detail.

Procedure 8 (Capped-CG). The well-known classical CG algorithm [216] is used to solve linear systems involving positive definite matrices. However, this positive definite requirement is often violated during the iterations for non-convex optimization due to indefiniteness of Hessians encountered at some iterates. Capped-CG, proposed by [199] and presented in Procedure 8 for completeness, is an original way to leverage and detect such negative curvature directions, when they are encountered during CG iterations.

Lines 13-17 in Procedure 8 contain the standard CG operations. When $\mathbf{H} \succeq -\epsilon\mathbf{I}$, the tests for negative curvature in lines 22, 26, and 28 will not be activated, and Capped-CG will return an approximate solution $\mathbf{d} \approx \bar{\mathbf{H}}^{-1}\mathbf{g}$. However, when $\mathbf{H} \not\succeq -\epsilon\mathbf{I}$, Capped-CG will identify and return a direction of "sufficient negative curvature" — a direction d satisfying $\mathbf{d}^T\mathbf{H}\mathbf{d} \leq -\epsilon\|\mathbf{d}\|^2$. This negative curvature direction is obtained under two circumstances. First, when the intermediate step (either $\mathbf{y}_j$ or $\mathbf{p}_j$) satisfies the negative curvature condition, i.e., $\mathbf{d}^T\bar{\mathbf{H}}\mathbf{d} \leq -\epsilon\|\mathbf{d}\|^2$ (Lines 22 and 26), Procedure 8 will be terminated and the intermediate step will be returned. Second, when the residual, $\mathbf{r}_j$, decays at a slower rate than anticipated by standard CG analysis (Line 28) (under an assumption that the eigenvalues of $\bar{\mathbf{H}}$ are bounded below by ϵ, a negative curvature can also be recovered. Note that Procedure 8 can be called with an optional input M, which is an upper bound on $\|\mathbf{H}\|$. However, even

Procedure 8 Capped Conjugate Gradient

```
1: Inputs: Symmetric Matrix $\mathbf{H} \in \mathbf{R}^{d\times d}$, vector $\mathbf{g} \neq 0$; damping parameter $\epsilon \in (0,1)$; ; desired accuracy $\zeta \in (0,1)$;
2: Optional input: positive scale $M$ (set to 0 if not provided)
3: Outputs: $d_{\text{type}}$, $\mathbf{d}$
4: Secondary Output: $M$, $\kappa$, $\tilde{\zeta}$, $\tau$, $T$
5: Set
   $$\bar{\mathbf{H}} := \mathbf{H} + 2\epsilon, \kappa := \frac{M+2\epsilon}{\epsilon}, \tilde{\zeta} := \frac{\zeta}{3\kappa}, T := \frac{4\kappa^4}{(1-\sqrt{1-\tau})^2}, \tau := \frac{1}{\sqrt{\kappa}+1};$$
6: $\mathbf{y}_0 \leftarrow 0, \mathbf{r}_0 \leftarrow \mathbf{g}, \mathbf{p}_0 \leftarrow -\mathbf{g}, j \leftarrow 0$
7: if $\mathbf{p}_0^T \bar{\mathbf{H}} \mathbf{p}_0 < \epsilon \|\mathbf{p}_0\|^2$ then
8:     Set $\mathbf{d} = p_0$ and terminate with $d_{\text{type}} = \text{NC}$;
9: else if $\|\mathbf{H}\mathbf{p}_0\| > M\|\mathbf{p}_0\|$ then
10:    $M \leftarrow \|\mathbf{H}\mathbf{p}_0\|/\|\mathbf{p}_0\|$ and update $\kappa$, $\tilde{\zeta}$, $\tau$, $T$;
11: end if
12: while TRUE do
13:    $\alpha_j \leftarrow \mathbf{r}_j^T\mathbf{r}_j/\mathbf{p}_j^T\bar{\mathbf{H}}\mathbf{p}_j$; (Traditional CG Begins)
14:    $\mathbf{y}_{j+1} \leftarrow \mathbf{y}_j + \alpha_j \mathbf{p}_i$;
15:    $\mathbf{r}_{j+1} \leftarrow \mathbf{r}_j + \alpha_j \bar{\mathbf{H}}\mathbf{p}_j$;
16:    $\beta_{j+1} \leftarrow \mathbf{r}_{j+1}^T\mathbf{r}_{j+1}/\mathbf{r}_j^T\mathbf{r}_j$;
17:    $\mathbf{p}_{j+1} \leftarrow -\mathbf{r}_{j+1} + \beta_{j+1}\mathbf{p}_j$; (Traditional CG Ends)
18:    $j \leftarrow j+1$;
19:    if $\max(\|\mathbf{H}\mathbf{p}_j\|/\|\mathbf{p}_j\|, \|\mathbf{H}\mathbf{y}_j\|/\|\mathbf{y}_j\|, \|\mathbf{H}\mathbf{r}_j\|/\|\mathbf{r}_j\|) > M$ then
20:        $M \leftarrow \max(\|\mathbf{H}\mathbf{p}_j\|/\|\mathbf{p}_j\|, \|\mathbf{H}\mathbf{y}_j\|/\|\mathbf{y}_j\|, \|\mathbf{H}\mathbf{r}_j\|/\|\mathbf{r}_j\|)$ and update $\kappa$, $\tilde{\zeta}$, $\tau$, $T$;
21:    end if
22:    if $\mathbf{y}_j^T\bar{\mathbf{H}}\mathbf{y}_j \leq \epsilon\|\mathbf{y}_j\|^2$ then
23:        Set $\mathbf{d} \leftarrow \mathbf{y}_j$ and terminate with $d_{\text{type}} = \text{NC}$;
24:    else if $\|\mathbf{r}_j\| \leq \hat{\zeta}\|\mathbf{r}_0\|$ then
25:        Set $\mathbf{d} \leftarrow \mathbf{y}_j$ and terminate with $d_{\text{type}} = \text{SOL}$;
26:    else if $\mathbf{p}_j^T\bar{\mathbf{H}}\mathbf{p}_j \leq \epsilon\|\mathbf{p}_j\|^2$ then
27:        Set $\mathbf{d} \leftarrow \mathbf{p}_j$ and terminate with $d_{\text{type}} = \text{NC}$;
28:    else if $\|\mathbf{r}_j\| \geq \sqrt{T}(1-\tau)^{j/2}\|\mathbf{r}_0\|$ then
29:        Compute $\alpha_j, \mathbf{p}_{j+1}$ as in the main loop above;
30:        Find $i \in \{0, \cdots, j-1\}$ such that
           $$\frac{(\mathbf{y}_{j+1}-\mathbf{y}_i)^T\bar{\mathbf{H}}(\mathbf{y}_{j+1}-\mathbf{y}_i)}{\|\mathbf{y}_{j+1}-\mathbf{y}_i\|^2} \leq \epsilon; \tag{B.1}$$
31:        Set $\mathbf{d} \leftarrow \mathbf{y}_{j+1} - \mathbf{y}_i$ and terminate with $d_{\text{type}} = \text{NC}$;
32:    end if
33: end while
34: Return: d
```

without a priori knowledge of this upper bound, M can be updated so that at any point in the execution of the procedure, M is an upper bound on the maximum curvature of **H** revealed to that point. Other parameters (κ, $\tilde{\zeta}$, τ, T) are also updated whenever the value of M changes. It is not hard to see that M is bounded by $U_{\mathbf{H}}$ throughout the execution of Procedure 8, provided that if the value of M is supplied to this procedure, the supplied value is at most $U_{\mathbf{H}}$.

Lemma 25 gives a bound on the number of iterations performed by Procedure 8.

Lemma 25 ([199, Lemma 1]). *The number of iterations of Procedure 8 is bounded by*

$$\min\{d, J(M,\epsilon,\zeta)\},$$

where $J = J(M,\epsilon,\zeta)$ is the smallest integer such that $\sqrt{T}(1-\tau)^{J/2} \le \hat{\zeta}$. The number of matrix-vector products required is bounded by $2\min\{d, J(M,\epsilon,\zeta)\}+1$, unless all iterates $\boldsymbol{y}_i$, $i = 1,2,\ldots$ are stored, in which case it is $\min\{d, J(M,\epsilon,\zeta)\}+1$. We can estimate the upper bound of $J(M,\epsilon,\zeta)$ as

$$J(M,\epsilon,\zeta) \le \min\left\{d, \tilde{\mathcal{O}}(\epsilon^{-1/2})\right\}. \tag{B.2}$$

When the slow decrease in residual is detected (Line 21), a direction of negative curvature for $\mathbf{H}$ can be extracted from the previous intermediate solutions, as the following result describes.

Lemma 26 ([199, Theorem 2]). *Suppose that the loop of Procedure 8 terminates with $j = \hat{J}$, where*

$$\hat{J} \in \{1,2,\ldots,\min\{n, J(M,\epsilon,\zeta)\}\}$$

satisfies

$$\|r_{\hat{J}}\| > \max\{\hat{\zeta}, \sqrt{T}(1-\tau)^{\hat{J}/2}\}\|r_0\|.$$

Suppose further that $y_{\hat{J}}^T\bar{\mathbf{H}}y_{\hat{J}} \ge \epsilon\|y_{\hat{J}}\|^2$, so that $y_{\hat{J}+1}$ is computed. Then we have

$$\frac{(y_{\hat{J}+1}-y_i)^T\bar{\mathbf{H}}(y_{\hat{J}+1}-y_i)}{\|y_{\hat{J}+1}-y_i\|^2} < \epsilon, \qquad for \quad some \quad i \in \{0,\ldots,\hat{J}-1\}.$$

Note that $d^T\bar{\mathbf{H}}d \le \epsilon\|d\|^2 \iff d^T\mathbf{H}d \le -\epsilon\|d\|^2$.

Procedure 8 is invoked by the Newton-CG procedure, Algorithm 3, when the current iterate $\mathbf{x}_k$ has $\|\nabla f(\mathbf{x}_k)\| = \|\mathbf{g}_k\| > \epsilon_g > 0$. The output $\mathbf{d}$ of Procedure 8 may be scaled according to the type of the result, in the manner described in the following result, which uses Lemmas 25 and 26 to summarize the outputs of Procedure 8.

Lemma 27 ([199, Lemma 3]). *Let Assumption 3, and Condition 7 hold. Suppose that Procedure 8 is invoked at an iterate $\boldsymbol{x}_k$ of Algorithm 3 (so that $\|\mathbf{g}_k\| > \epsilon_g > 0$) with inputs $\mathbf{H} = \mathbf{H}_k$, $\mathbf{g} = \mathbf{g}_k$, $\epsilon = \epsilon_H$, and ζ. Let $\mathbf{d}$ be the output of Procedure 8, which is subsequently scaled in Algorithm 3 to obtain $\mathbf{d}_k$. Then one of the two following statements holds:*

i. *$d_{\text{type}} = \text{SOL}$ and $\mathbf{d}_k = \mathbf{d}$ satisfies*

$$\mathbf{d}_k^T(\mathbf{H}_k + 2\epsilon_H \boldsymbol{I})\mathbf{d}_k \geq \epsilon_H \|\mathbf{d}_k\|^2, \tag{B.3a}$$

$$\|\mathbf{d}_k\| \leq \epsilon_H^{-1}\sqrt{1+\zeta^2/4}\|\mathbf{g}_k\|, \tag{B.3b}$$

$$\|\hat{\mathbf{r}}_k\| \leq \frac{1}{2}\epsilon_H \zeta \|\mathbf{d}_k\|, \tag{B.3c}$$

where

$$\hat{\mathbf{r}}_k = (\mathbf{H}_k + 2\epsilon_H \boldsymbol{I})\mathbf{d}_k + \mathbf{g}_k. \tag{B.4}$$

ii. *$d_{\text{type}} = \text{NC}$ and $\mathbf{d}_k$ satisfies*

$$\mathbf{d}_k = -\text{sgn}(\mathbf{d}^T\mathbf{g}_k)\frac{|\mathbf{d}^T\mathbf{H}_k\mathbf{d}|}{\|\mathbf{d}\|^2}\frac{\mathbf{d}}{\|\mathbf{d}\|},$$

and $\mathbf{d}_k$ satisfies

$$\mathbf{d}_k^T\mathbf{H}_k\mathbf{d}_k \leq -\epsilon_H \|\mathbf{d}_k\|^2, \tag{B.5a}$$

$$\|\mathbf{d}_k\| \geq \epsilon_H. \tag{B.5b}$$

Procedure 8 can either return the approximate Newton direction or a direction computed from the negative curvature. In the case of $\|\mathbf{g}_k\| < \epsilon_g$, however, the Capped-CG algorithm is not invoked by Algorithm 3, which resorts instead to Procedure 9, described next.

Procedure 9 Minimum Eigenvalue Oracle

```
1: Inputs: Symmetric matrix $\mathbf{H} \in \mathbf{R}^{d\times d}$, scalar $M \geq \lambda_{\max}(\mathbf{H})$ and $\epsilon > 0$;
2: Set $\delta \in [0, 1)$;
3: Outputs: Estimate $\lambda$ of $\lambda_{\min}(\mathbf{H})$ such that $\lambda \leq -\epsilon/2$ and vector $\mathbf{v}$ with $\|\mathbf{v}\| = 1$ such
   that $\mathbf{v}^T\mathbf{H}\mathbf{v} = \lambda$ OR certificate that $\lambda_{\min}(\mathbf{H}) \geq -\epsilon$. (If the certificate is the output, it is
   false with probability $\delta$.)
```

Procedure 9 (Minimum Eigenvalue Oracle). This procedure searches for a direction spanned by the negative spectrum of a given symmetric matrix or, alternately, verifies that the matrix is (almost) positive definite. More specifically, for a given $\epsilon > 0$, Procedure 9 finds a negative curvature direction $\mathbf{v}$ of $\mathbf{H}_k$ such that $\mathbf{v}^T\mathbf{H}\mathbf{v} \leq -\epsilon\|\mathbf{v}\|^2/2$, or else certifies that $\mathbf{H} \succeq -\epsilon\mathbf{I}$. In the latter case, the certification is false with a probability less than some specified value δ. As indicated in [199], this minimum eigenvalue oracle can be implemented using the Lanczos process or the classical CG algorithm. (In this paper, we choose the former.) Both of these approaches have the same complexity, given in the following result.

Lemma 28 ([199, Lemma 2]). *Suppose that the Lanczos method is used to estimate the smallest eigenvalue of* $\mathbf{H}$ *starting from a random vector drawn from the uniform distribution on the unit sphere, where* $\|\mathbf{H}\| \le M$*. For any* $\delta \in (0,1)$*, this approach finds the smallest eigenvalue of* H *to an absolute precision of* $\epsilon/2$*, together with a corresponding direction* $\boldsymbol{v}$*, in at most*

$$\min\left\{d, 1 + \left\lceil \frac{\ln(2.75d/\delta^2)}{2}\sqrt{\frac{M}{\epsilon}} \right\rceil \right\} \quad \textit{iterations,} \tag{B.6}$$

with probability at least $1-\delta$*.*

B.2 Proof of Theorem 4

Given Condition 7, the proofs of the complexity bounds boil down to three parts. First, we bound the decrease in the objective function $f(\mathbf{x}_k)$ (Lemma 29) when taking the damped Newton step $\mathbf{d}_k$ (that is, when $d_{\text{type}} = \text{SOL}$ on return from Procedure 8). Second, we bound the decrease in the objective when a negative curvature direction is encountered in Procedure 8 (Lemma 30) or 9 (Lemma 31). Third, for Lines 10-19 in Algorithm 3, we show that the algorithm can be terminated after the update in Line 13. In particular, when the update direction is sufficiently small from Procedure 8 and a large negative curvature from Procedure 9 has not been detected, Line 13 terminates at a point satisfying the required optimality conditions (Lemma 32).

We start with the case in which an inexact Newton step is used.

Lemma 29. *Suppose that Assumption 3, and Condition 7 hold. Suppose that at iteration* k *of Algorithm 3, we have* $\|\mathbf{g}_k\| > \epsilon_g$*, so that Procedure 8 is called. When Procedure 8 outputs a direction* $\mathbf{d}_k$ *with* $d_{\text{type}} = \text{SOL}$ *and* $\|\mathbf{d}_k\| \ge \epsilon_g/\epsilon_H$ *, then the backtracking line search requires at most* $j_k \le j_{\text{sol}} + 1$ *iterations, where*

$$j_{\text{sol}} = \left\lceil \frac{1}{2}\log_\theta \frac{3(1-\zeta)\epsilon_H^2}{4U_g(L_H+\eta)\sqrt{1+\zeta^2/4}} \right\rceil,$$

and the resulting step $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \mathbf{d}_k$ *satisfies*

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1}) \ge c_{\text{sol}} \max\left\{0, \min\left(\frac{(\|\mathbf{g}_{k+1}\| - \delta_{g,k} - \delta_{g,k+1})^3}{\epsilon_H^3}, \epsilon_H^3, \epsilon_g^{3/2}\right)\right\}, \tag{B.7}$$

where

$$c_{\text{sol}} = \frac{\eta}{6}\min\left\{\left[\frac{1}{(3+\frac{1}{2}\zeta) + \sqrt{(3+\frac{1}{2}\zeta)^2 + 2L_H}}\right]^3, \left[\frac{3\theta^2(1-\zeta)}{4(L_H+\eta)}\right]^{3/2}\right\}.$$

Proof. When the $d_{\text{type}} = \text{SOL}$, the output solution $\mathbf{d}_k$ is the solution of inexact regularized Newton step. We first prove that when $\mathbf{d}_k^T \mathbf{g}_k < 0$, $\mathbf{g}_k^T \nabla f(\mathbf{x}_k)$ is also negative:

$$\begin{aligned}
\mathbf{d}_k^T \nabla f(\mathbf{x}_k) &\leq \mathbf{d}_k^T \mathbf{g}_k + \delta_{g,k}\|\mathbf{d}_k\| \\
&= \mathbf{d}_k^T \hat{r}_k - \mathbf{d}_k^T(\mathbf{H}_k + 2\epsilon_H \mathbf{I})\mathbf{d}_k + \delta_{g,k} g\|\mathbf{d}_k\| \text{ ((B.4))} \\
&\leq \|\mathbf{d}_k^T\|\|\hat{r}_k\| - \epsilon_H\|\mathbf{d}_k\|^2 + \delta_{g,k}\|\mathbf{d}_k\| \text{ ((B.3a))} \\
&\leq -\frac{1}{2}\epsilon_H\|\mathbf{d}_k\|^2 + \frac{1-\zeta}{8}\max\left(\epsilon_g, \epsilon_H\|\mathbf{d}_k\|\right)\|\mathbf{d}_k\| \text{ ((B.3c) and Condition 7)} \qquad \text{(B.8)} \\
&= -\frac{1}{2}\epsilon_H\|\mathbf{d}_k\|^2 + \frac{1-\zeta}{8}\epsilon_H\|\mathbf{d}_k\|^2 \\
&< -\frac{3}{8}\epsilon_H\|\mathbf{d}_k\|^2.
\end{aligned}$$

We consider two cases here. **Case 1:** Consider first $\alpha_k = 1$. We note first that in the case $\|\mathbf{g}_{k+1}\| - \delta_{g,k} - \delta_{g,k+1} \leq 0$, the claim (B.7) is satisfied trivially. Thus we assume in the rest of the argument for this case that $\|\mathbf{g}_{k+1}\| - \delta_{g,k} - \delta_{g,k+1} > 0$. We have

$$\begin{aligned}
\|\mathbf{g}_{k+1}\| &= \|\mathbf{g}_{k+1} - \mathbf{g}_k + \mathbf{g}_k\| \\
&= \|\mathbf{g}_{k+1} - \nabla f_{k+1} + \nabla f_{k+1} - \mathbf{g}_k - \nabla f_k + \nabla f_k - \nabla^2 f(\mathbf{x}_k)\mathbf{d}_k - 2\epsilon_H d_k + \nabla^2 f(\mathbf{x}_k)\mathbf{d}_k - \mathbf{H}_k\mathbf{d}_k + r_k\| \\
&\leq \delta_{g,k} + \delta_{g,k+1} + \|\nabla f_{k+1} - \nabla f_k - \nabla^2 f(\mathbf{x}_k)\mathbf{d}_k\| + \|2\epsilon_H d_k\| + \|\nabla^2 f(\mathbf{x}_k)\mathbf{d}_k - \mathbf{H}_k\mathbf{d}_k\| + \|r_k\| \\
&\leq \delta_{g,k} + \delta_{g,k+1} + \frac{L_H}{2}\|\mathbf{d}_k\|^2 + 2\epsilon_H\|\mathbf{d}_k\| + \delta_H\|\mathbf{d}_k\| + \frac{1}{2}\epsilon_H\zeta\|\mathbf{d}_k\| \qquad \text{((B.3c))} \\
&= \delta_{g,k} + \delta_{g,k+1} + (2\epsilon_H + \delta_H + \frac{1}{2}\epsilon_H\zeta)\|\mathbf{d}_k\| + \frac{L_H}{2}\|\mathbf{d}_k\|^2.
\end{aligned}$$

By rearranging the terms above (see [200, Lemma 17]), we have that

$$\|\mathbf{d}_k\| \geq \frac{1}{(2 + \delta_H/\epsilon_H + \frac{1}{2}\zeta) + \sqrt{(2 + \delta_H/\epsilon_H + \frac{1}{2}\zeta)^2 + 2L_H}}\min\{(\|\mathbf{g}_{k+1}\| - \delta_{g,k} - \delta_{g,k+1})/\epsilon_H, \epsilon_H\}.$$

Since $\alpha_k = 1$ was accepted by the backtracking line search, we have for the case of $\|\mathbf{g}_{k+1}\| - \delta_{g,k} - \delta_{g,k+1} > 0$ that

$$\begin{aligned}
&f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{d}_k) \\
&\geq \frac{\eta}{6}\frac{1}{\left[2 + \delta_H/\epsilon_H + \frac{1}{2}\zeta) + \sqrt{(2 + \delta_H/\epsilon_H + \frac{1}{2}\zeta)^2 + 2L_H}\right]^3}\min\left\{\frac{(\|\mathbf{g}_{k+1}\| - \delta_{g,k} - \delta_{g,k+1})^3}{\epsilon_H^3}, \epsilon_H^3\right\} \\
&\geq \frac{\eta}{6}\frac{1}{\left[3 + \frac{1}{2}\zeta) + \sqrt{(3 + \frac{1}{2}\zeta)^2 + 2L_H}\right]^3}\min\left\{\frac{(\|\mathbf{g}_{k+1}\| - \delta_{g,k} - \delta_{g,k+1})^3}{\epsilon_H^3}, \epsilon_H^3\right\},
\end{aligned}$$

where the last inequality follows from $\delta_H \leq \epsilon_H$. By combining this inequality with the trivial inequality obtained when $\|\mathbf{g}_{k+1}\| - \delta_{g,k} - \delta_{g,k+1} \leq 0$, we obtain (B.7) for the case of $\alpha_k = 1$.

Case 2: When $\alpha_k < 1$. In this case, we know that there is $j \geq 0$ such that the acceptance condition for the backtracking line search is not satisfied. For this j, we have

$$
\begin{aligned}
-\frac{\eta}{6}\theta^{3j}\|\mathbf{d}_k\|^3 &\leq f(\mathbf{x}_k + \theta^j \mathbf{d}_k) - f(\mathbf{x}_k) \\
&\leq \theta^j \nabla f_k^T \mathbf{d}_k + \frac{\theta^{2j}}{2}\mathbf{d}_k^T \nabla^2 f(\mathbf{x}_k)\mathbf{d}_k + \frac{L_H}{6}\theta^{3j}\|\mathbf{d}_k\|^3 \\
&= \theta^j \mathbf{g}_k^T \mathbf{d}_k + \frac{\theta^{2j}}{2}\mathbf{d}_k^T \mathbf{H}_k \mathbf{d}_k + \theta^j(\nabla f(\mathbf{x}_k) - \mathbf{g}_k)^T \mathbf{d}_k + \frac{\theta^{2j}}{2}\mathbf{d}_k^T(\nabla^2 f(\mathbf{x}_k) - \mathbf{H}_k)\mathbf{d}_k + \frac{L_H}{6}\theta^{3j}\|\mathbf{d}_k\|^3 \\
&\leq \theta^j \mathbf{g}_k^T \mathbf{d}_k + \frac{\theta^{2j}}{2}\mathbf{d}_k^T \mathbf{H}_k \mathbf{d}_k + \theta^j \delta_{g,k}\|\mathbf{d}_k\| + \frac{\theta^{2j}}{2}\delta_H\|\mathbf{d}_k\|^2 + \frac{L_H}{6}\theta^{3j}\|\mathbf{d}_k\|^3 \\
&= \theta^j(\hat{r}_k - (\mathbf{H}_k + 2\epsilon_H \mathbf{I})\mathbf{d}_k)^T \mathbf{d}_k + \frac{\theta^{2j}}{2}\mathbf{d}_k^T \mathbf{H}_k \mathbf{d}_k + \theta^j \delta_{g,k}\|\mathbf{d}_k\| \\
&+ \frac{\theta^{2j}}{2}\delta_H\|\mathbf{d}_k\|^2 + \frac{L_H}{6}\theta^{3j}\|\mathbf{d}_k\|^3 \text{ ((B.4))} \\
&= -\theta^j(1 - \frac{\theta^j}{2})\mathbf{d}_k^T(\mathbf{H}_k + 2\epsilon_H \mathbf{I})\mathbf{d}_k - \theta^{2j}\epsilon_H\|\mathbf{d}_k\|^2 + \theta^j \hat{r}_k^T \mathbf{d}_k \\
&+ \theta^j \delta_{g,k}\|\mathbf{d}_k\| + \frac{\theta^{2j}}{2}\delta_H\|\mathbf{d}_k\|^2 + \frac{L_H}{6}\theta^{3j}\|\mathbf{d}_k\|^3 \\
&\leq -\frac{\theta^j}{2}(1-\zeta)\epsilon_H\|\mathbf{d}_k\|^2 + \theta^j \delta_{g,k}\|\mathbf{d}_k\| + \frac{\theta^{2j}}{2}\delta_H\|\mathbf{d}_k\|^2 + \frac{L_H}{6}\theta^{3j}\|\mathbf{d}_k\|^3 \\
&\leq \theta^j \delta_{g,k}\|\mathbf{d}_k\| - \frac{\theta^j}{2}\|\mathbf{d}_k\|^2\big((1-\zeta)\epsilon_H - \delta_H\big) + \frac{L_H}{6}\theta^{3j}\|\mathbf{d}_k\|^3 \ (\theta^j < 1,\ \text{(B.3a)}\ \textit{and}\ \text{(B.3c)}).
\end{aligned}
$$

By rearranging this expression, we obtain

$$
\theta^{2j} \geq \frac{3}{L_H + \eta}\frac{\big((1-\zeta)\epsilon_H - \delta_H\big)\|\mathbf{d}_k\| - \delta_{g,k}}{\|\mathbf{d}_k\|^2}.
$$

Because $\delta_H \leq (1-\zeta)\epsilon_H/2$, from Condition 7, this bound implies that

$$
\theta^{2j} \geq \frac{3}{L_H + \eta}\frac{(1-\zeta)\epsilon_H\|\mathbf{d}_k\|/2 - \delta_{g,k}}{\|\mathbf{d}_k\|^2}. \tag{B.9}
$$

Since by assumption $\|\mathbf{d}_k\| \geq \epsilon_g/\epsilon_H$, we have either that

$$
\delta_{g,k} < \frac{1-\zeta}{4}\epsilon_g = \frac{1-\zeta}{4}\epsilon_H\frac{\epsilon_g}{\epsilon_H} \leq \frac{(1-\zeta)\epsilon_H\|\mathbf{d}_k\|}{4}, \tag{B.10}
$$

or else

$$
\delta_{g,k} < \frac{1-\zeta}{8}\min(\epsilon_H\|\mathbf{d}_k\|, \|\mathbf{g}_k\|, \|\mathbf{g}_{k+1}\|) < \frac{(1-\zeta)\epsilon_H\|\mathbf{d}_k\|}{4}. \tag{B.11}
$$

In either case, we have that $(1-\zeta)\epsilon_H\|\mathbf{d}_k\|/2 - \delta_g \geq (1-\zeta)\epsilon_H\|\mathbf{d}_k\|/4$, so we have from (B.9) that

$$
\theta^{2j} \geq \frac{3}{L_H + \eta}\frac{(1-\zeta)\epsilon_H}{4\|\mathbf{d}_k\|}. \tag{B.12}
$$

Since in the case under consideration, the acceptance condition for the backtracking line search fails for $j = 0$, the latter expression holds with $j = 0$, and we have

$$\|\mathbf{d}_k\| \geq \frac{3(1-\zeta)\epsilon_H}{4(L_H+\eta)}. \tag{B.13}$$

From (B.12), (B.3b), and (3.5), we know that

$$\theta^{2j} \geq \frac{3(1-\zeta)\epsilon_H}{4(L_H+\eta)}\|\mathbf{d}_k\|^{-1} \geq \frac{3(1-\zeta)\epsilon_H}{4(L_H+\eta)}\frac{\epsilon_H}{U_g\sqrt{1+\zeta^2/4}}. \tag{B.14}$$

Since

$$j_{\text{sol}} = \left\lceil \frac{1}{2}\log_\theta \frac{3(1-\zeta)\epsilon_H^2}{4U_g(L_H+\eta)\sqrt{1+\zeta^2/4}} \right\rceil,$$

then for any $j > j_{\text{sol}}$,

$$\theta^{2j} < \theta^{2j_{\text{sol}}} < \frac{3(1-\zeta)\epsilon_H^2}{4U_g(L_H+\eta)\sqrt{1+\zeta^2/4}}.$$

By comparing this expression with (B.14), we conclude that the line-search must terminate with $j_k \leq j_{\text{sol}} + 1$, and the previous index $j = j_k - 1$ satisfies

$$\theta^{2j_k-2} \geq \frac{3(1-\zeta)\epsilon_H}{4(L_H+\eta)}\|\mathbf{d}_k\|^{-1}.$$

So,

$$\theta^{j_k} \geq \sqrt{\frac{3\theta^2(1-\zeta)}{4(L_H+\eta)}}\epsilon_H^{1/2}\|\mathbf{d}_k\|^{-1/2}.$$

Then, we have

$$\begin{aligned} f(\mathbf{x}_k) - f(\mathbf{x}_k + \theta^{j_k}\mathbf{d}_k) &\geq \frac{\eta}{6}\theta^{3j_k}\|\mathbf{d}_k\|^3 \\ &\geq \frac{\eta}{6}\left[\frac{3\theta^2(1-\zeta)}{4(L_H+\eta)}\right]^{3/2}\epsilon_H^{3/2}\|\mathbf{d}_k\|^{3/2} \\ &\geq \frac{\eta}{6}\left[\frac{3\theta^2(1-\zeta)}{4(L_H+\eta)}\right]^{3/2}\epsilon_g^{3/2}, \end{aligned} \tag{B.15}$$

where the last inequality follows from (B.13).

Combining the two cases completes the proof. ∎

Next, we deal with the negative curvature directions, for which $d_{\text{type}} = \text{NC}$. Lemma 30 and Lemma 31 present the results of the negative curvature direction from Procedure 8 and 9, respectively.

Lemma 30. *Let Assumption 3, and Condition 7 hold. Suppose that at iteration k of Algorithm 3, we have $\|\mathbf{g}_k\| > \epsilon_g$, so that Procedure 8 is called. When Procedure 8 outputs a direction $\mathbf{d}_k$ with $d_{\text{type}} = \text{NC}$, the backtracking line search requires at most $j_k \leq j_{\text{sol}} + 1$ iterations, where*

$$j_{\text{sol}} = 2\left\lceil \log_\theta \frac{3}{2(L_H+\eta)} \right\rceil.$$

The resulting step $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \mathbf{d}_k$ satisfies

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1}) \geq c_{\text{nc}} \epsilon_H^3,$$

where

$$c_{\text{nc}} = \frac{\eta}{6}\left[\frac{3\theta}{2(L_H+\eta)}\right]^3.$$

Proof. First, if $\alpha_k = 1$, then the result holds trivially. Hence, we only need to consider the case where the non-unit step size is accepted. From (B.5), we have

$$\mathbf{d}_k^T \mathbf{H}_k \mathbf{d}_k \leq -\|\mathbf{d}_k\|^3 \leq -\epsilon_H \|\mathbf{d}_k\|^2.$$

Also,

$$\|\mathbf{d}_k^T(\mathbf{H}_k - \nabla^2 f(\mathbf{x}_k))\mathbf{d}_k\| \leq \delta_H \|\mathbf{d}_k\|^2.$$

Combining the two above together, we have

$$\mathbf{d}_k^T \nabla^2 f(\mathbf{x}_k) \mathbf{d}_k \leq -\|\mathbf{d}_k\|^3 + \delta_H \|\mathbf{d}_k\|^2.$$

When θ^j does not satisfy the termination criterion, either $\nabla f(\mathbf{x}_k)^T \mathbf{d}_k \leq 0$ or $-\nabla f(\mathbf{x}_k)^T \mathbf{d}_k \leq 0$. Here, we suppose $\nabla f(\mathbf{x}_k)^T \mathbf{d}_k \leq 0$, and then

$$\begin{aligned}
-\frac{\eta}{6}\theta^{3j}\|\mathbf{d}_k\|^3 &\leq f(\mathbf{x}_k + \theta^j \mathbf{d}_k) - f(\mathbf{x}_k) \\
&\leq \theta^j \nabla f(\mathbf{x}_k)^T \mathbf{d}_k + \frac{\theta^{2j}}{2}\mathbf{d}_k^T \nabla^2 f(\mathbf{x}_k)\mathbf{d}_k + \frac{L_H}{6}\theta^{3j}\|\mathbf{d}_k\|^3 \\
&\leq -\frac{\theta^{2j}}{2}\|\mathbf{d}_k\|^3 + \frac{\theta^{2j}}{2}\delta_H\|\mathbf{d}_k\|^2 + \frac{L_H}{6}\theta^{3j}\|\mathbf{d}_k\|^3,
\end{aligned}$$

which implies

$$\theta^j \geq \frac{6}{L_H+\eta}\frac{\|\mathbf{d}_k\| - \delta_H}{2\|\mathbf{d}_k\|} = \frac{3}{L_H+\eta} - \frac{3\delta_H}{(L_H+\eta)\|\mathbf{d}_k\|} \geq \frac{3}{L_H+\eta} - \frac{3\delta_H}{(L_H+\eta)\epsilon_H}.$$

When $\delta_H < \epsilon_H/2$,

$$\theta^j \geq \frac{3}{2(L_H+\eta)}.$$

Therefore, when

$$j_{\text{nc}} = 2\left\lceil \log_\theta \frac{3}{2(L_H+\eta)} \right\rceil,$$

line-search must terminate with $j_k \le j_{\text{nc}} + 1$; and the previous index $j_k - 1$ satisfies

$$\theta^{j_k-1} \ge \frac{3}{2(L_H+\eta)}.$$

Hence, from (B.5), we have

$$f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) \ge c_{\text{nc}}\epsilon_H^3,$$

where

$$c_{\text{nc}} = \frac{\eta}{6}\left[\frac{3\theta}{2(L_H+\eta)}\right]^3.$$

∎

We now turn our attention to the property of Procedure 9. The following lemma shows that when $d_{\text{type}} = \text{NC}$ obtained from Procedure 9, there is a sufficient descent in the function.

Lemma 31. *Let Assumption 3 and Condition 7 hold. Suppose that at iteration k of Algorithm 3, the search direction $\mathbf{d}_k$ is of negative curvature type, obtained either directly from Procedure 9 or as the output of Procedure 8 and $d_{\text{type}} = \text{NC}$. Then the line-search terminates with step size $\alpha_k = \theta^{j_k}$ with $j_k \le j_{\text{nc}} + 1$ where j_{nc} is defined as Lemma 30 and the decrease of the function value resulting from the chose step size satisfies*

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k + \alpha_k\mathbf{d}_k) \ge \frac{c_{\text{nc}}}{8}\epsilon_H^3, \tag{B.16}$$

where c_{nc} is given in Lemma 30.

Proof. Note that

$$\mathbf{d}_k^T\mathbf{H}\mathbf{d}_k \le \frac{1}{2}\|\mathbf{d}_k\|^3 \le \frac{1}{2}\epsilon_H\|\mathbf{d}_k\|^2,$$

i.e., $\|\mathbf{d}_k\| \ge \epsilon_H/2$. The proof can be obtained by replacing ϵ_H with $\epsilon_H/2$ in Lemma 30. ∎

Now comes the last but a crucial part. When the output direction $\mathbf{d}_k$ from Procedure 8 satisfies $\|\mathbf{d}_k\| \le \epsilon_g/\epsilon_H$ and Procedure 9 detects no large negative curvature in the Hessian, the update of $\mathbf{x}_k$ with unit step along $\mathbf{d}_k$ is the last step. Dealing with this case is particularly critical to obtaining the convergence rate of our inexact damped Netwon-CG algorithm.

Lemma 32. *Let Assumption 3, and Condition 7 hold. Suppose that, at iteration k of Algorithm 3, $\mathbf{H}_k \succeq -\epsilon_H I$ and the update $\mathbf{d}_k$ obtained from Procedure 8 satisfies $\|\mathbf{d}_k\| \le \epsilon_g/\epsilon_H$. Then*

$$\|\nabla f(\boldsymbol{x}_k + \mathbf{d}_k)\| \le \frac{L_H}{2}\frac{\epsilon_g^2}{\epsilon_H^2} + 4\epsilon_g, \quad \lambda_{\min}(\nabla^2 f(\boldsymbol{x}_k + \mathbf{d}_k)) \ge -(2\epsilon_H + L_H\frac{\epsilon_g}{\epsilon_H})I, \tag{B.17}$$

i.e., the algorithm terminates at the next step.

Proof. To begin, we check the first-order condition as

$$\begin{aligned}
\|\nabla f(\mathbf{x}_k+\mathbf{d}_k)\| &= \|\nabla f(\mathbf{x}_k+\mathbf{d}_k)-\mathbf{g}_k+\mathbf{g}_k\| \\
&\le \left\|\nabla f(\mathbf{x}_k+\mathbf{d}_k)-\nabla f(\mathbf{x}_k)-\nabla^2 f(\mathbf{x}_k)\mathbf{d}_k+\mathbf{H}_k\mathbf{d}_k+\mathbf{g}_k\right\| \\
&\quad + \|\nabla f(\mathbf{x}_k)-\mathbf{g}_k\| + \left\|\nabla^2 f(\mathbf{x}_k)\mathbf{d}_k-\mathbf{H}_k\mathbf{d}_k\right\| \\
&\le \left\|\nabla f(\mathbf{x}_k+\mathbf{d}_k)-\nabla f(\mathbf{x}_k)-\nabla^2 f(\mathbf{x}_k)\mathbf{d}_k\right\| + \|\mathbf{H}_k\mathbf{d}_k+\mathbf{g}_k\| + \delta_{g,k}+\delta_H\|\mathbf{d}_k\| \\
&\le \left\|\nabla f(\mathbf{x}_k+\mathbf{d}_k)-\nabla f(\mathbf{x}_k)-\nabla^2 f(\mathbf{x}_k)\mathbf{d}_k\right\| + \|\hat{r}_k\| + 2\epsilon_H\|\mathbf{d}_k\| + \delta_{g,k}+\delta_H\|\mathbf{d}_k\| \\
&\le \frac{L_H}{2}\|\mathbf{d}_k\|^2 + \frac{1}{2}\epsilon_H\zeta\|\mathbf{d}_k\| + (2\epsilon_H+\delta_H)\|\mathbf{d}_k\| + \delta_{g,k} \\
&\le \frac{L_H}{2}\|\mathbf{d}_k\|^2 + ((\zeta/2+2)\epsilon_H+\delta_H)\|\mathbf{d}_k\| + \delta_{g,k} \\
&\le \frac{L_H}{2}\|\mathbf{d}_k\|^2 + (\zeta/2+3)\epsilon_H\|\mathbf{d}_k\| + \frac{1-\zeta}{8}\max\left(\epsilon_g, \min(\epsilon_H\|\mathbf{d}_k\|, \|\mathbf{g}_k\|, \|\mathbf{g}_{k+1}\|)\right) \\
&\le \frac{L_H}{2}\frac{\epsilon_g^2}{\epsilon_H^2} + (\zeta/2+3)\epsilon_g + \frac{1-\zeta}{8}\max\left(\epsilon_g, \epsilon_H\|\mathbf{d}_k\|\right) \\
&\le \frac{L_H}{2}\frac{\epsilon_g^2}{\epsilon_H^2} + (\zeta/2+3)\epsilon_g + \frac{1-\zeta}{8}\epsilon_g \\
&\le \frac{L_H}{2}\frac{\epsilon_g^2}{\epsilon_H^2} + 4\epsilon_g.
\end{aligned}$$

The third inequality above uses the fact that $\|\hat{r}_k\| \le \epsilon_H\zeta\|\mathbf{d}_k\|/2$.

Now, we check the second-order condition. Since $\mathbf{H}_k \succeq -\epsilon_H I$, then

$$\nabla^2 f(\mathbf{x}_k+\mathbf{d}_k) \succeq \nabla^2 f(\mathbf{x}_k) - L_H\|\mathbf{d}_k\| I \succeq \mathbf{H}_k - \delta_H I - L_H\frac{\epsilon_g}{\epsilon_H}I \succeq -(2\epsilon_H + L_H\frac{\epsilon_g}{\epsilon_H})I.$$

This completes the proof. ∎

Proof of Theorem 4

Proof. First of all, as Lemma 28 states, the failure rate of each time Procedure 9 called is δ. If there are at most $\bar{K}$ steps, then the success rate is at least $(1-\delta)^{\bar{K}}$. Next, let us show that Algorithm 3 will be terminated after at most $\bar{K}$ step to satisfy the second-order optimility. Suppose Algorithm 3 terminates after K steps. We will show that $K \le \bar{K}$ conditioning that Procedure 9 always succeeds during the whole process. Particularly, we partition the

K steps into 5 possible cases.

$$\begin{aligned}
&\mathcal{K}_1 := \{k = 0, 1, 2, \cdots, K-1 | \|\mathbf{g}_k\| \le \epsilon\} \\
&\mathcal{K}_2 := \{k = 0, 1, 2, \cdots, K-1 | \|\mathbf{g}_k\| \ge \epsilon, \|\mathbf{g}_{k+1}\| \le \epsilon\} \\
&\mathcal{K}_3 := \{k = 0, 1, 2, \cdots, K-1 | \|\mathbf{g}_k\| \ge \epsilon, \|\mathbf{g}_{k+1}\| \ge \epsilon, \|\mathbf{d}_k\| \ge \sqrt{\epsilon/L_H}\} \\
&\mathcal{K}_4 := \{k = 0, 1, 2, \cdots, K-1 | \|\mathbf{g}_k\| \ge \epsilon, \|\mathbf{g}_{k+1}\| \ge \epsilon, \|\mathbf{d}_k\| \le \sqrt{\epsilon/L_H}, \mathbf{H}_k \succeq -\sqrt{L_H\epsilon}\mathbf{I}\} \\
&\mathcal{K}_5 := \{k = 0, 1, 2, \cdots, K-1 | \|\mathbf{g}_k\| \ge \epsilon, \|\mathbf{g}_{k+1}\| \ge \epsilon, \|\mathbf{d}_k\| \le \sqrt{\epsilon/L_H}, \mathbf{H}_k \preceq -\sqrt{L_H\epsilon}\mathbf{I}\}
\end{aligned}$$

Obviously, $K = |\mathcal{K}_1|+|\mathcal{K}_2|+|\mathcal{K}_3|+|\mathcal{K}_4|+|\mathcal{K}_5|$. **Case 1:** $\mathcal{K}_1 := \{k = 0, 1, 2, \cdots, \bar{K}-1 | \|\mathbf{g}_k\| \le \epsilon\}$. The update $\mathbf{d}_k$ in this case must come from Procedure 9. With Lemma 31, we have

$$f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) \ge \frac{c_{\text{nc}}}{8}\epsilon_H^3 = \frac{c_{\text{nc}}L_H^{3/2}}{8}\epsilon^{3/2}. \tag{B.18}$$

Case 2: $\mathcal{K}_2 := \{k = 0, 1, 2, \cdots, \bar{K}-1 | \|\mathbf{g}_k\| \ge \epsilon, \|\mathbf{g}_{k+1}\| \le \epsilon\}$. With Lemma 29, we can only guarantee that $f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) \ge 0$. However, since $\|\mathbf{g}_{k+1}\| \le \epsilon$, the next iterate must fall into the case $\mathcal{K}_1$. Therefore we have $|\mathcal{K}_2| \le |\mathcal{K}_1| + 1$.

Case 3: $\mathcal{K}_3 := \{k = 0, 1, 2, \cdots, \bar{K}-1 | \|\mathbf{g}_k\| \ge \epsilon, \|\mathbf{g}_{k+1}\| \ge \epsilon, \|\mathbf{d}_k\| \ge \sqrt{\epsilon/L_H}\}$. With Lemma 29 and 30, we have

$$\begin{aligned}
f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) &\ge \min\left(c_{\text{sol}} \min\left(\frac{(\|\mathbf{g}_{k+1}\| - \delta_{g,k} - \delta_{g,k+1})^3}{\epsilon_H^3}, \epsilon_H^3, \epsilon_g^{3/2}\right), c_{\text{nc}}\epsilon_H^3\right) \\
&\ge \min\left(c_{\text{sol}} \min\left(\frac{\epsilon_g^3}{8\epsilon_H^3}, \epsilon_H^3, \epsilon_g^{3/2}\right), c_{\text{nc}}\epsilon_H^3\right) \\
&= \min\left(c_{\text{sol}} \min\left(\frac{\epsilon^{3/2}}{8L_H^{3/2}}, L_H^{3/2}\epsilon^{3/2}, \epsilon^{3/2}\right), c_{\text{nc}}L_H^{3/2}\epsilon^{3/2}\right) \\
&= \min\left(\frac{c_{\text{sol}}}{8L_H^{3/2}}, c_{\text{sol}}L_H^{3/2}, c_{\text{nc}}L_H^{3/2}, c_{\text{sol}}\right)\epsilon^{3/2}.
\end{aligned} \tag{B.19}$$

Case 4: $\mathcal{K}_4 := \{k = 0, 1, 2, \cdots, \bar{K}-1 | \|\mathbf{g}_k\| \ge \epsilon, \|\mathbf{g}_{k+1}\| \ge \epsilon, \|\mathbf{d}_k\| \le \sqrt{\epsilon/L_H}, \mathbf{H}_k \succeq -\sqrt{L_H\epsilon}\mathbf{I}\}$. With Lemma 32, this iterate must be the second last step since the next iterate will terminate. Therefore we have $|\mathcal{K}_4| \le 1$.

Case 5: $\mathcal{K}_5 := \{k = 0, 1, 2, \cdots, \bar{K}-1 | \|\mathbf{g}_k\| \ge \epsilon, \|\mathbf{g}_{k+1}\| \ge \epsilon, \|\mathbf{d}_k\| \le \sqrt{\epsilon/L_H}, \mathbf{H}_k \preceq -\sqrt{L_H\epsilon}\mathbf{I}\}$. In this case, the final update for this iterate will go through Procedure 9. Similar to Case 1, we will have

$$f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) \ge \frac{c_{\text{nc}}}{8}\epsilon_H^3 = \frac{c_{\text{nc}}L_H^{3/2}}{8}\epsilon^{3/2}. \tag{B.20}$$

Now we consider the total decrease of f over all the K steps[1]

$$
\begin{aligned}
f(\mathbf{x}_0) - f_{\text{low}} &\geq \sum_{k=0}^{K-1} (f(\mathbf{x}_k) - f(\mathbf{x}_{k+1})) \\
&\geq \sum_{k\in\mathcal{K}_1} (f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) + \sum_{k\in\mathcal{K}_3} (f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) + \sum_{k\in\mathcal{K}_5} (f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) \\
&\geq (|\mathcal{K}_1| + |\mathcal{K}_5|) \frac{c_{\text{nc}} L_H^{3/2}}{8} \epsilon^{3/2} + |\mathcal{K}_3| \min\left(\frac{c_{\text{sol}}}{8L_H^{3/2}}, c_{\text{sol}} L_H^{3/2}, c_{\text{sol}}, c_{\text{nc}} L_H^{3/2} \right) \epsilon^{3/2}.
\end{aligned}
$$

Therefore

$$
|\mathcal{K}_1| + |\mathcal{K}_5| \leq \frac{f(\mathbf{x}_0) - f_{\text{low}}}{c_{\text{nc}} L_H^{3/2}/8} \epsilon^{-3/2}, |\mathcal{K}_3| \leq \frac{f(\mathbf{x}_0) - f_{\text{low}}}{\min\left(\frac{c_{\text{sol}}}{8L_H^{3/2}}, c_{\text{sol}} L_H^{3/2}, c_{\text{sol}}, c_{\text{nc}} L_H^{3/2} \right)} \epsilon^{-3/2}.
$$

Finally, we have

$$
\begin{aligned}
K &= |\mathcal{K}_1| + |\mathcal{K}_2| + |\mathcal{K}_3| + |\mathcal{K}_4| + |\mathcal{K}_5| \\
&\leq \frac{f(\mathbf{x}_0) - f_{\text{low}}}{c_{\text{nc}} L_H^{3/2}/8} \epsilon^{-3/2} + \frac{f(\mathbf{x}_0) - f_{\text{low}}}{c_{\text{nc}} L_H^{3/2}/8} \epsilon^{-3/2} + 1 + \frac{f(\mathbf{x}_0) - f_{\text{low}}}{\min\left(\frac{c_{\text{sol}}}{8L_H^{3/2}}, c_{\text{sol}} L_H^{3/2}, c_{\text{sol}}, c_{\text{nc}} L_H^{3/2} \right)} \epsilon^{-3/2} + 1 \\
&\leq \frac{3(f(\mathbf{x}_0) - f_{\text{low}})}{\min\left(c_{\text{sol}}/(8L_H^{3/2}), c_{\text{sol}} L_H^{3/2}, c_{\text{sol}}, c_{\text{nc}} L_H^{3/2}/8 \right)} \epsilon^{-3/2} + 2.
\end{aligned}
$$

This completes the proof. ∎

B.3 Proof of Theorem 5

We now show that the fixed step size can result in a sufficient descent in the function $f(\mathbf{x}_k)$ when $d_{\text{type}} = \text{SOL}$ and $\|\mathbf{d}_k\| \geq \sqrt{\epsilon_g/L_H}$. The following lemma can be seen as a modification of Lemma 29 with fixed step size.

Lemma 33. *Let Assumption 3, and Condition 8 hold. Suppose that at iteration k of Algorithm 4, we have $\|\mathbf{g}_k\| > \epsilon_g$, so that Procedure 8 is called. When Procedure 8 outputs a direction $\mathbf{d}_k$ with $d_{\text{type}} = \text{SOL}$ and $\|\mathbf{d}_k\| \geq \sqrt{\epsilon_g/L_H}$, we can choose a pre-defined step size,*

$$
\alpha_k = \left[\frac{3(1-\zeta)}{4(L_H + \eta)} \right]^{1/2} \frac{\epsilon_H^{1/2}}{\|\mathbf{d}_k\|^{1/2}}.
$$

[1]If $|\mathcal{K}_4| = 1$, we should consider the function decrease over the $K-1$ steps. The analysis on bounding $|\mathcal{K}_1|, |\mathcal{K}_3|, |\mathcal{K}_5|$ remains the same.

The resulting step $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \mathbf{d}_k$ *satisfies*

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1}) \geq \bar{c}_{\text{sol}} \epsilon_H^3,$$

where

$$\bar{c}_{\text{sol}} = \frac{\eta}{6} \left[\frac{3(1-\zeta)}{4L_H(L_H+\eta)} \right]^3 .$$

Proof. First, we prove that $\alpha \leq 1$. Indeed,

$$\begin{aligned} \alpha_k^2 &= \left[\frac{3(1-\zeta)}{4(L_H+\eta)} \right] \frac{\epsilon_H}{\|\mathbf{d}_k\|} \\ &\leq \left[\frac{3(1-\zeta)}{4(L_H+\eta)} \right] \frac{\sqrt{L_H \epsilon_g}}{\sqrt{\epsilon_g / L_H}} \\ &= \frac{3(1-\zeta)L_H}{4(L_H+\eta)} \\ &= \frac{3}{4} \frac{L_H}{L_H+\eta} (1-\zeta) \\ &< 1. \end{aligned} \tag{B.21}$$

It is not hard to see if

$$-\frac{\eta}{6} \alpha_k^3 \|\mathbf{d}_k\|^3 \geq f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) - f(\mathbf{x}_k),$$

then the statement in the lemma is correct by substitute α_k into the formula. Now, we prove the lemma by contradiction. Suppose we have

$$\begin{aligned} -\frac{\eta}{6} \alpha_k^3 \|\mathbf{d}_k\|^3 &\leq f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) - f(\mathbf{x}_k) \\ &\leq \alpha_k^3 \nabla f_k^T \mathbf{d}_k + \frac{\alpha_k^2}{2} \mathbf{d}_k^T \nabla^2 f(\mathbf{x}_k) \mathbf{d}_k + \frac{L_H}{6} \alpha_k^3 \|\mathbf{d}_k\|^3 \\ &= \alpha_k \mathbf{g}_k^T \mathbf{d}_k + \alpha_k (\nabla f_k - \mathbf{g}_k)^T \mathbf{d}_k + \frac{\alpha_k^2}{2} \mathbf{d}_k^T (\nabla^2 f(\mathbf{x}_k) - \mathbf{H}_k) \mathbf{d}_k + \frac{L_H}{6} \alpha_k^3 \|\mathbf{d}_k\|^3 \\ &\quad + \frac{\alpha_k^2}{2} \mathbf{d}_k^T \mathbf{H}_k \mathbf{d}_k \\ &\leq \alpha_k \delta_{g,k} \|\mathbf{d}_k\| - \frac{\alpha_k}{2} (1-\zeta) \epsilon_H \|\mathbf{d}_k\|^2 + \frac{\alpha_k^2}{2} \delta_H \|\mathbf{d}_k\|^2 + \frac{L_H}{6} \alpha_k^3 \|\mathbf{d}_k\|^3 \\ &< \alpha_k \delta_{g,k} \|\mathbf{d}_k\| - \frac{\alpha_k}{2} \|\mathbf{d}_k\|^2 \big((1-\zeta)\epsilon_H - \delta_H \big) + \frac{L_H}{6} \alpha_k^3 \|\mathbf{d}_k\|^3 \qquad (\alpha_k < 1). \end{aligned}$$

Substituting α_k and $\delta_H \leq (1-\zeta)\epsilon_H/2$ into the above formula, we have

$$\frac{L_H+\eta}{6} \left[\frac{3(1-\zeta)}{4(L_H+\eta)} \right] \frac{\epsilon_H}{\|\mathbf{d}_k\|} \|\mathbf{d}_k\|^2 - \frac{(1-\zeta)\epsilon_H}{4} \|\mathbf{d}_k\| + \delta_{g,k} > 0,$$

which can be simplified as

$$\frac{1}{2}\epsilon_H\|\mathbf{d}_k\| - \epsilon_H\|\mathbf{d}_k\| + \frac{1}{2}\max\left(\epsilon_g, \min(\epsilon_H\|\mathbf{d}_k\|, \|\mathbf{g}_k\|, \|\mathbf{g}_{k+1}\|)\right) > 0.$$

If $\epsilon_g > \min(\epsilon_H\|\mathbf{d}_k\|, \|\mathbf{g}_k\|, \|\mathbf{g}_{k+1}\|)$, since $\epsilon_H = \sqrt{L_H\epsilon_g}$, we have

$$-\sqrt{L_H\epsilon_g}\|\mathbf{d}_k\| + \epsilon_g > 0 \Rightarrow \|\mathbf{d}_k\| < \sqrt{\frac{\epsilon_g}{L_H}},$$

which contradicts our assumption $\|\mathbf{d}_k\| \geq \sqrt{\epsilon_g/L_H}$. If $\epsilon_g < \min(\epsilon_H\|\mathbf{d}_k\|, \|\mathbf{g}_k\|, \|\mathbf{g}_{k+1}\|)$, then

$$\begin{aligned}
0 &< \frac{1}{2}\epsilon_H\|\mathbf{d}_k\| - \epsilon_H\|\mathbf{d}_k\| + \frac{1}{2}\max\left(\epsilon_g, \min(\epsilon_H\|\mathbf{d}_k\|, \|\mathbf{g}_k\|, \|\mathbf{g}_{k+1}\|)\right)\\
&= \frac{1}{2}\epsilon_H\|\mathbf{d}_k\| - \epsilon_H\|\mathbf{d}_k\| + \frac{1}{2}\min(\epsilon_H\|\mathbf{d}_k\|, \|\mathbf{g}_k\|, \|\mathbf{g}_{k+1}\|)\\
&\leq \frac{1}{2}\epsilon_H\|\mathbf{d}_k\| - \epsilon_H\|\mathbf{d}_k\| + \frac{1}{2}\epsilon_H\|\mathbf{d}_k\| = 0\\
&\Rightarrow 0 > 0,
\end{aligned} \tag{B.22}$$

which is again a contradiction. ∎

Next, let us deal with the case when $d_{\text{type}} = \text{NC}$, which can be considered as a fixed step size alternative of Lemma 30.

Lemma 34. *Let Assumption 3 and Condition 8 hold. Suppose that at iteration k of Algorithm 3, we have $\|\mathbf{g}_k\| > \epsilon_g$, so that Procedure 8 is called. When Procedure 8 outputs a direction $\mathbf{d}_k$ with $d_{\text{type}} = \text{NC}$, we can choose a constant step size*

$$\alpha_k = \tilde{\theta}\frac{(\|\mathbf{d}_k\| - \delta_H)/2 + \sqrt{((\|\mathbf{d}_k\| - \delta_H)/2)^2 - 4(L_H + \eta)\delta_{g,k}/6}}{(L_H + \eta)\|\mathbf{d}_k\|/6},$$

where $2 - \sqrt{3} < \tilde{\theta} < 1$. The resulting step $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\mathbf{d}_k$ satisfies

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1}) \geq \bar{c}_{\text{nc}}\epsilon_H^3,$$

where

$$\bar{c}_{\text{nc}} = \frac{\eta}{6}\left[\frac{3\tilde{\theta}}{(L_H + \eta)}\right]^3.$$

Proof. Same as in the proof of Lemma 30, we have

$$\mathbf{d}_k^T\nabla^2 f(\mathbf{x}_k)\mathbf{d}_k \leq -\|\mathbf{d}_k\|^3 + \delta_H\|\mathbf{d}_k\|^2.$$

When α_k does not satisfy the termination criterion, then

$$\begin{aligned}-\frac{\eta}{6}\alpha_k^3\|\mathbf{d}_k\|^3 &\le f(\mathbf{x}_k+\alpha_k\mathbf{d}_k)-f(\mathbf{x}_k)\\ &\le \alpha_k\nabla f(\mathbf{x}_k)^T\mathbf{d}_k+\frac{\alpha_k^2}{2}\mathbf{d}_k^T\nabla^2 f(\mathbf{x}_k)\mathbf{d}_k+\frac{L_H}{6}\alpha_k^3\|\mathbf{d}_k\|^3\\ &\le \alpha_k\delta_{g,k}\|\mathbf{d}_k\|-\frac{\alpha_k^2}{2}\|\mathbf{d}_k\|^3+\frac{\alpha_k^2}{2}\delta_H\|\mathbf{d}_k\|^2+\frac{L_H}{6}\alpha_k^3\|\mathbf{d}_k\|^3.\end{aligned}$$

This shows

$$-\frac{\eta}{6}\alpha_k^2\|\mathbf{d}_k\|^2 \le \delta_{g,k}-\frac{\alpha_k}{2}\|\mathbf{d}_k\|^2+\frac{\alpha_k}{2}\delta_H\|\mathbf{d}_k\|+\frac{L_H}{6}\alpha_k^2\|\mathbf{d}_k\|^2.$$

By re-arranging the terms, we have the following quadratic function:

$$\frac{(L_H+\eta)\|\mathbf{d}_k\|^2}{6}\alpha_k^2-\frac{\|\mathbf{d}_k\|(\|\mathbf{d}_k\|-\delta_H)}{2}\alpha_k+\delta_{g,k}\ge 0.$$

Since

$$\begin{aligned}\Delta &= (\frac{\|\mathbf{d}_k\|(\|\mathbf{d}_k\|-\delta_H)}{2})^2-4\frac{(L_H+\eta)\|\mathbf{d}_k\|^2}{6}\delta_{g,k}\\ &\ge \|\mathbf{d}_k\|^2(\frac{1}{8}\|\mathbf{d}_k\|^2-\frac{2(L_H+\eta)}{3}\delta_{g,k}) \quad (\text{Since } \|\mathbf{d}_k\|\ge\epsilon_H,\ \delta_H\le\frac{1}{2}\epsilon_H)\\ &> 0 \quad (\mathit{Condition}\ 8),\end{aligned}$$

there are two solutions of the above quadratic inequality, i.e.,

$$\begin{aligned}\alpha_k &\ge \frac{(\|\mathbf{d}_k\|-\delta_H)/2+\sqrt{((\|\mathbf{d}_k\|-\delta_H)/2)^2-4(L_H+\eta)\delta_{g,k}/6}}{(L_H+\eta)\|\mathbf{d}_k\|/6}\equiv\beta_1,\\ \text{or } \alpha_k &\le \frac{(\|\mathbf{d}_k\|-\delta_H)/2-\sqrt{((\|\mathbf{d}_k\|-\delta_H)/2)^2-4(L_H+\eta)\delta_{g,k}/6}}{(L_H+\eta)\|\mathbf{d}_k\|/6}\equiv\beta_2.\end{aligned}$$

Next, we show that our pre-defined step size setting is in the between of β_1 and β_2. First, it is obvious that $\alpha_k=\tilde{\theta}\beta_1<\beta_1$ since $\theta<1$. Second, proving $\alpha_k>\beta_2$ is equivalent to proving

$$\begin{aligned}&\tilde{\theta}_k\ge\frac{\beta_1}{\beta_2}\\ \Leftrightarrow\ &\tilde{\theta}_k\ge\frac{(\|\mathbf{d}_k\|-\delta_H)/2-\sqrt{(\|\mathbf{d}_k\|-\delta_H)^2/4-4(L_H+\eta)\delta_{g,k}/6}}{(\|\mathbf{d}_k\|-\delta_H)/2+\sqrt{(\|\mathbf{d}_k\|-\delta_H)^2/4-4(L_H+\eta)\delta_{g,k}/6}}\\ \Leftrightarrow\ &\tilde{\theta}_k\ge\frac{x-\sqrt{x^2-c}}{x+\sqrt{x^2-c}} \quad (\text{Let } x=\frac{(\|\mathbf{d}_k\|-\delta_H)}{2},\ \text{and } c=4\frac{(L_H+\eta)}{6}\delta_{g,k})\\ \Leftrightarrow\ &\tilde{\theta}_k\ge\frac{(x-\sqrt{x^2-c})^2}{c}.\end{aligned}$$

Since $x > 2c$ (Condition 8) and $(x - \sqrt{x^2 - c})$ is a decreasing function when $x > c$, we know that $\tilde{\theta}_k > 2 - \sqrt{3}$ satisfies $\alpha_k > \beta_2$.

Since $c < x/2$ and $\delta_h < \|d_k\|^2/2$, we have

$$\begin{aligned}\alpha_k &= \tilde{\theta}\frac{(\|\mathbf{d}_k\| - \delta_H)/2 + \sqrt{((\|\mathbf{d}_k\| - \delta_H)/2)^2 - 4(L_H + \eta)\delta_{g,k}/6}}{(L_H + \eta)\|\mathbf{d}_k\|/6} \\ &\geq \tilde{\theta}\frac{\|\mathbf{d}_k\|/4 + \sqrt{3}\|\mathbf{d}_k\|/4}{(L_H + \eta)\|\mathbf{d}_k\|/6} > \frac{3\tilde{\theta}}{L_H + \eta}.\end{aligned}$$

∎

The following lemma shows that when $d_{\text{type}} = \text{NC}$ obtained from Procedure 9, fixed step size can also be applied.

Lemma 35. *Let Assumption 3 and Condition 8 hold. Suppose that at iteration k of Algorithm 3, the search direction $\mathbf{d}_k$ is of negative curvature type, obtained either directly from Procedure 9 or as the output of Procedure 8 and $\mathrm{d}_{\text{type}} = \text{NC}$. We can choose a constant step size*

$$\alpha_k = \tilde{\theta}\frac{(\|\mathbf{d}_k\| - \delta_H)/2 + \sqrt{((\|\mathbf{d}_k\| - \delta_H)/2)^2 - 4(L_H + \eta)\delta_{g,k}/6}}{(L_H + \eta)\|\mathbf{d}_k\|/6},$$

where $2 - \sqrt{3} < \tilde{\theta} < 1$, and the decrease of the function value resulting from the chose step size satisfies

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k + \alpha_k\mathbf{d}_k) \geq \frac{\bar{c}_{\text{nc}}}{8}\epsilon_H^3, \tag{B.23}$$

where $\bar{c}_{\text{nc}}$ is given in Lemma 34.

Proof. Note that

$$\mathbf{d}_k^T\mathbf{H}\mathbf{d}_k \leq \frac{1}{2}\|\mathbf{d}_k\|^3 \leq \frac{1}{2}\epsilon_H\|\mathbf{d}_k\|^2,$$

i.e., $\|\mathbf{d}_k\| \geq \epsilon_H/2$. The proof can be obtained by replacing ϵ_H with $\epsilon_H/2$ in Lemma 34. ∎

Appendix C

Appendix for Chapter 4

C.1 Descending Property of (4.6)

Assume that $f(w)$ is a strongly convex and strictly smooth function in $\mathbb{R}^d$, such that there exists positive constants α and β so that $\alpha I \leq \nabla^2 f(w) \leq \beta I$ for all w. We can show that the update formulation of (4.6) is a converging algorithm. Particularly, we can show that with the proper learning rate:

$$f(w_{t+1}) - f(w_t) \leq -\frac{\alpha^k}{2\beta^{1+k}}\|\mathbf{g}_t\|^2. \tag{C.1}$$

Note that when $k = 0$ or 1, the convergence rate is the same as gradient descent or Newton method[1] [28], respectively. Our proof is similar to [28] for Newton method.

Let us define $\lambda(w_t) = ({\mathbf{g}_t}^T {\mathbf{H}_t}^{-k} \mathbf{g}_t)^{1/2}$. Since $f(w)$ is strongly convex, we have

$$\begin{aligned} f(w_t - \eta \Delta w_t) &\leq f(w_t) - \eta \mathbf{g}_t^T \Delta w_t + \frac{\eta^2 \beta \|\Delta w_t\|^2}{2} \\ &\leq f(w_t) - \eta \lambda(w_t)^2 + \frac{\beta}{2\alpha^k}\eta^2 \lambda(w_t)^2. \end{aligned} \tag{C.2}$$

The last inequality comes from the fact that

$$\lambda(w_t)^2 = {\Delta w_t}^T \mathbf{H}_t^k \Delta w_t \geq \alpha^k \|\Delta w_t\|^2. \tag{C.3}$$

Therefore, the step size $\hat{\eta} = \frac{\alpha^k}{\beta}$ will make f decreases as follows,

$$f(w_t - \hat{\eta}\Delta w_t) \leq f(w_t) - \frac{1}{2}\hat{\eta}\lambda(w_t)^2. \tag{C.4}$$

Since $\alpha \preceq \mathbf{H}_t \preceq \beta$, we have

$$\lambda(w_t)^2 = {\mathbf{g}_t}^T {\mathbf{H}_t}^{-k} \mathbf{g}_t \geq \frac{1}{\beta^k}\|\mathbf{g}_t\|^2. \tag{C.5}$$

[1]The convergence rate here denotes the global convergence rate of Newton's method.

Therefore,

$$f(w_t - \hat{\eta}\Delta w_t) - f(w_t) \leq -\frac{1}{2\beta^k}\hat{\eta}\|\mathbf{g}_t\|^2 = -\frac{\alpha^k}{2\beta^{1+k}}\|\mathbf{g}_t\|^2.$$

C.2 Descending Property of (4.7)

When $f(w)$ is a strongly convex and strictly smooth function in $\mathbb{R}^d$, such that there exists positive constants α and β so that $\alpha I \leq \nabla^2 f(w) \leq \beta I$ for all w, we can prove that (4.7) has the same convergence rate as (4.6).

First of all, it is not hard to see the diagonal elements in $\boldsymbol{D}$ are all positive since $f(w)$ is a strongly convex problem. That is,

$$\alpha \leq e_i^T \mathbf{H} e_i = e_i^T \boldsymbol{D} e_i = D_{i,i}, \tag{C.6}$$

where e_i is the vector whose coordinates are all zero, except the i-th one that equals 1. Similarly, we have

$$D_{i,i} = e_i^T \boldsymbol{D} e_i = e_i^T \mathbf{H} e_i \leq \beta. \tag{C.7}$$

Therefore, the diagonal elements in $\boldsymbol{D}$ are in the range $[\alpha, \beta]$. Using the same proof as in Appendix C.1, we will get the result.

C.3 Descending Property of (4.10)

When $f(w)$ is a strongly convex and strictly smooth function in $\mathbb{R}^d$, such that there exists positive constants α and β so that $\alpha I \leq \nabla^2 f(w) \leq \beta I$ for all w, we can prove that (4.10) has the same convergence rate as (4.6).

As shown in Appendix C.2, the diagonal elements in $\boldsymbol{D}$ are in the range $[\alpha, \beta]$. Therefore, the average of a subset of those numbers is still in the range $[\alpha, \beta]$. Using the same proof as in Appendix C.1, we will get the result.

C.4 Experimental Setup

Here, we provide more details on the experimental setup for the empirical evaluation.

Image Classification. The training/test sets for Cifar10 [129] dataset contain 50k/10k images, respectively. The models used on Cifar10 are standard ResNet20/32. We train both models with 160 epochs and decay the learning rate by a factor of 10 at epoch 80 and 120. The batch size is set to be 256. For SGD/Adam/AdamW, the initial learning rates are tuned and set to be 0.1/0.001/0.01. For AdaHessian, we set the block size as 9, k to be 1, and learning rate as 0.15 for both ResNet20/32. For Adam/AdamW/AdaHessian, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We run each experiment 5 times on Cifar10 and report the mean and standard deviation of the results. The training/test sets for ImageNet dataset [65] contain 1.2M/50k images, respectively. Our code is modified from the official PyTorch example[2]. The batch size is set to be 256. We train ResNet18 for 90 epochs. All the settings of different optimizers are the same as used in Cifar10 example.

Neural Machine Translation. The training/validation/test sets for the IWSLT14 dataset contain about 153K/7K/7K sentence pairs, respectively. We use a vocabulary of 10K tokens via a joint source and target byte pair encoding (BPE). For the WMT14 dataset, we follow the setting of [232], which contains 4.5M parallel sentence pairs for training. We use Newstest2014 as the test set, and Newstest2013 as the validation set. The 37K vocabulary for WMT14 is also via a joint source and target BPE factorization. We set dropout as 0.0 for Transformer `base/small` model. For AdamW, we follow the optimizer setting and learning rate schedule in [236]. For AdaHessian, we set the block size as 32 for IWSLT/WMT, k to be 1.0, and learning rate as 0.047/1.0 for IWSLT/WMT. For both AdamW/AdaHessian, we set $\beta_1 = 0.9$ and $\beta_2 = 0.98$. We fix the label smoothing value as $\epsilon_{\text{ls}} = 0.1$ in all experiments. We implement our code for MT based on *fairseq-py* [173]. We employ BLEU[3] [176] as the evaluation metric for MT. Following standard practice, we measure tokenized case-sensitive BLEU and case-insensitive BLEU for WMT14 En-De and IWSLT14 De-En, respectively. For a fair comparison, we do not include other external datasets. We train 130/55 epochs for WMT/IWSLT, respectively. We set the maximum token size to be 4096×8 (eight gpus)/4096 (one gpu) for WMT/IWSLT. For inference, we average the last 10/5 checkpoints, and we set the length penalty as 0.6/1.0 and beam size as 4/5 for WMT/IWSLT, following [173]. We run each experiment 5 times on IWSLT and report the mean and standard deviation of the results.

Language Modeling. PTB [159] has 0.93M training tokens, 0.073M validation tokens, and 0.082M test tokens. Wikitext-103 [157] contains 0.27M unique words, and 100M training words from 28K articles, with an average length of 3.6K words per article. We use the same evaluation scheme following [61]. We use a three-layer tensorized transformer core-1 for PTB and a six-layer tensorized transformer core-1 for Wikitext-103, following [149]. We set the dropout rate as 0.3 in all the LM experiments. The model is trained for 30 epochs on both PTB and WikiText-103. For AdamW, we follow the learning rate setting in [149]. For AdaHessian, we set the block size as 4 and k as 0.5 for PTB and Wikitext-103. We set the learning rate as 2.0/1.0 for PTB/Wikitext-103, respectively. We set the batch size to be 60/120 for PTB/Wikitext-103. For AdamW/AdaHessian, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We set the warmup steps to be 4000 and label smoothing to be $\epsilon_{\text{ls}} = 0.1$ in all LM experiments. We run each experiment 5 times on PTB and report the mean and standard deviation of the results.

Natural Language Understanding. The SqueezeBERT model is pre-trained as the authors suggested [116]. Particularly, we pretrain SqueezeBERT from scratch using the LAMB [258] optimizer with a global batch size of 8192, a learning rate of 2.5e-3, and a warmup proportion of 0.28. We pretrain for 56k steps with a maximum sequence length of

[2]https://github.com/pytorch/examples/tree/master/imagenet

[3]https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl

128 and then for 6k steps with a maximum sequence length of 512 followed [117]. Moreover, before directly using the pre-trained model on GLUE tasks, we apply transfer learning from the MNLI GLUE task [234] to other GLUE tasks [117, 182]. We refer readers to [117] for more detailed instruction.

For finetuning, we set the batch size as 16, $\beta_1 = 0.9$, and $\beta_2 = 0.999$ for both AdamW/AdaHessian as suggested in [117]. For AdaHessian, we set the block size b as 4 and k as 1 for all tasks. As in [117] we perform hyperparameter tuning on the learning rate and dropout rate.

Recommendation System. The Criteo Ad Kaggle dataset consists of click logs for ad CTR prediction for 7 days. Each data set contains 13 continuous and 26 categorical features. The dataset contains about 45 million samples over 7 days. In experiments, we follow the setting from [166]. Our code is also modified from [166][4]. The testing metric for Recommendation Systems is Click Through Rate (CTR), measured on training and test sets. For Adagrad, the learning rate is set to be 0.01. For AdaHessian, we set the block size as 1, k as 0.5, learning rate as 0.043, $\beta_1 = 0.9$, and $\beta_2 = 0.98$. We set the batch size to be 128, following [166].

Delayed Hessian Update. For ResNets on Cifar10, we use 5 epochs for warmup. In particular, within 5 epochs, the Hessian diagonal is still computed for every iteration. After that, the Hessian diagonal computation frequency is set to be between 1 to 5 iterations.

C.5 Additional Results

In this section, we present additional empirical results that were discussed in Section 4.4. See Figure C.1 and C.2.

[4]https://github.com/facebookresearch/dlrm

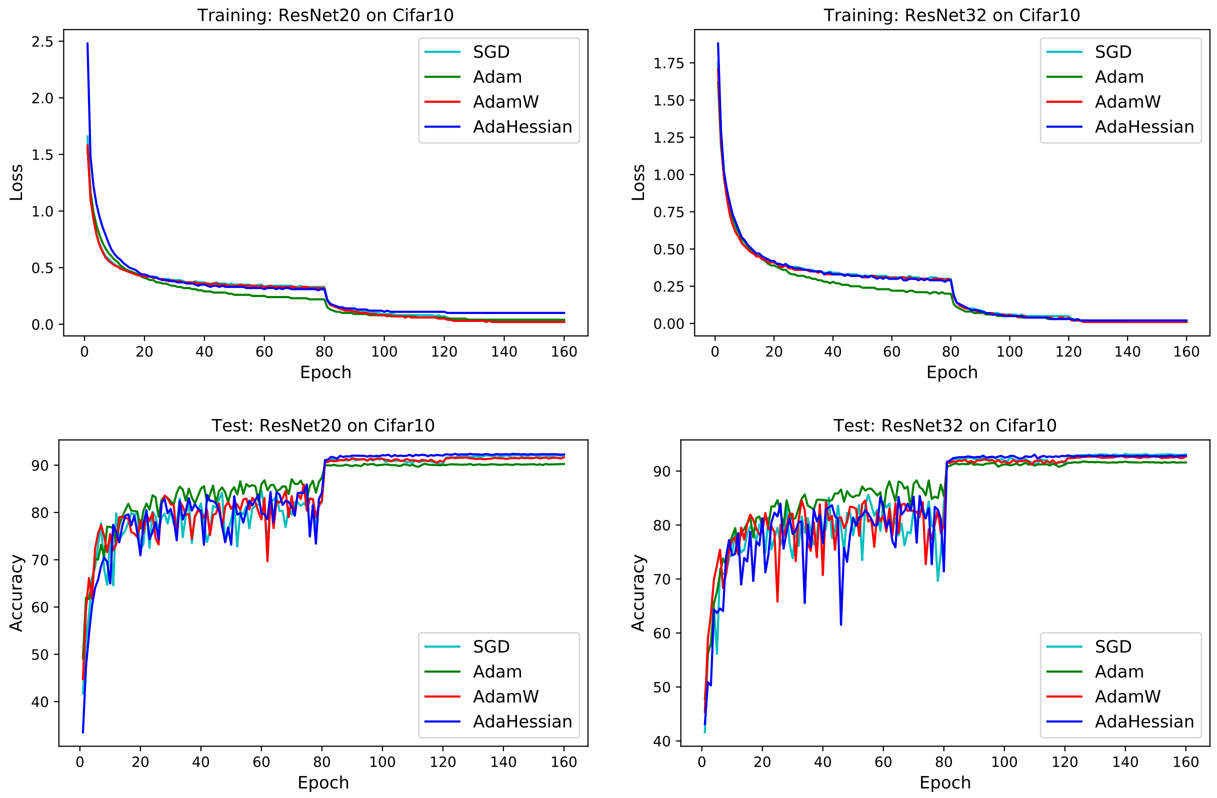

Figure C.1: Training and testing loss curves of SGD, Adam, AdamW, AdaHessian for ResNet20/32 on Cifar10. SGD and AdaHessian consistently achieve better accuracy as compared to Adam and AdamW. The final accuracy results are reported in Table 4.2.

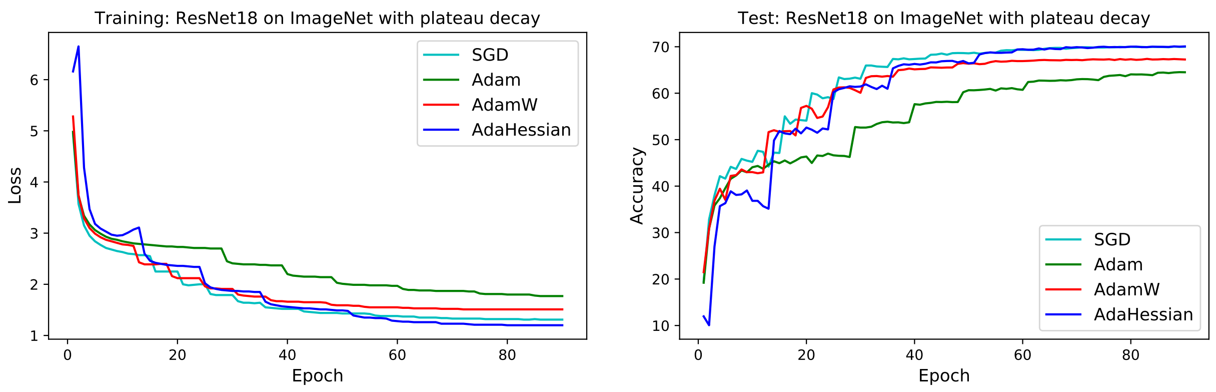

Figure C.2: Training/Test loss curve of SGD, Adam, AdamW, AdaHessian for ResNet18 on ImageNet. SGD and AdaHessian consistently achieve better accuracy as compared to Adam and AdamW. The final accuracy results are reported in Table 4.2.

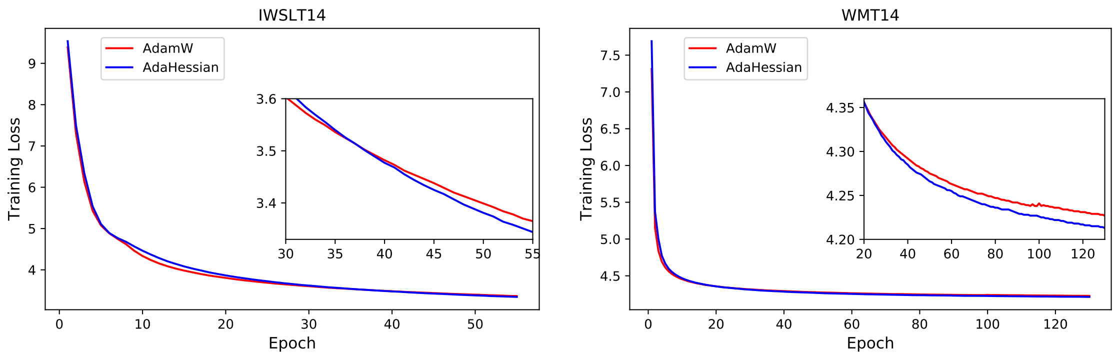

Figure C.3: Training loss curves of AdamW and AdaHessian for Transformer on IWSLT14 and WMT14. The training loss of AdaHessian is lower than that of AdamW on both IWSLT14 and WMT14. Testing results are reported in Table 4.3.

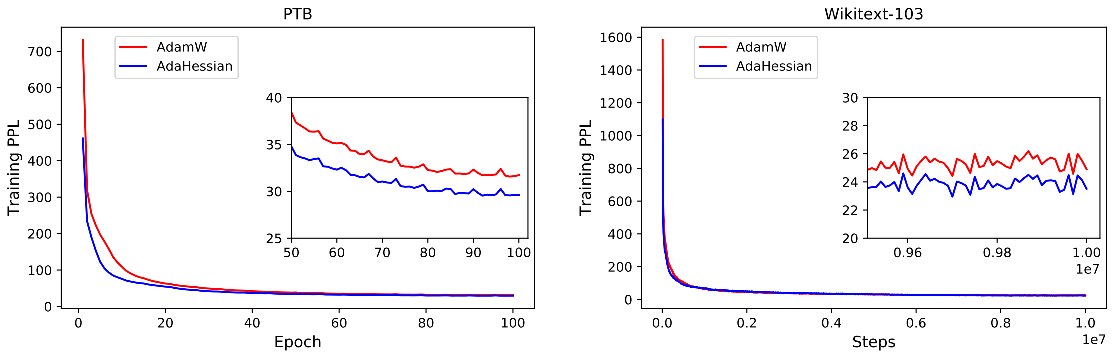

Figure C.4: Training PPL curves of AdamW and AdaHessian for Transformer on PTB and Wikitest-103. The losses of AdaHessian are consistently lower than AdamW from the beginning of the training. AdaHessian achieves 29.56/23.51 final training perplexity (PPL) on PTB/Wikitext-103 as compared to AdamW (31.72/24.01). Testing results are reported in Table 4.4.

Appendix D

Appendix for Chapter 5

D.1 Proof of Theorem 6

In this section, we give the proof of Theorem 6. The first thing we want to point out is that, although we prove the Hessians of these NNs are positive semi-definite almost everywhere, these NNs are not convex w.r.t. inputs, i.e., $\mathbf{x}$. The discontinuity of ReLU is the cause. (For instance, consider a combination of two step functions in 1-D, e.g. $f(x) = 1_{x\geq 1} + 1_{x\geq 2}$ is not a convex function but has 0 second derivative almost everywhere.) However, this has an important implication, that the problem is saddle-free.

Before we go to the proof of Theorem 6, let us prove the following lemma for cross-entropy loss with soft-max layer.

Lemma 36. *Let us denote by $\boldsymbol{s} \in \boldsymbol{R}^d$ the input of the soft-max function, by $y \in \{1, 2, \dots, d\}$ the correct label of the inputs $\boldsymbol{x}$, by $g(\boldsymbol{s})$ the soft-max function, and by $L(\boldsymbol{s}, y)$ the cross-entropy loss. Then we have*

$$\frac{\partial^2 L(\boldsymbol{s}, y)}{\partial \boldsymbol{s}^2} \succeq 0.$$

Proof. Let $s_d = \sum_{j=1}^d e^{\mathbf{s}_j}$, $\mathbf{p}_i = \frac{e^{\mathbf{s}_i}}{s_d}$, and then it follows that

$$L(\mathbf{s}, y) = -\sum_{i=1}^d y_i \log \mathbf{p}_i.$$

Further, it is not hard to see that

$$\begin{aligned}\frac{\partial L(\mathbf{s}, y)}{\partial \mathbf{s}_j} &= -\sum_{i=1}^d y_i \frac{\partial \log \mathbf{p}_i}{\partial \mathbf{s}_j} \\ &= -y_j(1 - \mathbf{p}_j) - \sum_{i\neq j} y_i \frac{\mathbf{p}_k \mathbf{p}_j}{\mathbf{p}_k} \\ &= \mathbf{p}_j - y_j.\end{aligned}$$

Then, the second-order derivative of L w.r.t. $\mathbf{s}_i\mathbf{s}_j$ is

$$\frac{\partial^2 L(\mathbf{s},y)}{\partial \mathbf{s}_j^2} = \mathbf{p}_j(1-\mathbf{p}_j), \qquad \text{and} \qquad \frac{\partial^2 L(\mathbf{s},y)}{\partial \mathbf{s}_j \partial \mathbf{s}_i} = -\mathbf{p}_j\mathbf{p}_i.$$

Since

$$\frac{\partial^2 L(\mathbf{s},y)}{\partial \mathbf{s}_j^2} + \sum_{i\neq j} \frac{\partial^2 L(\mathbf{s},y)}{\partial \mathbf{s}_j \partial \mathbf{s}_i} = 0, \quad \text{and} \quad \frac{\partial^2 L(\mathbf{s},y)}{\partial \mathbf{s}_j^2} \geq 0,$$

we have

$$\frac{\partial^2 L(\mathbf{s},y)}{\partial \mathbf{s}^2} \succeq 0.$$

■

Now, let us give the proof of Theorem 6:

Assume the input of the soft-max layer is $\mathbf{s}$ and the cross-entropy is $L(\mathbf{s},y)$. Based on Chain Rule, it follows that

$$\frac{\partial \mathcal{J}(\boldsymbol{\theta},\mathbf{x},y)}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{s}}\frac{\partial \mathbf{s}}{\partial \mathbf{x}}.$$

From Assumption. 4 we know that all the layers before the soft-max are either linear or ReLU, which indicates $\frac{\partial^2 \mathbf{s}}{\partial \mathbf{x}^2} = 0$ (a tensor) almost everywhere. Therefore, applying chain rule again for the above equation,

$$\begin{aligned}\frac{\partial^2 \mathcal{J}(\boldsymbol{\theta},\mathbf{x},y)}{\partial \mathbf{x}^2} &= (\frac{\partial \mathbf{s}}{\partial \mathbf{x}})^T \frac{\partial^2 L}{\partial \mathbf{s}^2}\frac{\partial \mathbf{s}}{\partial \mathbf{x}} + \frac{\partial L}{\partial \mathbf{s}}\frac{\partial^2 \mathbf{s}}{\partial \mathbf{x}^2} \\ &= (\frac{\partial \mathbf{s}}{\partial \mathbf{x}})^T \frac{\partial^2 L}{\partial \mathbf{s}^2}\frac{\partial \mathbf{s}}{\partial \mathbf{x}}.\end{aligned}$$

It is easy to see $\frac{\partial^2 \mathcal{J}(\boldsymbol{\theta},\mathbf{x},y)}{\partial \mathbf{x}^2} \succeq 0$ almost everywhere since $\frac{\partial^2 L}{\partial \mathbf{s}^2} \succeq 0$ from Lemma 36.

From above we could see that the Hessian of NNs w.r.t. $\mathbf{x}$ is at most a rank c (the number of class) matrix, since the rank of the Hessian matrix

$$\frac{\partial^2 \mathcal{J}(\boldsymbol{\theta},\mathbf{x},y)}{\partial \mathbf{x}^2} = (\frac{\partial \mathbf{s}}{\partial \mathbf{x}})^T \frac{\partial^2 L}{\partial \mathbf{s}^2}\frac{\partial \mathbf{s}}{\partial \mathbf{x}}$$

is dominated by the term $\frac{\partial^2 L}{\partial \mathbf{s}^2}$, which is at most rank c.

D.2 Attacks Mentioned

In this section, we show the details about the attacks used in Chapter 5. Please see Table D.1 for details.

Table D.1: The definition of all attacks used in the paper. Here $\mathbf{g}_x \triangleq \dfrac{\partial \mathcal{J}(\mathbf{x},\theta)}{\partial \mathbf{x}}$ and $\mathbf{H}_x \triangleq \dfrac{\partial^2 \mathcal{J}(\mathbf{x},\theta)}{\partial \mathbf{x}^2}$.

	$\Delta\mathbf{x}$
FGSM	$\epsilon \cdot \text{sign}(\mathbf{g}_x)$
FGSM-10	$\epsilon \cdot \text{sign}(\mathbf{g}_x)$ (iterate 10 times)
L_2GRAD	$\epsilon \cdot \mathbf{g}_x/\|\mathbf{g}_x\|$
FHSM	$\epsilon \cdot \text{sign}(\mathbf{H}_x^{-1}\mathbf{g}_x)$
L_2HESS	$\epsilon \cdot \mathbf{H}_x^{-1}\mathbf{g}_x/\|\mathbf{H}_x^{-1}\mathbf{g}_x\|$

Table D.2: The definition of all models used in the paper.

Name	Structure
C1 (for CIFAR-10)	Conv(5,5,64) – MP(3,3) – BN–Conv(5,5,64) –MP(3,3)–BN–FN(384)–FN(192)–SM(10)
C2 (for CIFAR-10)	Conv(3,3,63)–BN–Conv(3,3,64)–BN–Conv(3,3,128) –BN–Conv(3,3,128)–BN–FC(256)–FC(256)–SM(10)
C3 (for CIFAR-10)	Conv(3,3,64)–Conv(3,3,64)–Conv(3,3,128) –Conv(3,3,128)–FC(256)–FC(256)–SM(10)
M1 (for MNIST)	Conv(5,5,20)–Conv(5,5,50)–FC(500)–SM(10)
CR (for CIFAR-100)	ResNet18 For CIFAR-100

D.3 Models

In this section, we give the details about the NNs used in Chapter 5. For clarification, We omit the ReLu activation here. However, in practice, we implement ReLu regularity. Also, for all convolution layers, we add padding to make sure there is no dimension reduction. We denote Conv(a,a,b) as a convolution layer having b channels with a by a filters, MP(a,a) as a a by a max-pooling layer, FN(a) as a fully-connect layer with a output and SM(a) is the soft-max layer with a output. For our Conv(5,5,b) (Conv(3,3,b))layers, the stride is 2 (1). See Table D.2 for details of all models used in this paper.

D.4 Discussion on second-order Method

Although second-order adversarial attack looks well for MNIST (see Table 5.3), but for most our experiments on CIFAR-10 (see Table 5.4), the second-order methods are weaker than variations of the gradient based methods. Also, notice that the robust models trained by second-order method are also more prone to attack on CIFAR-10, particularly $\mathbf{M}_{FHSM}$ and $\mathbf{M}_{L_2HESS}$. We give two potential explanation here.

Table D.3: Result on MNIST dataset for M1 model (LeNet-5). We shows the Hessian spectrum of different batch training models, and the corresponding performances on adversarial dataset generated by training/testing dataset. The testing results are shown in parenthesis. We report the adversarial accuracy of three different magnitudes of attack. The interesting observation is that the λ_1^θ is increasing while the adversarial accuracy is decreasing for fixed ϵ. Meanwhile, we do not know if there is a relationship between λ_1^θ and Clean accuracy or not. Also, we cannot see the relation between $\lambda_1^{\mathbf{x}}$, $\|\nabla_{\mathbf{x}}\mathcal{J}(\theta,\mathbf{x},y)\|$ and the adversarial accuracy.

Batch	Acc	λ_1^θ	$\lambda_1^{\mathbf{x}}$	$\|\partial_{\mathbf{x}}\mathcal{J}(\theta,\mathbf{x},y)\|$	Acc $\epsilon = 0.2$	Acc $\epsilon = 0.1$
64	100 (99.21)	0.49 (2.96)	0.07 (0.41)	0.007 (0.10)	0.53 (0.53)	0.85 (0.85)
128	100 (99.18)	1.44 (8.10)	0.10 (0.51)	0.009 (0.12)	0.50 (0.51)	0.83 (0.83)
256	100 (99.04)	2.71 (13.54)	0.09 (0.50)	0.008 (0.12)	0.45 (0.46)	0.81 (0.82)
512	100 (99.04)	5.84 (26.35)	0.12 (0.52)	0.010 (0.13)	0.42 (0.42)	0.79 (0.80)
1024	100 (99.05)	21.24 (36.96)	0.25 (0.42)	0.032 (0.11)	0.32 (0.33)	0.73 (0.74)
2048	100 (98.99)	44.30 (49.36)	0.36 (0.39)	0.075 (0.11)	0.19 (0.19)	0.72 (0.73)

First note that the Hessian w.r.t. input is a low rank matrix. In fact, as mentioned above, the rank of the input Hessian for CIFAR-10 is at most ten; see Proposition 1, the matrix itself is $3K \times 3K$. Even though we use inexact Newton method [75] along with Conjugate Gradient solver, but this low rank nature creates numerical problems. Designing preconditioners for second-order attack is part of our future work. The second point is that, as we saw in the previous section the input Hessian does not directly correlate with how robust the network is. In fact, the most effective attack method would be to perturb the input towards the decision boundary, instead of just maximizing the loss.

D.5 More Numerical Result for Section 5.3 and 5.4

In this section, we provide more numerical results for Section 5.3 and 5.4. All conclusions from the numerical results are consistency with those in Section 5.3 and 5.4.

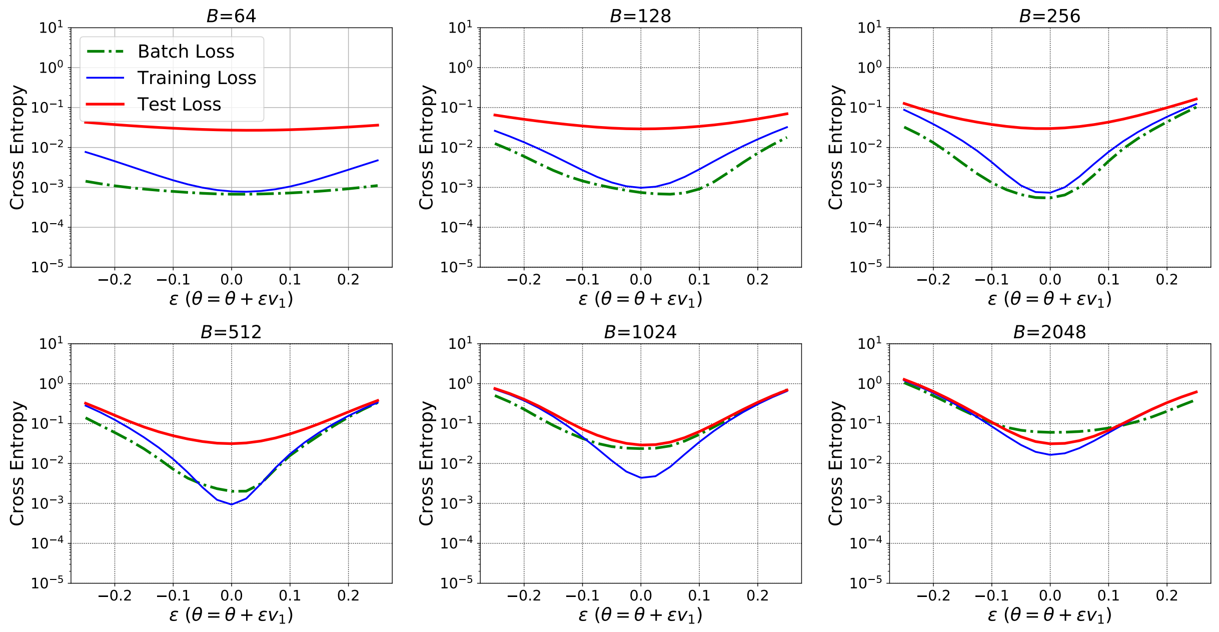

Figure D.1: The landscape of the loss functional is shown along the dominant eigenvector of the Hessian on MNIST for M1. Note that the $y - axis$ is in logarithm scale. Here ϵ is a scalar that perturbs the model parameters along the dominant eigenvector denoted by v_1. The green line is the loss for a randomly batch with batch-size 320 on MNIST. The blue and red line are the training and test loss, respectively. From the figure we could see that the curvature of test loss is much larger than training.

Table D.4: Baseline accuracy is shown for large batch size for C1 model along with results aciheved with scaling learning rate method proposed by [90] (denoted by "FB Acc"). The last column shows results when training is performed with robust optimization. As we can see, the performance of the latter is actually better for large batch size. We emphasize that the goal is to perform analysis to better understand the problems with large batch size training. More extensive tests are needed before one could claim that robust optimization performs better than other methods.

Batch	Baseline Acc	FB Acc	Robust Acc
8000	0.7559	0.752	0.7612
10000	0.7561	0.1	0.7597
25000	0.7023	0.1	0.7409
50000	0.5523	0.1	0.7116

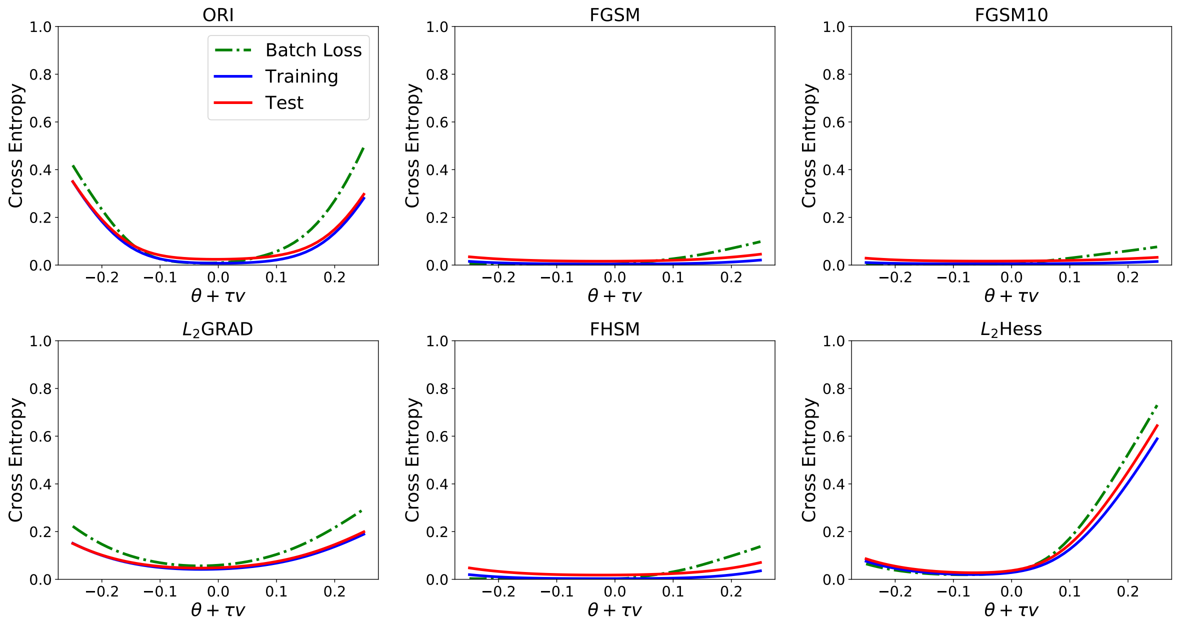

Figure D.2: We show the landscape of the test and training objective functional along the first eigenvector of the sub-sampled Hessian with $B = 320$, i.e. 320 samples from training dataset, on MNIST for M1. We plot both the batch loss as well as the total training and test loss. One can see that visually the results show that the robust models converge to a region with smaller curvature.

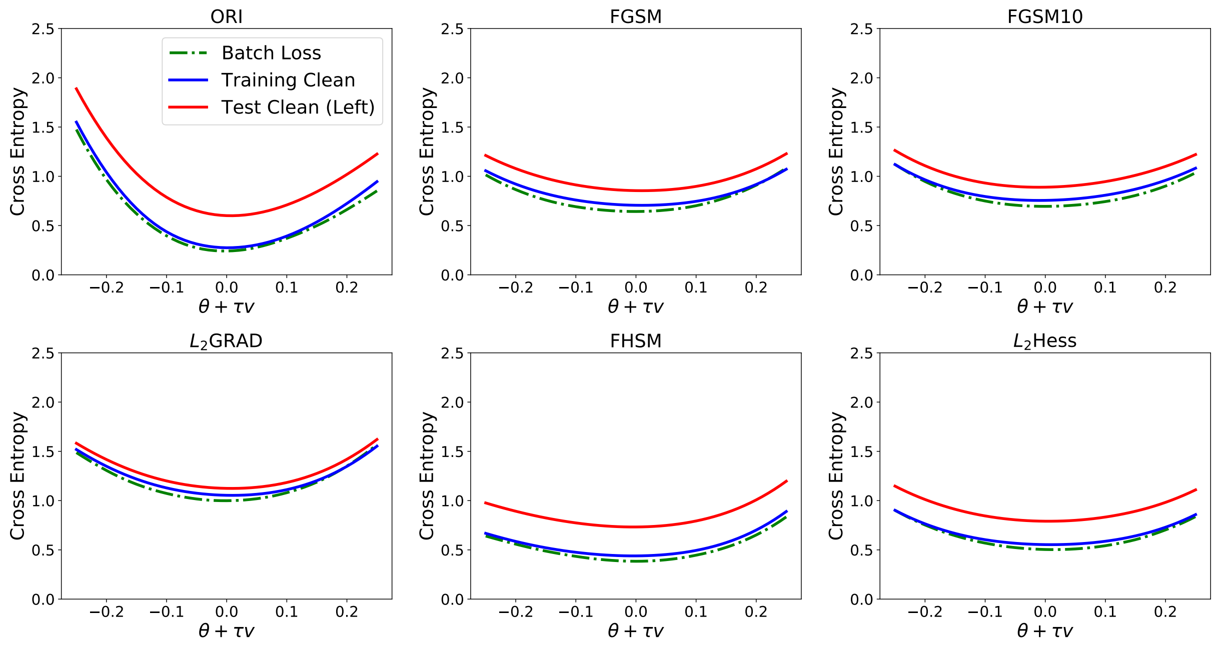

Figure D.3: We show the landscape of the test and training objective functional along the first eigenvector of the sub-sampled Hessian with $B = 320$, i.e. 320 samples from training, on CIFAR-10 for C3. We plot both the batch loss as well as the total training and test loss. One can see that visually the results show that the curvature of robust models is smaller.

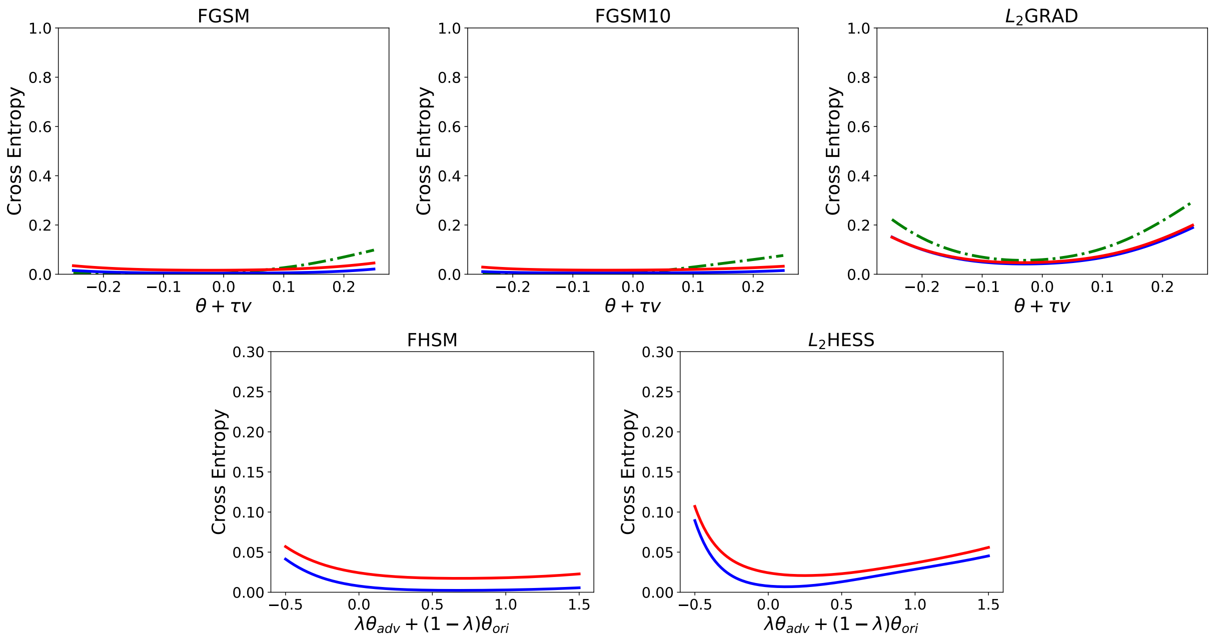

Figure D.4: 1-D Parametric Plot on MNIST for M1 of $\mathbf{M}_{ORI}$ and adversarial models. Here we are showing how the landscape of the total loss functional changes when we interpolate from the original model ($\lambda = 0$) to the robust model ($\lambda = 1$). For all cases the robust model ends up at a point that has relatively smaller curvature compared to the original network.

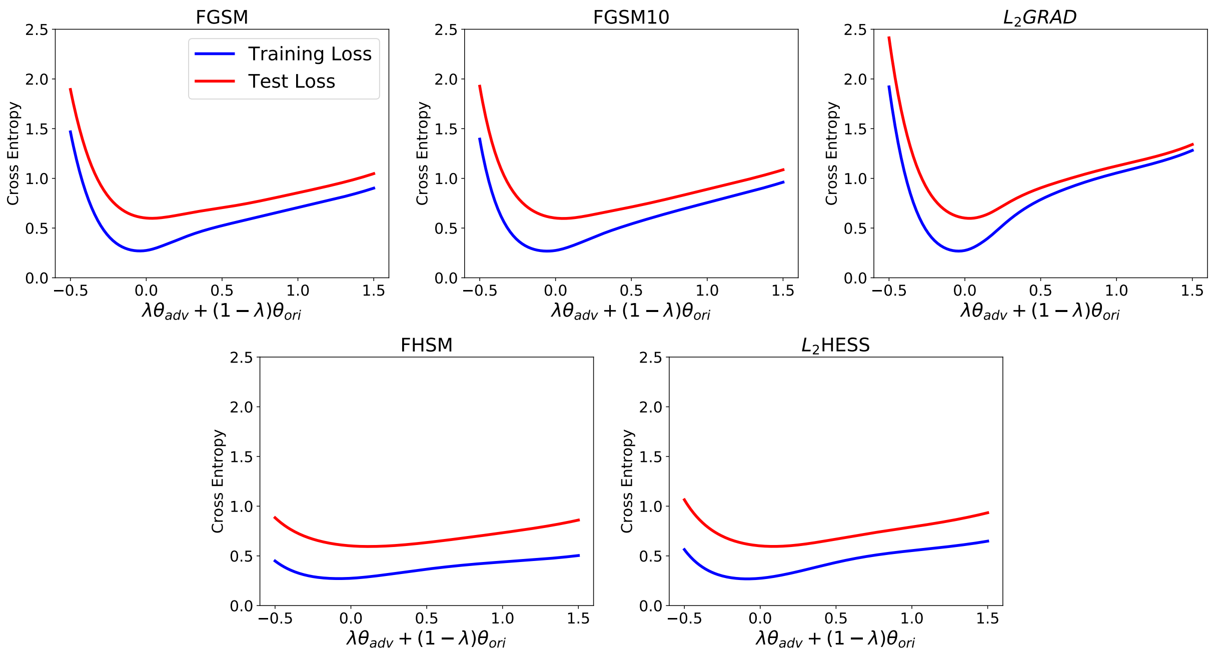

Figure D.5: 1-D Parametric Plot on CIFAR-10 of $\mathbf{M}_{ORI}$ and adversarial models, i.e. total loss functional changes interpolating from the original model ($\lambda = 0$) to the robust model ($\lambda = 1$). For all cases the robust model ends up at a point that has relatively smaller curvature compared to the original network.

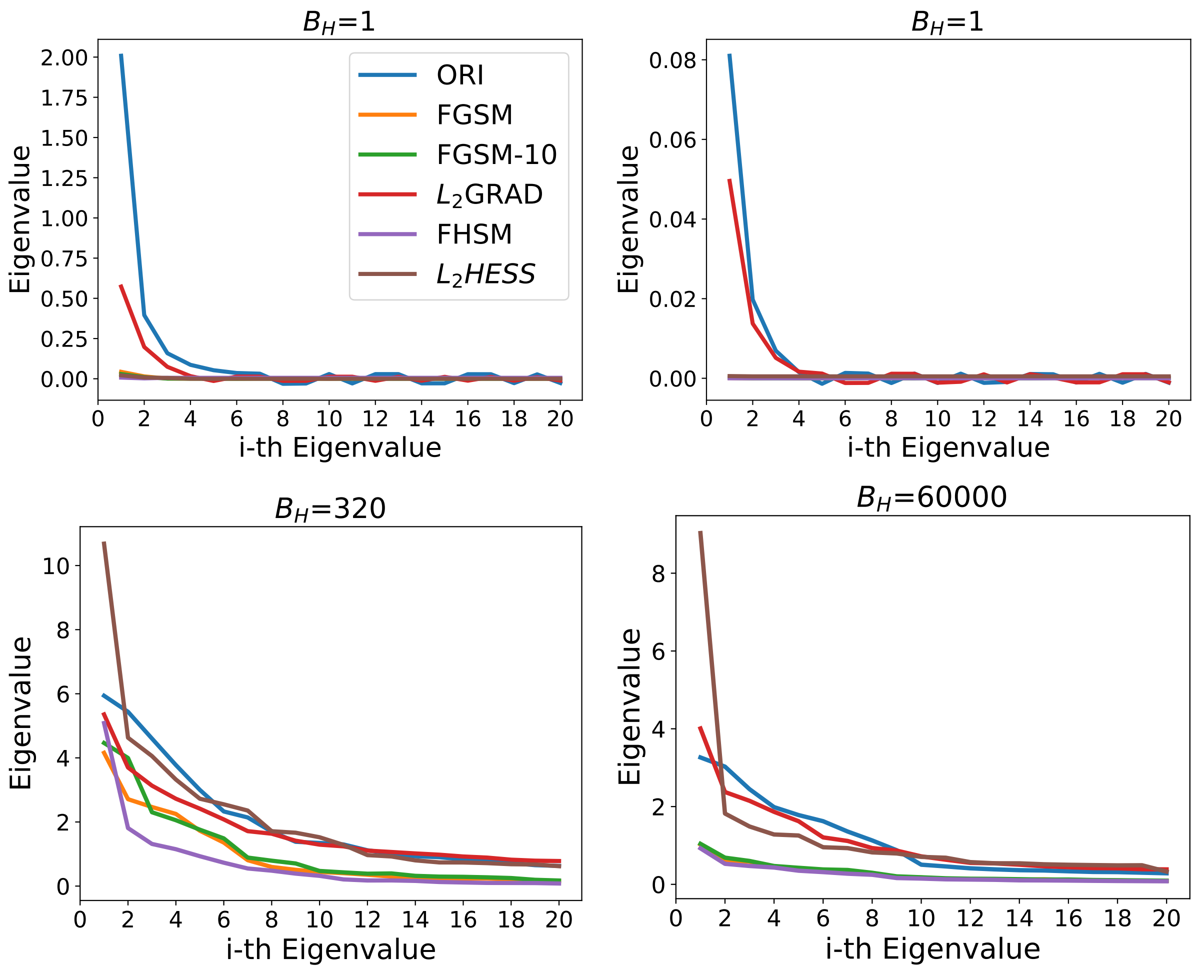

Figure D.6: Spectrum of the sub-sampled Hessian of the loss functional w.r.t. the model parameters computed by power iteration on MNIST of M1. The results are computed for different batch sizes of $B = 1$, $B = 320$, and $B = 60000$. We report two cases for the single batch experiment, which is drawn randomly from the clean training data. The results show that the sub-sampled Hessian spectrum decreases for robust models. An interesting observation is that for the MNIST dataset, the original model has actually converged to a saddle point, even though it has a good generalization error. Also notice that the results for $B = 320$ and $B = 60,000$ are relatively close, which hints that the curvature for the full Hessian should also be smaller for the robust methods. This is demonstrated in Figure D.2.

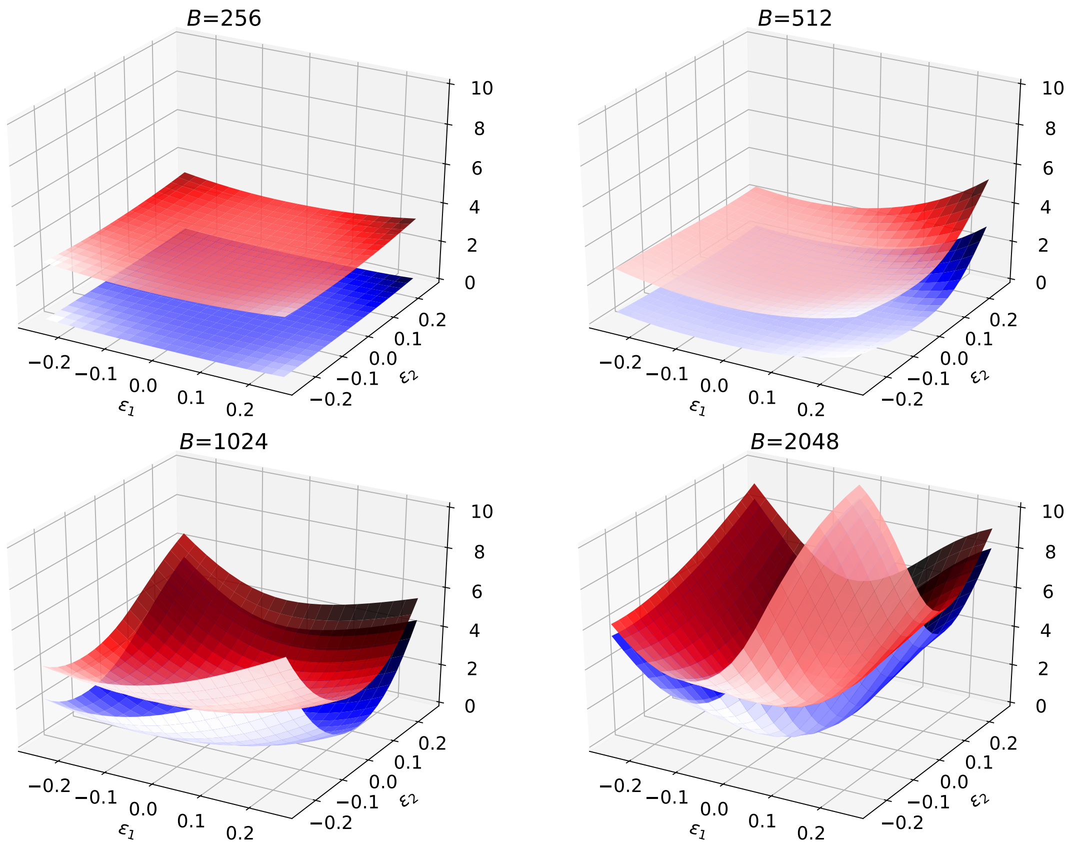

Figure D.7: The landscape of the loss functional is shown when the C2 model parameters are changed along the first two dominant eigenvectors of the Hessian. Here $\epsilon_1,\ \epsilon_2$ are scalars that perturbs the model parameters along the first and second dominant eigenvectors.

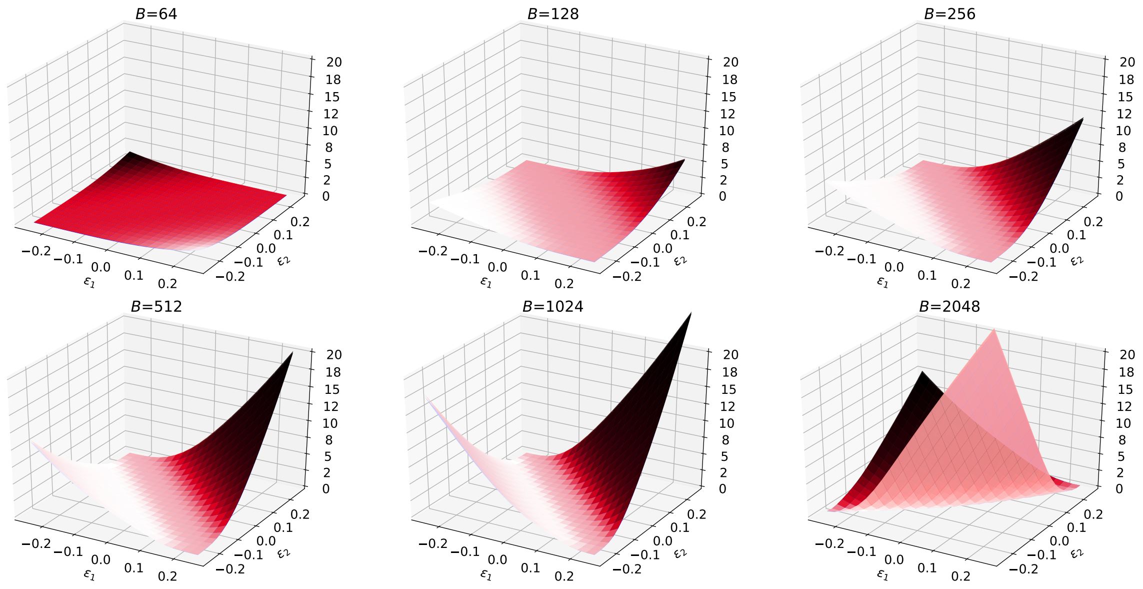

Figure D.8: The landscape of the loss functional is shown when the M1 model parameters are changed along the first two dominant eigenvectors of the Hessian. Here $\epsilon_1,\ \epsilon_2$ are scalars that perturbs the model parameters along the first and second dominant eigenvectors.

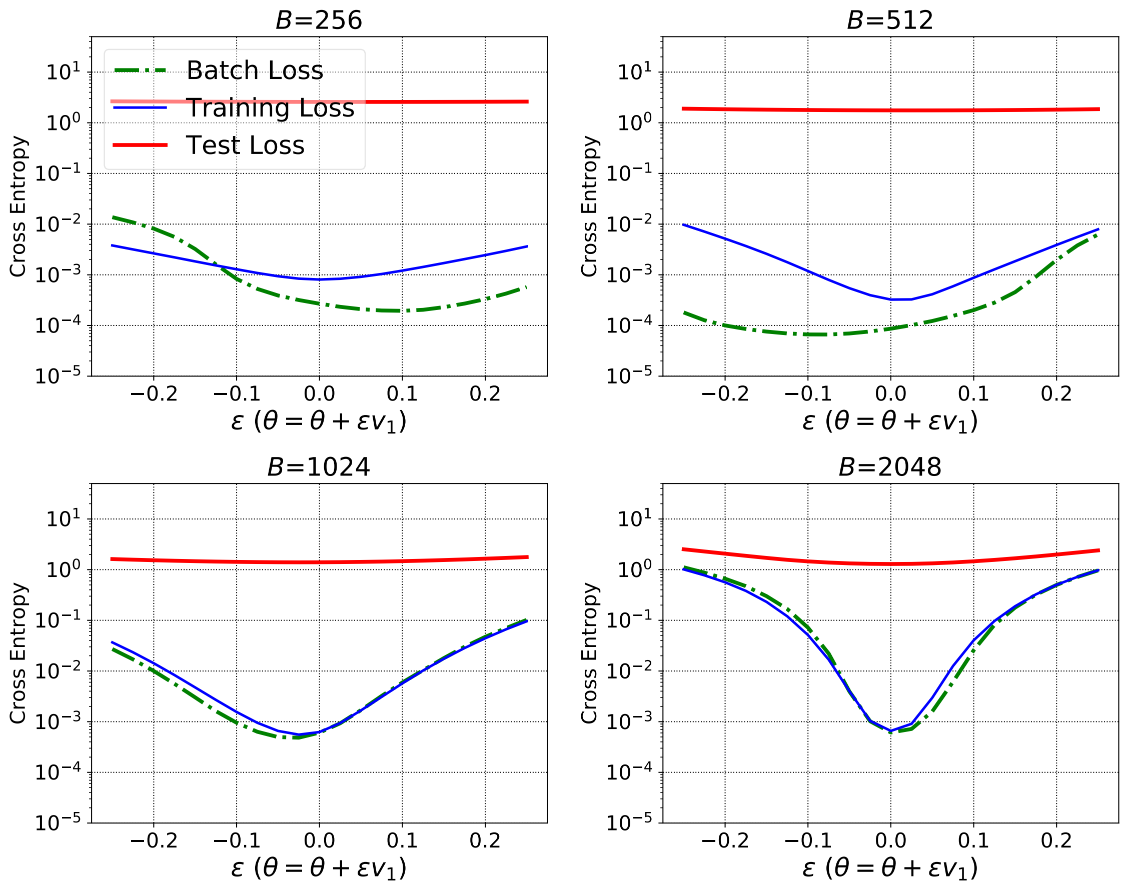

Figure D.9: The landscape of the loss functional is shown along the dominant eigenvector of the Hessian for C2 architecture on CIFAR-10 dataset. Here ϵ is a scalar that perturbs the model parameters along the dominant eigenvector denoted by v_1.

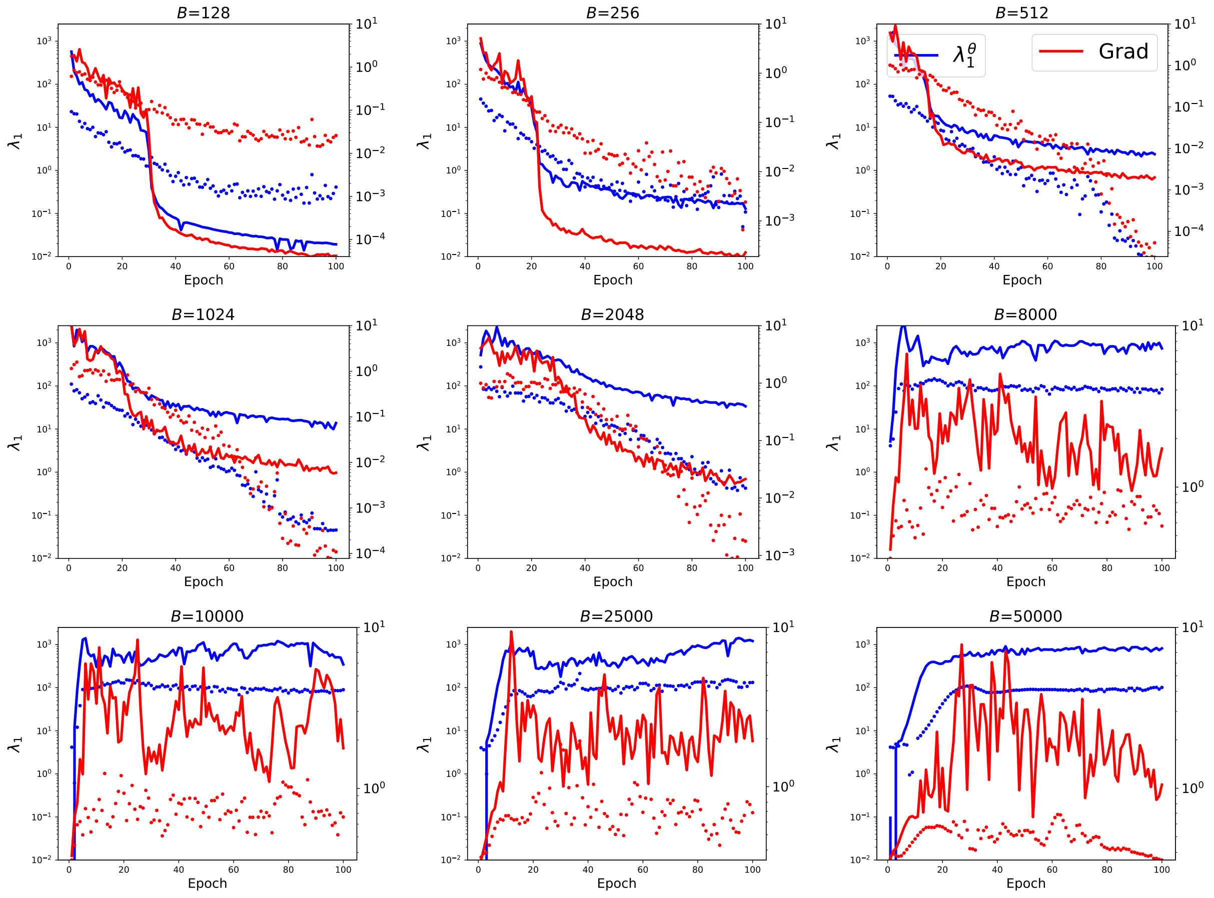

Figure D.10: Changes in the dominant eigenvalue of the Hessian w.r.t weights and the total gradient is shown for different epochs during training. Note the increase in λ_1^θ (blue curve) for large batch vs small batch. In particular, note that the values for total gradient along with the Hessian spectrum show that large batch does not get "stuck" in saddle points, but areas in the optimization landscape that have high curvature. The dotted points show the corresponding results when we use robust optimization. We can see that this pushes the training to flatter areas. This clearly demonstrates the potential to use robust optimization as a means to avoid sharp minimas.

Appendix E

Appendix for Chapter 6

E.1 Illustration of ResNet Stages

In Figure E.1, we show the illustration of ResNet20 on Cifar-10/100 and its three stages.

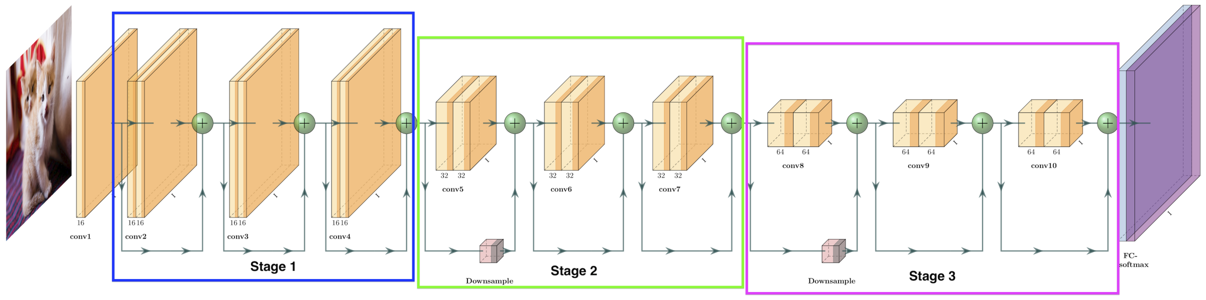

Figure E.1: Illustration of ResNet20 on Cifar-10/100 and its three stages. Blue, green, and purple boxes shows the first, second and third stages, respectively.

E.2 Algorithms

We provide the pseudo-code for power iteration, Hutchinson algorithm, and stochastic Lanczos Quadrature in this section. See Algorithm 10 and Algorithm 11. (Algorithm 6 is presented in the main text.)

E.3 Training Details

We train each model (ResNet, ResNet$_{-BN}$, and ResNet$_{-Res}$) for 180 epochs, with five different initial learning rates (0.1, 0.05, 0.01, 0.005, 0.001) on Cifar-10, and ten different initial

Algorithm 10 Power Iteration for Top Eigenvalue Computation

```
1: **Input**:
       - Parameter: $\theta$
2: Compute the gradient of $\theta$ by backpropagation, i.e., compute $g_\theta = \frac{dL}{d\theta}$.
3: Draw a random vector $v$ from $N(0,1)$ (same dimension as $\theta$).
4: Normalize $v$, $v = \frac{v}{\|v\|_2}$
5: **for** i $= 1, 2, \ldots$ **do**
6:     Compute $gv = g_\theta^T v$
7:     Compute $Hv$ by backpropagation, $Hv = \frac{d(gv)}{d\theta}$
8:     Normalize and reset $v$, $v = \frac{Hv}{\|Hv\|_2}$
9: **end for**
```

Algorithm 11 Hutchinson Method for Trace Computation

```
1: **Input**:
       - Parameter: $\theta$
2: Compute the gradient of $\theta$ by backpropagation, i.e., compute $g_\theta = \frac{dL}{d\theta}$.
3: **for** i $= 1, 2, \ldots$ **do**
4:     Draw a random vector $v$ from Rademacher distribution (same dimension as $\theta$).
5:     Compute $gv = g_\theta^T v$
6:     Compute $Hv$ by backpropagation, $Hv = \frac{d(gv)}{d\theta}$
7:     Compute and record $v^T Hv$
8: **end for**
9: **Output**: the average of all computed $v^T Hv$.
```

learning rates (0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0004, 0.0003, 0.0002, 0.00001) on Cifar-100. The optimizer is SGD with momentum (0.9). The learning rate decays by a factor of 10 at epoch 80, 120.

Table E.1: Accuracy of ResNet, ResNet$_{-BN}$, and ResNet$_{-Res}$ with different depths, on Cifar-100. Results are similar to those shown in Table 6.1, i.e., removing BN (ResNet$_{-BN}$) or residual connections (ResNet$_{-Res}$) results in performance degradation.

Model\Depth	20	32	38
ResNet	66.47%	68.26%	69.06%
ResNet$_{-BN}$	62.82%	25.89%	11.25%
ResNet$_{-Res}$	64.59%	62.08%	62.75%

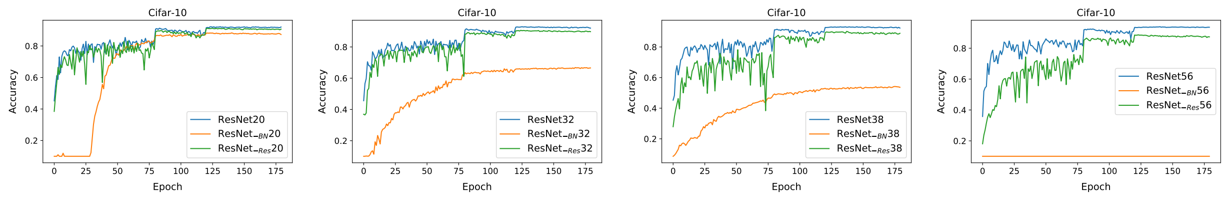

Figure E.2: Testing curve of all models reported in Table 6.1. The generalization performance of models without BN (denoted as ResNet$_{-BN}$) is much worse than the baseline (denoted as ResNet). We see a similar but much smaller generalization loss when the residual connection is removed (denoted as ResNet$_{-Res}$).

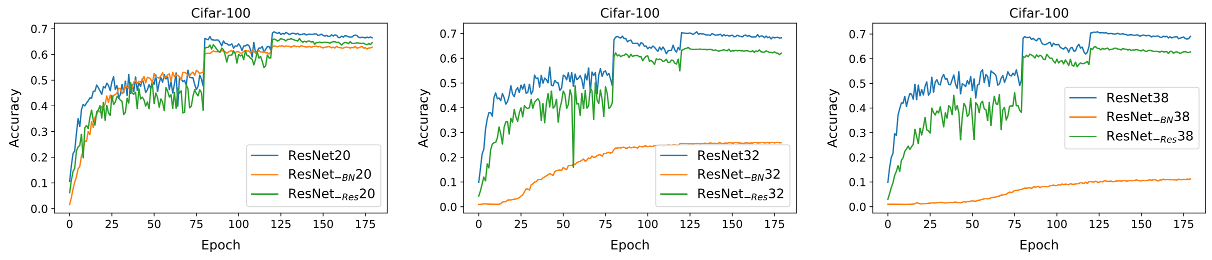

Figure E.3: Testing curve of all models reported in Table E.1. The generalization performance of models without BN (denoted as ResNet$_{-BN}$) is much worse than the baseline (denoted as ResNet). We see a similar but much smaller generalization loss when the residual connection is removed (denoted as ResNet$_{-Res}$).

E.4 Loss Landscape Details

The parametric loss landscape plots are plotted by perturbing the model parameters, θ, along the first and second top eigenvectors of the Hessian, denoted as v_1 and v_2. Then, we compute the loss of K (in our case, $K = 4096$) data points with the following formula,

$$loss = \tilde{L}(\theta + \epsilon_1 v_1 + \epsilon_2 v_2) = \frac{1}{K}\sum_{i=1}^{K} l(M(x_i), y_i; \theta + \epsilon_1 v_1 + \epsilon_2 v_2).$$

E.5 Extra Results

In the remainder of this appendix, we present additional results that we described in the main text. See Table 6.4 for a summary.

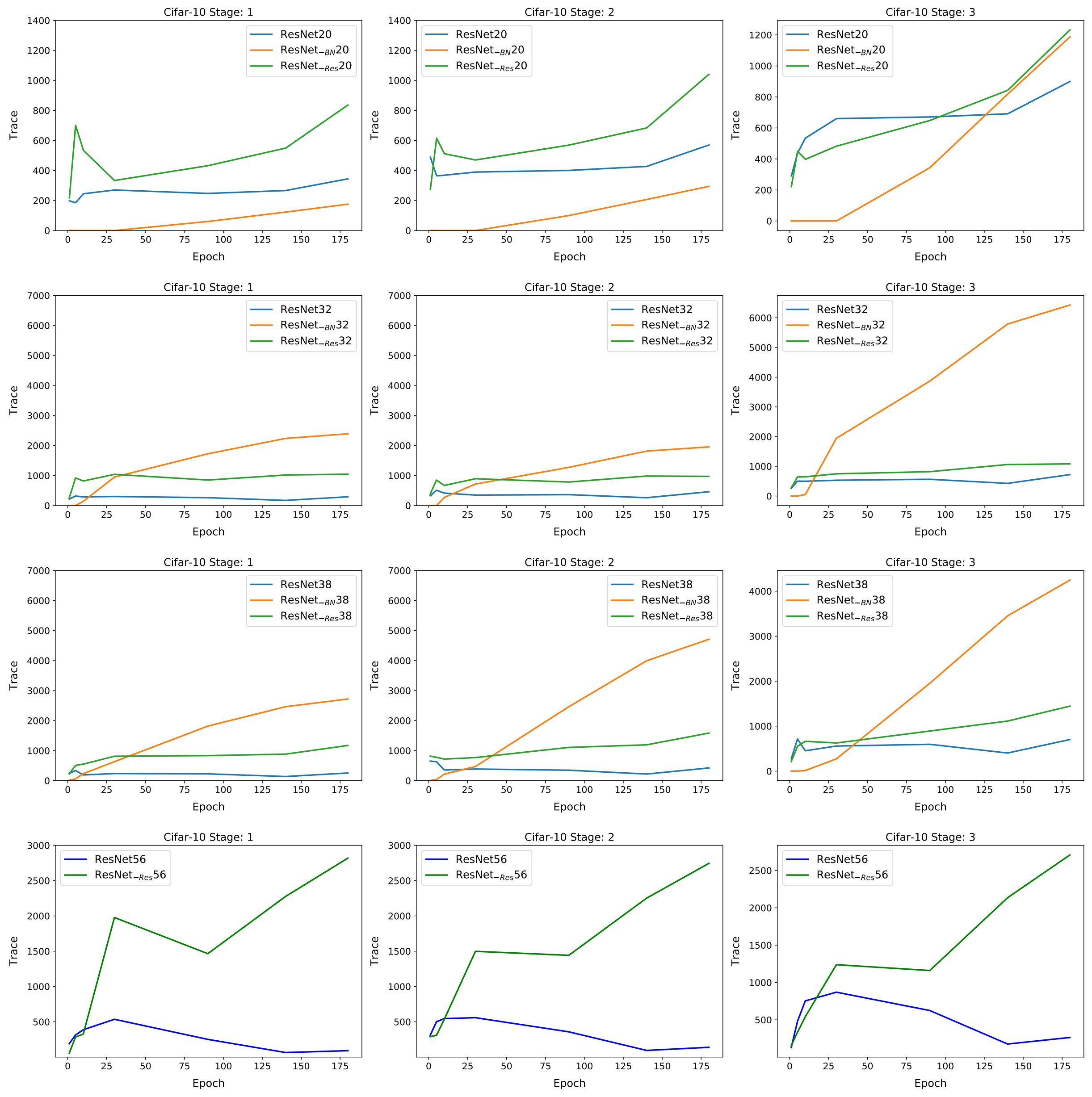

Figure E.4: Stage-wise Hessian trace of ResNet/ResNet$_{-BN}$/ResNet$_{-Res}$ with depth 20/32/38/56 on Cifar-10. See Figure E.1 for stage illustration. Removing BN layer from the third stage significantly increases the trace, compared to removing BN layer from the first/second stage. This has a direct correlation with the final generalization performance, as shown in Table 6.2.

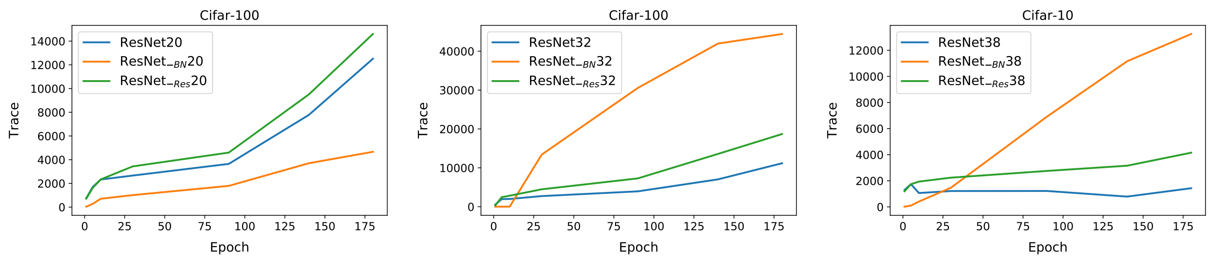

Figure E.5: The Hessian trace of the entire network for ResNet/ResNet$_{-BN}$/ResNet$_{-Res}$ with depth 20/32/38 on Cifar-100. Similar to the results for Cifar-10, shown in Figure 6.2, we see that removing the BN layer results in a rapid increase of the Hessian trace, and that removing the residual connection leads to sharper loss landscape throughout training.

Table E.2: Accuracy of ResNet models on Cifar-100 with different depths is shown in the first row. In the second through the last rows, we report the accuracy of the corresponding architectures, but with BN layer removed from one of the stages, respectively. (See Figure E.1 for stage definition.) For instance, the last row reports ResNet model with no BN layer in the third stage.

Model\Depth	20	32	38
ResNet	66.47%	68.26%	69.06%
RM BN stage 1	65.69%	65.74%	67.31%
RM BN stage 2	65.62%	64.68%	66.46%
RM BN stage 3	65.63%	64.57%	61.04%

Table E.3: Accuracy of ResNet on Cifar-100 is reported for baseline (first row), along with architectures where the residual connection is removed at different stages.

Model\Depth	20	32	38
ResNet	66.47%	68.26%	69.06%
RM Res stage 1	66.46%	66.94%	67.61%
RM Res stage 2	65.70%	66.05%	66.70%
RM Res stage 3	66.21%	66.38%	66.03%

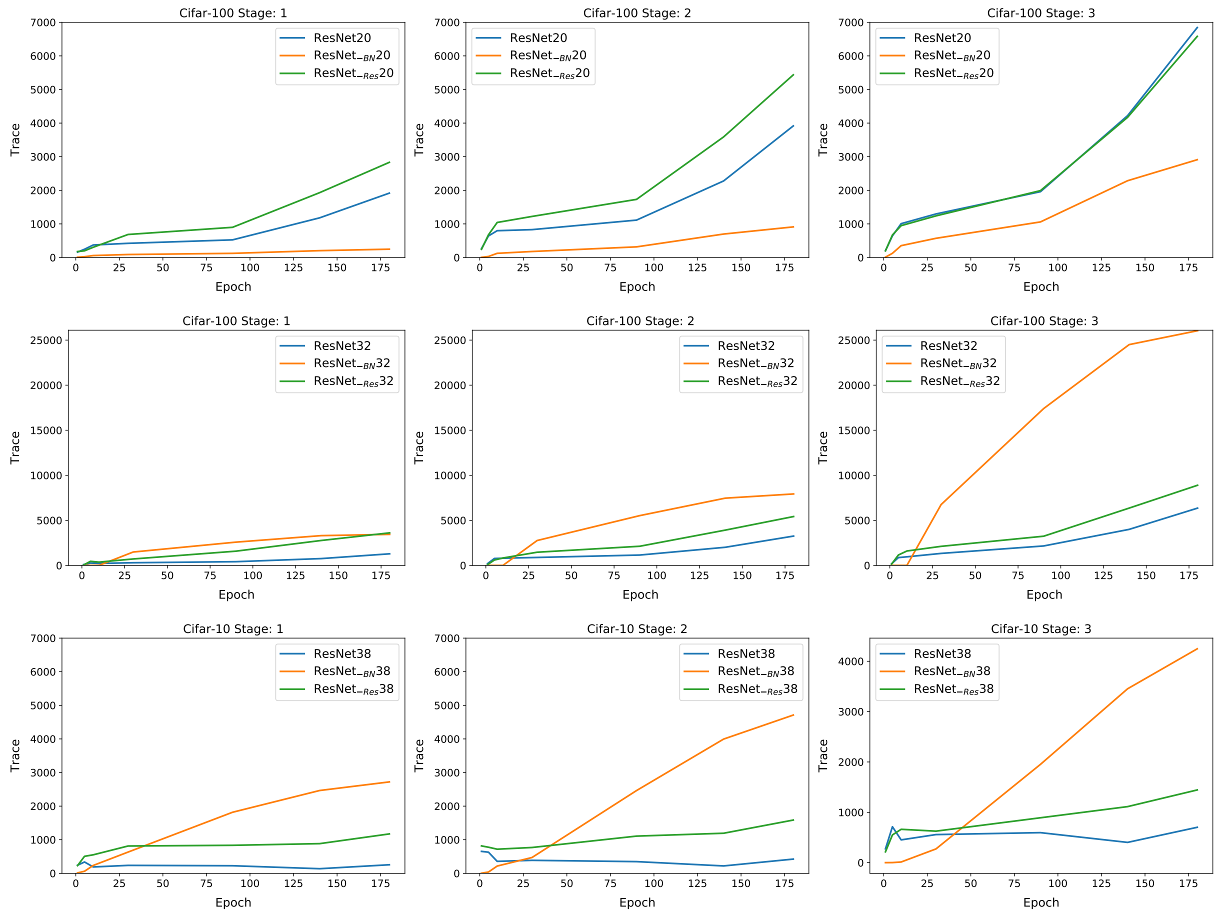

Figure E.6: Stage-wise Hessian trace of ResNet/ResNet$_{-BN}$/ResNet$_{-Res}$ with depth 20/32/38 on Cifar-100. See Figure E.1 for stage illustration. Removing BN layer from the third stage significantly increases the trace, compared to removing BN layer from the first/second stage. This has a direct correlation with the final generalization performance, as shown in Table E.2.

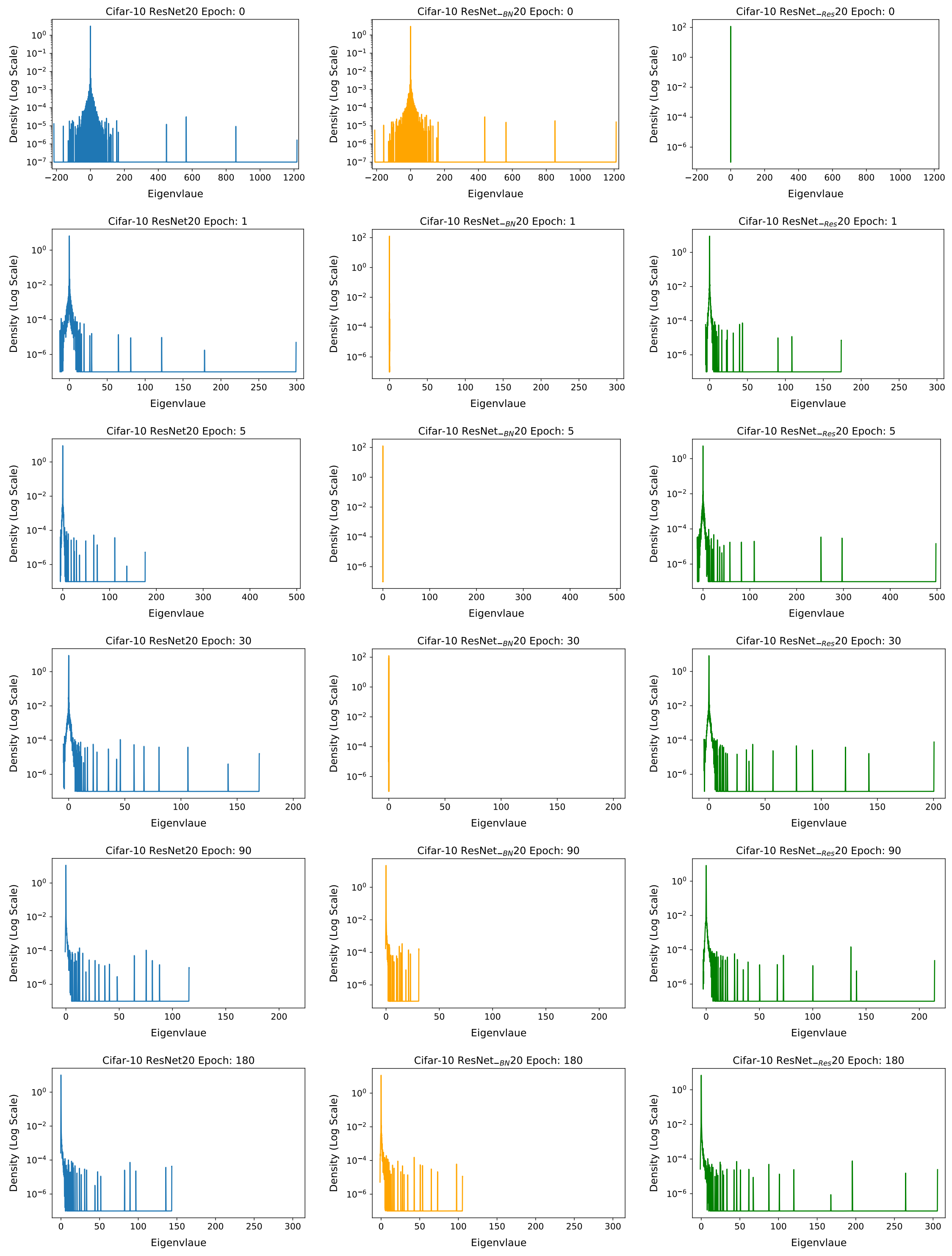

Figure E.7: Hessian ESD of the entire network for ResNets with depth 20 on Cifar-10.

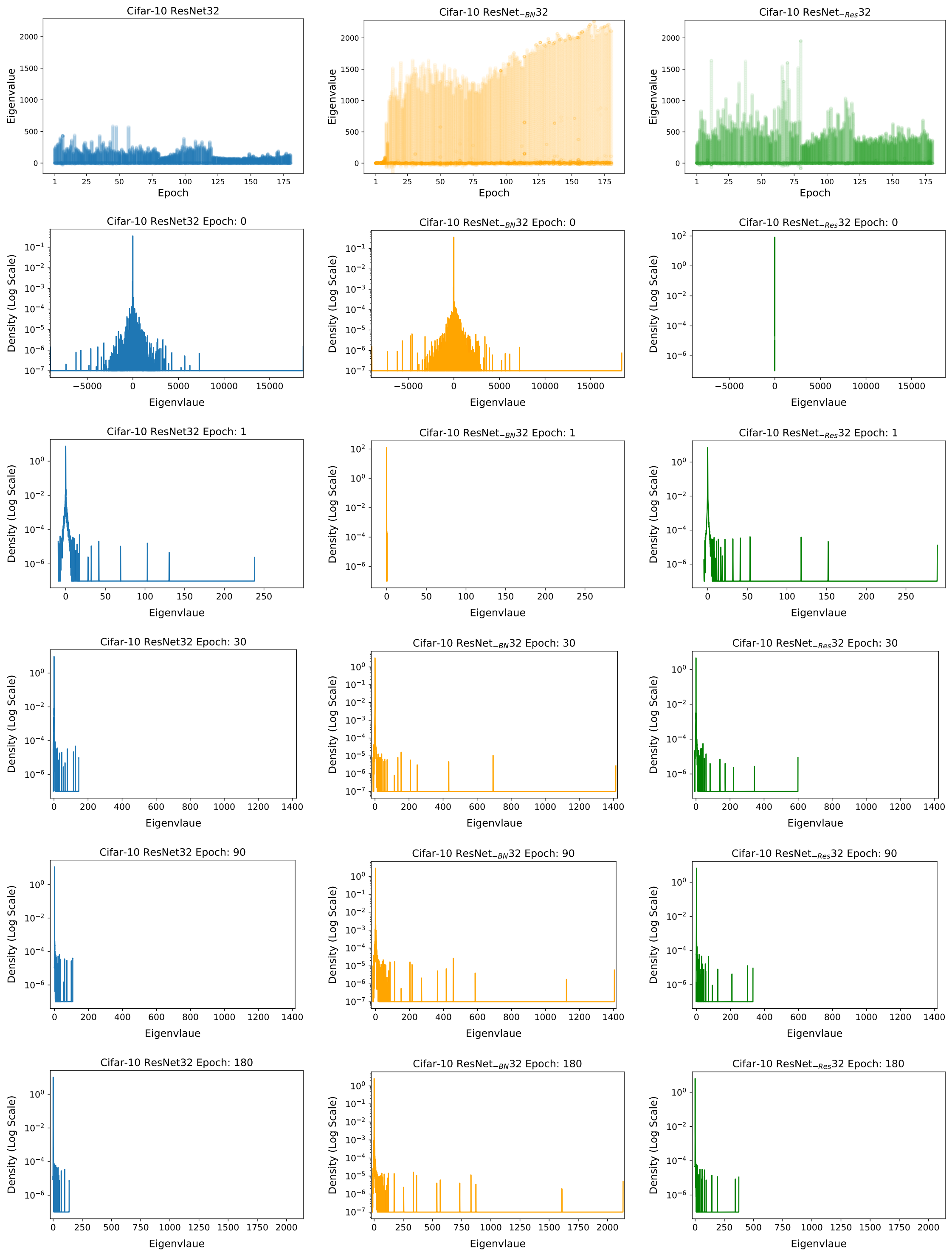

Figure E.8: Hessian ESD of the entire network for ResNets with depth 32 on Cifar-10.

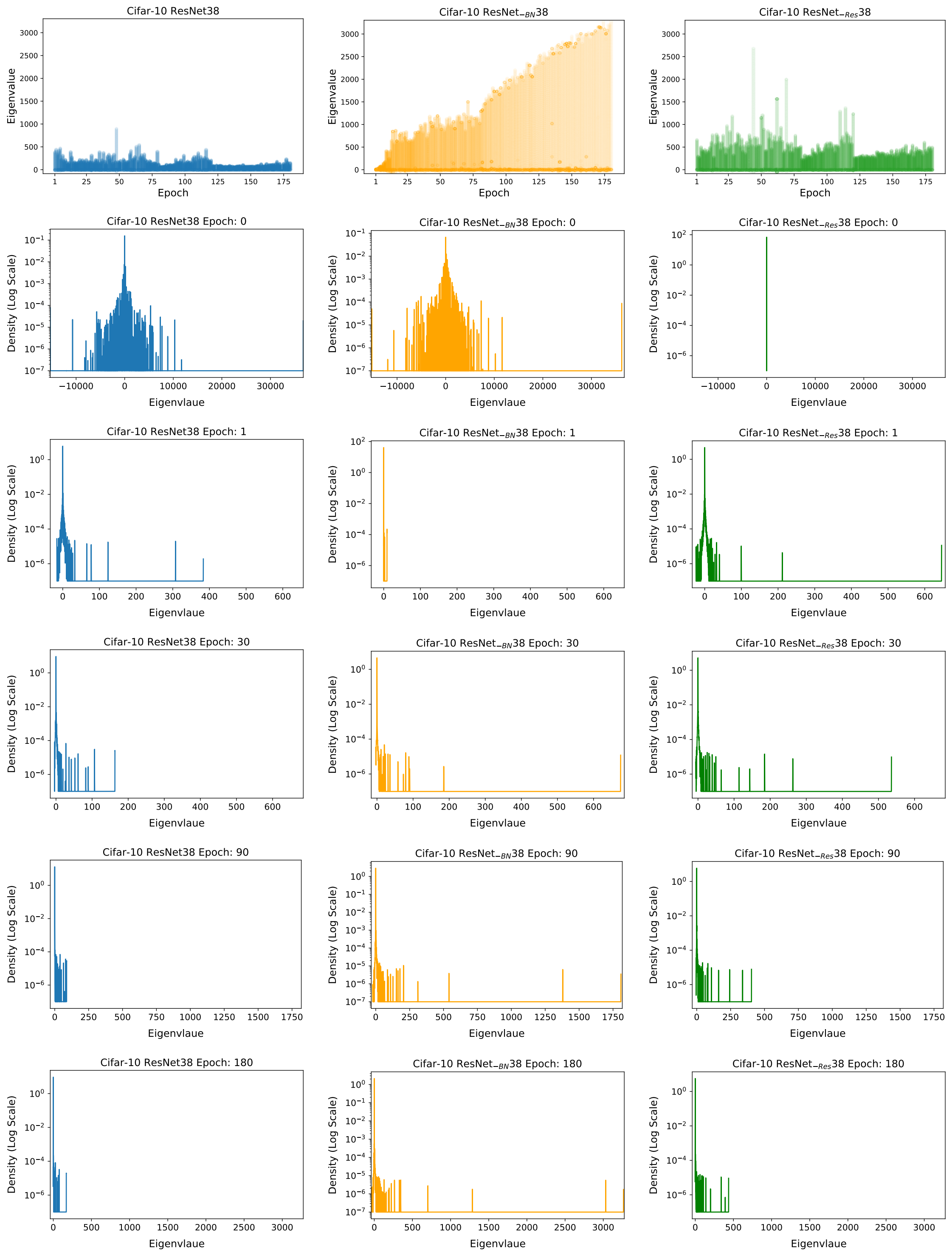

Figure E.9: Hessian ESD of the entire network for ResNets with depth 38 on Cifar-10.

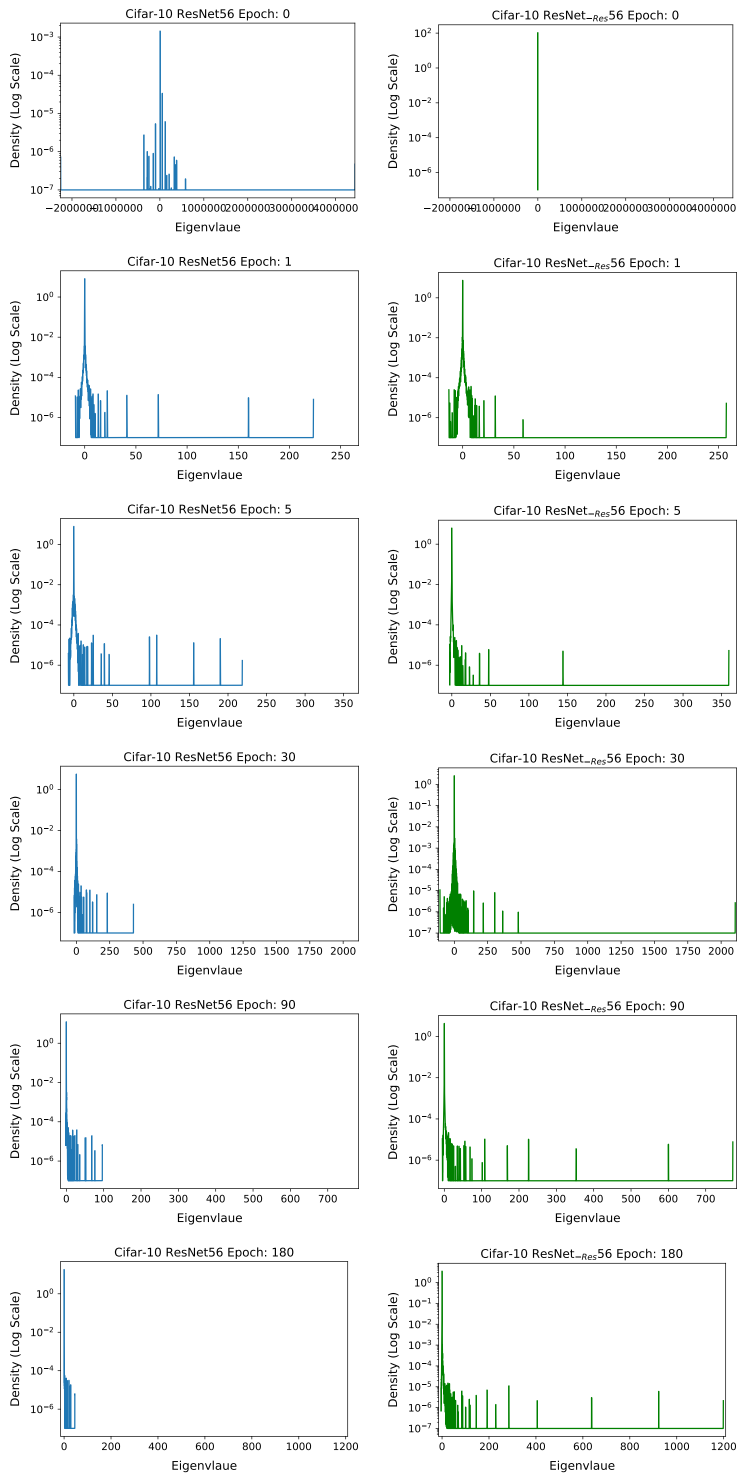

Figure E.10: Hessian ESD of the entire network for ResNets with depth 56.

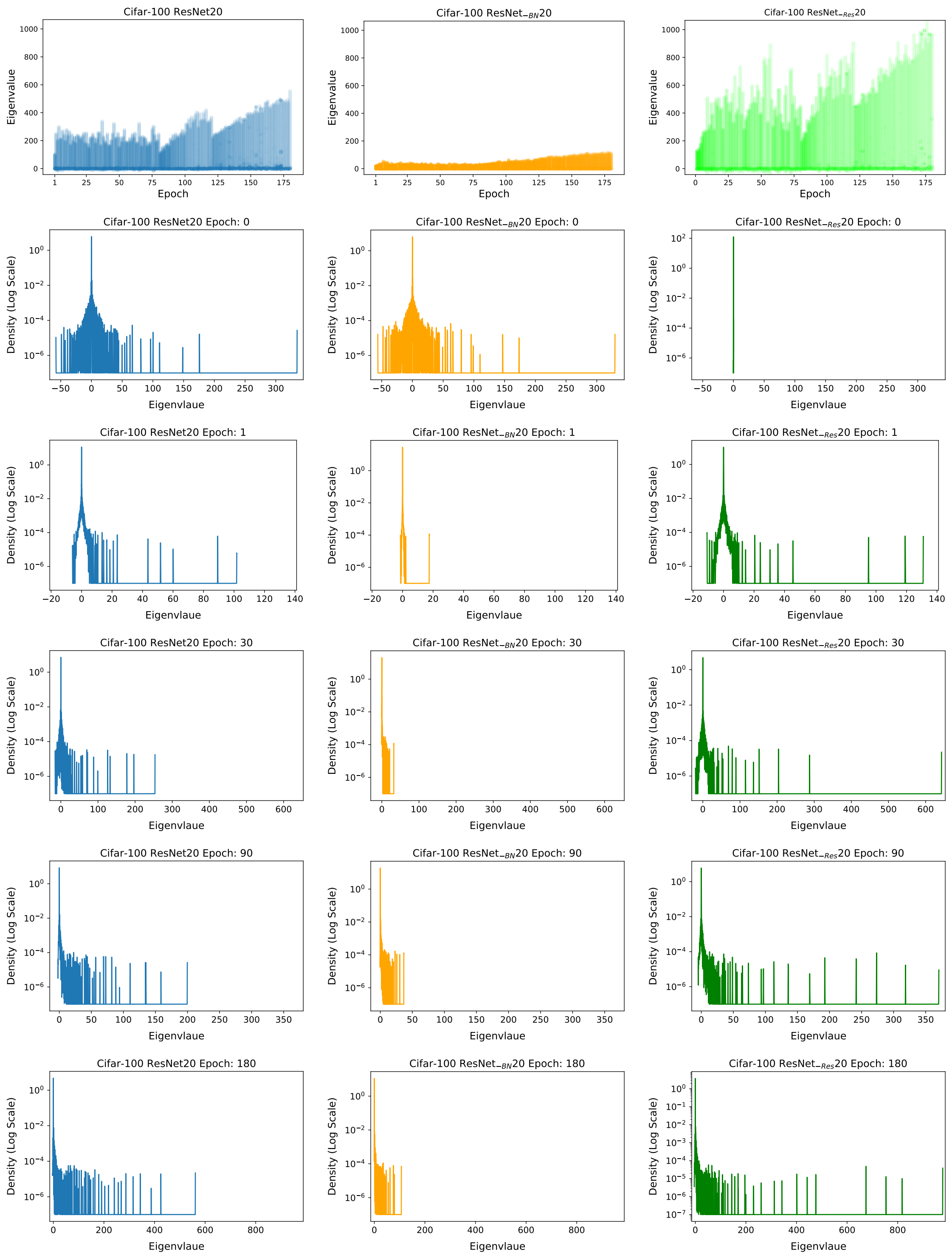

Figure E.11: Hessian ESD of the entire network for ResNets with depth 20 on Cifar-100.

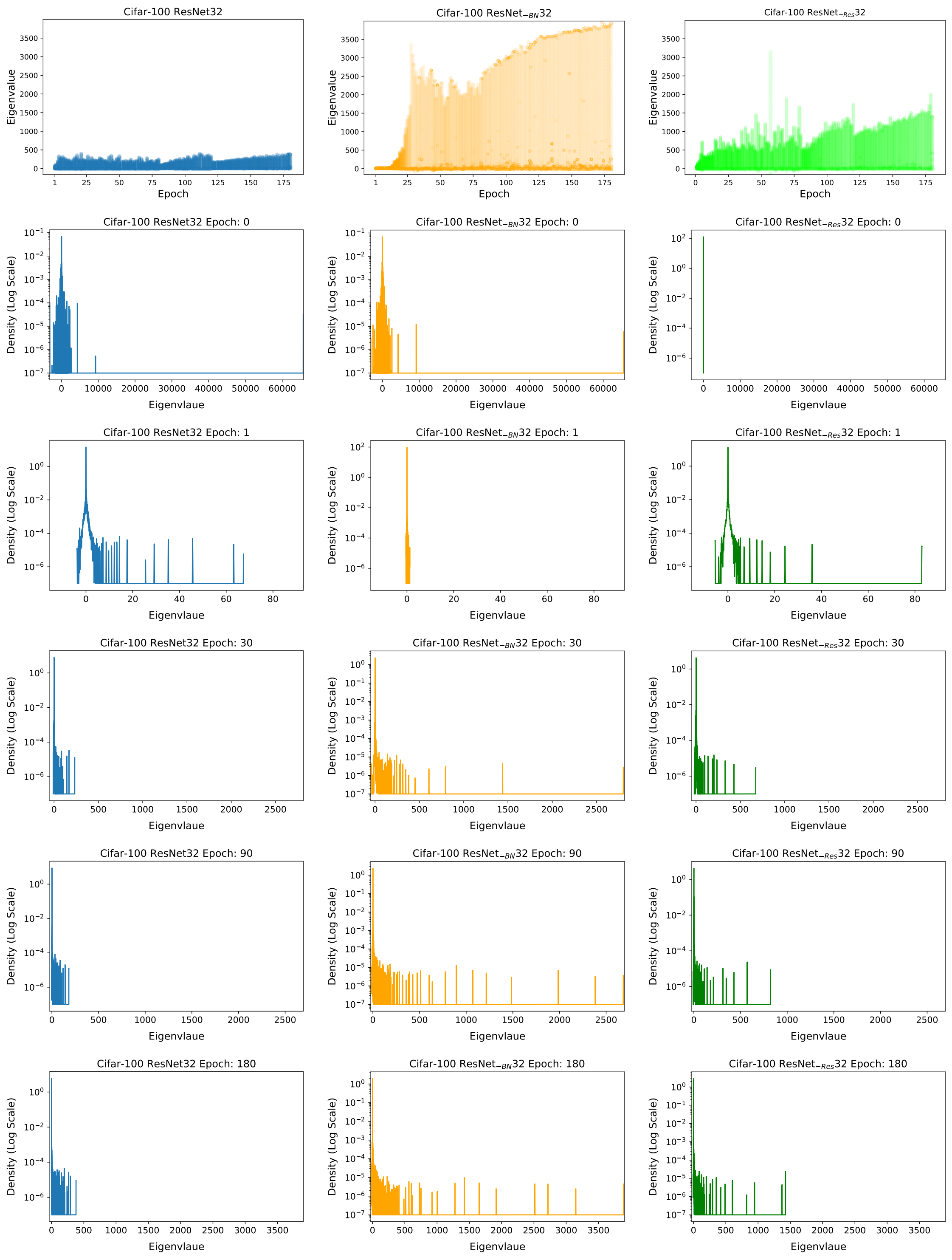

Figure E.12:   Hessian ESD of the entire network for ResNets with depth 32 on Cifar-100.

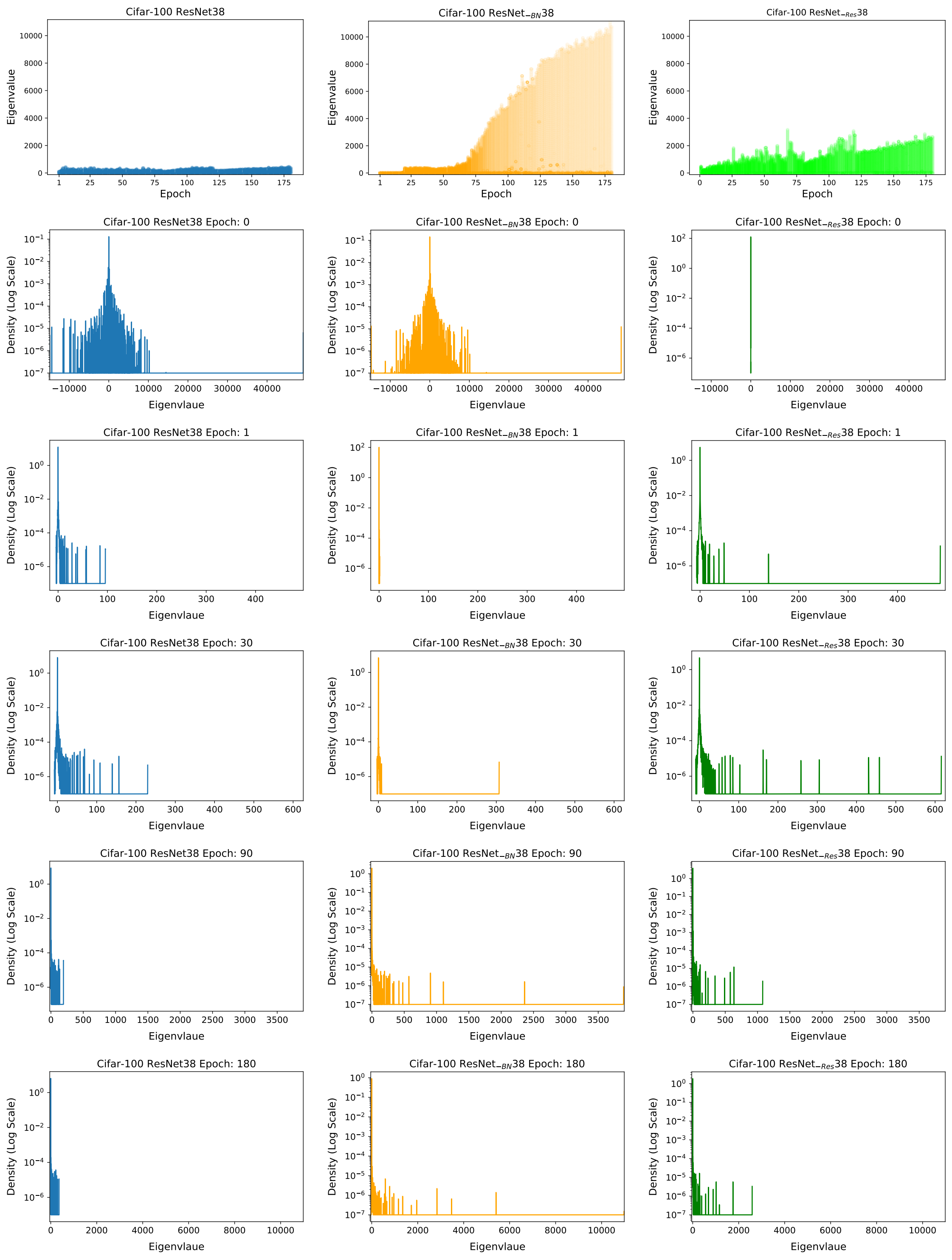

Figure E.13: Hessian ESD of the entire network for ResNets with depth 38 on Cifar-100.

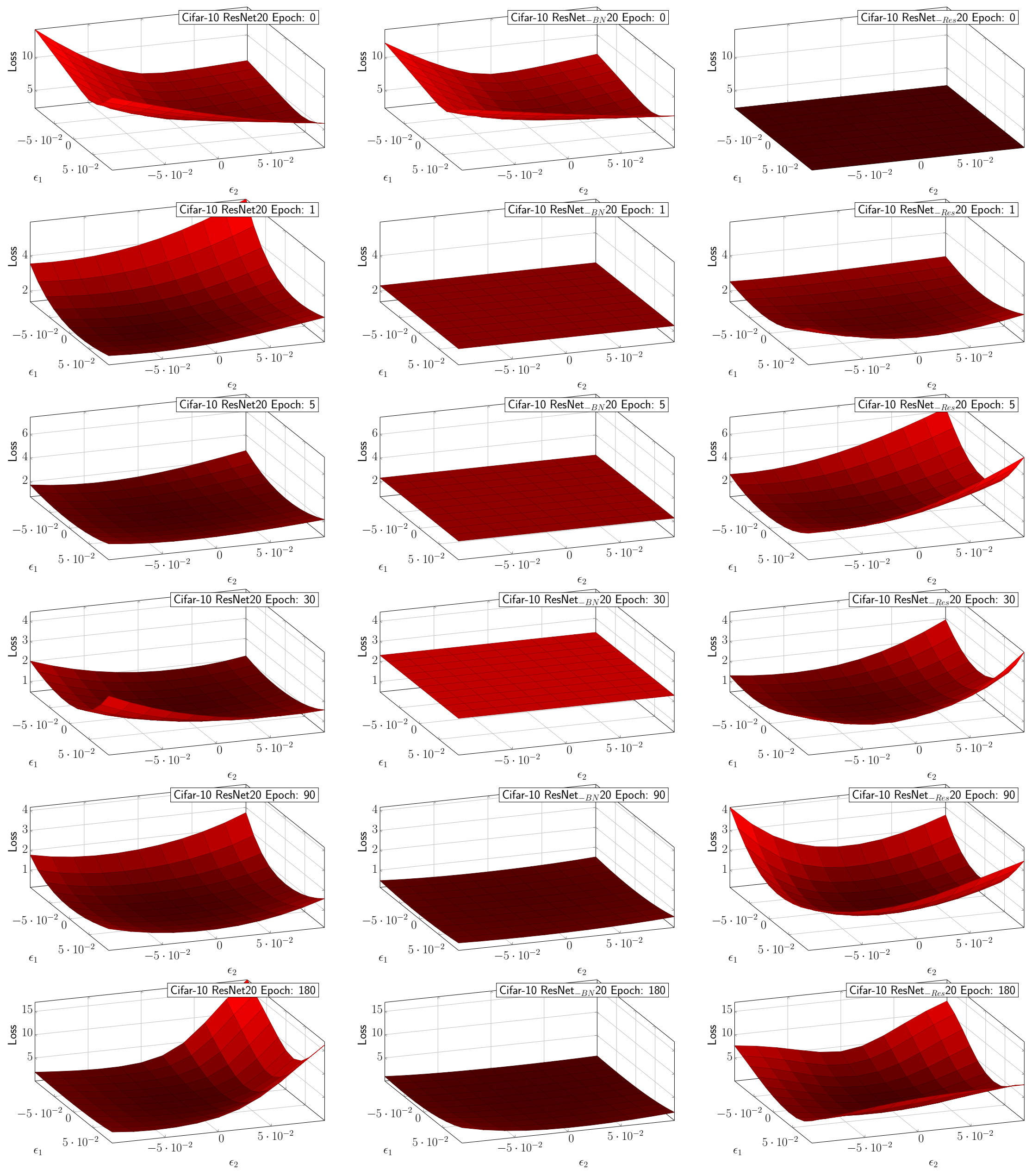

Figure E.14: Loss landscape of ResNet/ResNet$_{-BN}$/ResNet$_{-Res}$ 20 on Cifar-10 with batch size 4096 by perturbing the parameters along the first two dominant eigenvectors of the Hessian. The loss landscape of ResNet$_{-BN}$ 20 (ResNet$_{-Res}$ 20) is indeed smoother (sharper) than that of ResNet 20, which is align with the trace plot in Figure 6.2 and the Hessian ESD plot in Figure 6.3.

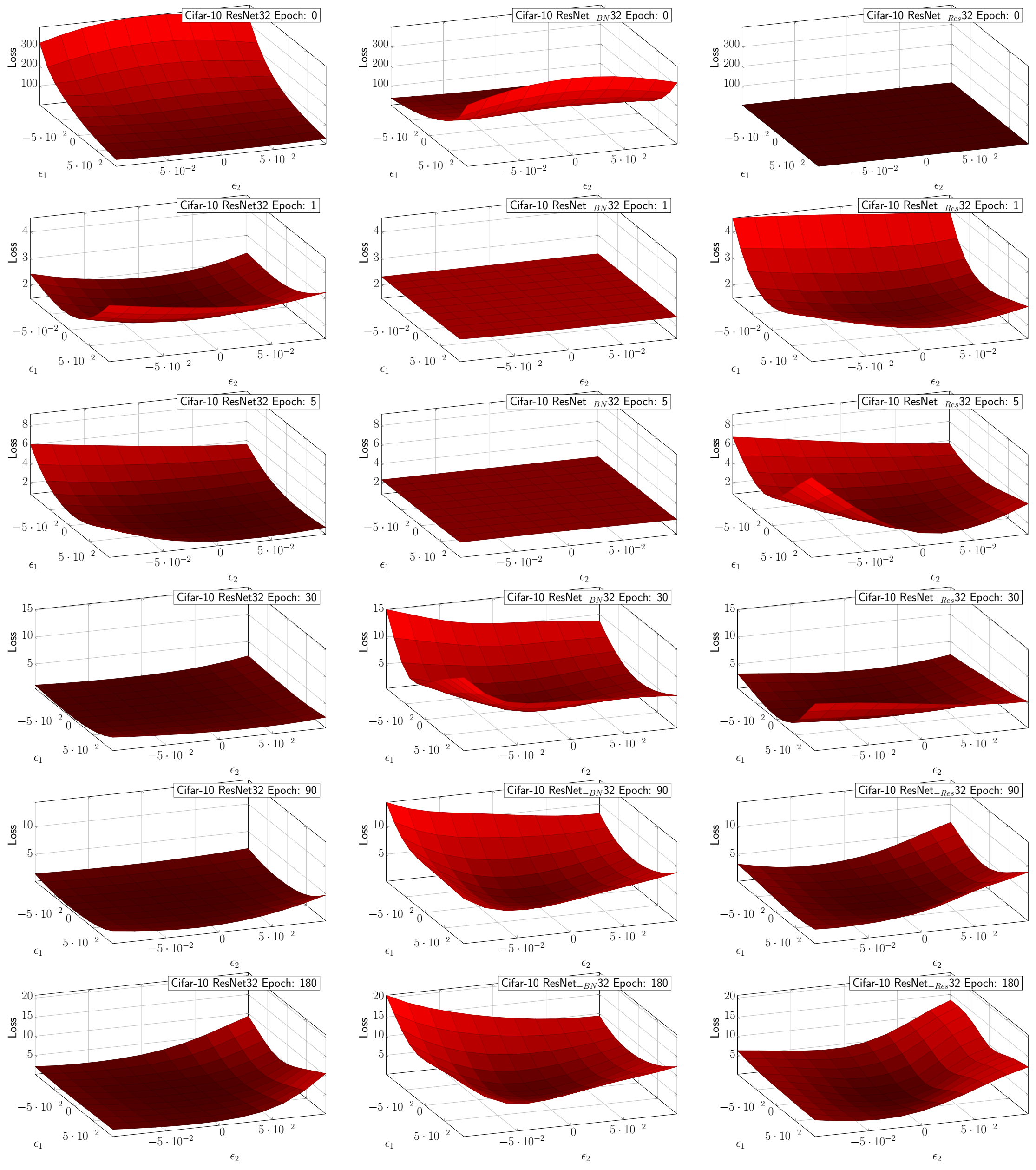

Figure E.15: Loss landscape of ResNet/ResNet$_{-BN}$/ResNet$_{-Res}$ 32 on Cifar-10 with batch size 4096 by perturbing the parameters along the first two dominant eigenvectors of the Hessian. The loss landscape of ResNet$_{-BN}$ 32/ResNet$_{-Res}$ 32 is indeed sharper than that of ResNet 32, which is align with the trace plot in Figure 6.2 and the Hessian ESD plot in Figure E.8.

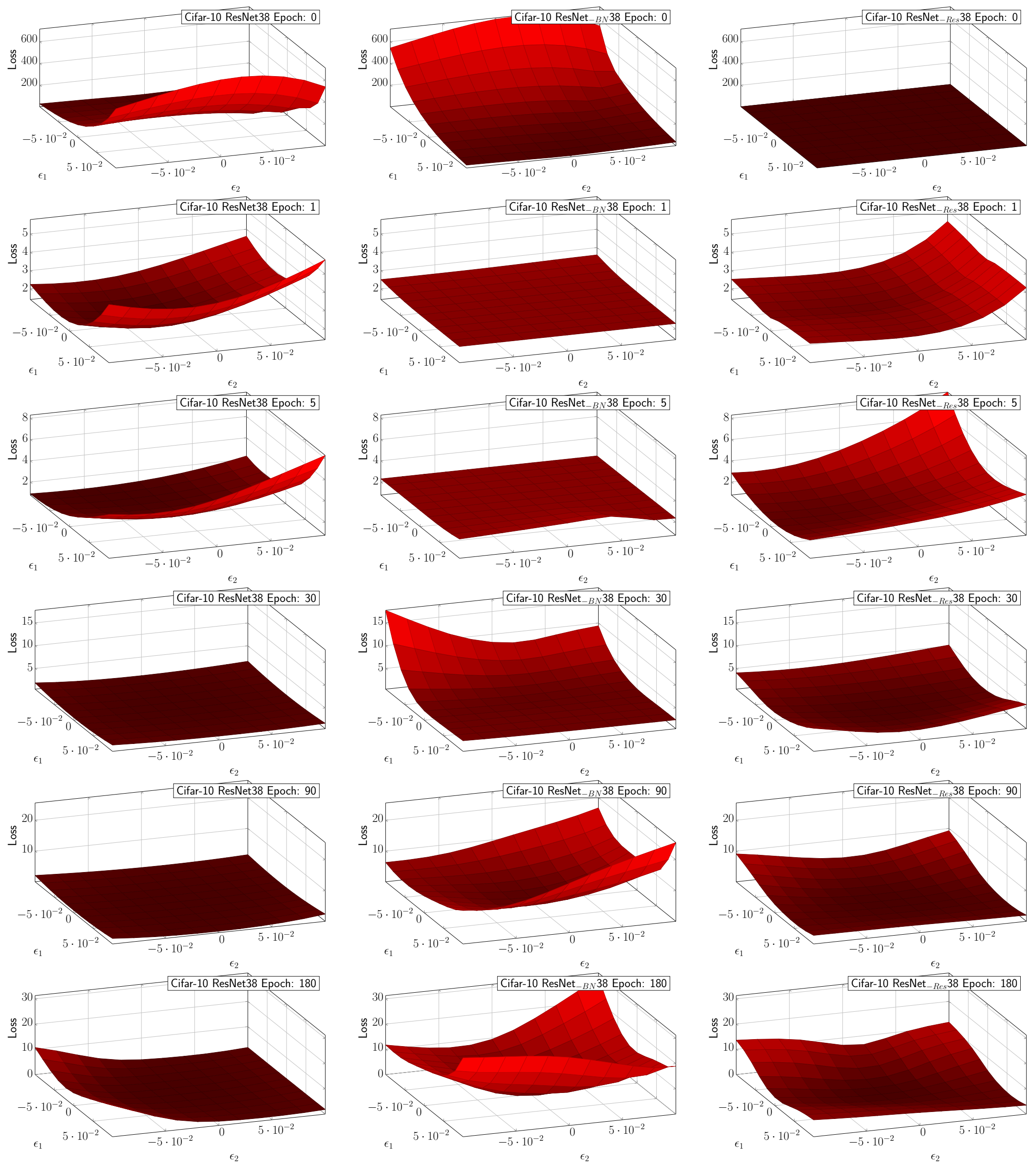

Figure E.16: Loss landscape of ResNet/ResNet$_{-BN}$/ResNet$_{-Res}$ 38 on Cifar-10 with batch size 4096 by perturbing the parameters along the first two dominant eigenvectors of the Hessian. The loss landscape of ResNet$_{-BN}$ 38/ResNet$_{-Res}$ 38 is indeed sharper than that of ResNet 38, which is align with the trace plot in Figure 6.2 and the Hessian ESD plot in Figure E.9.

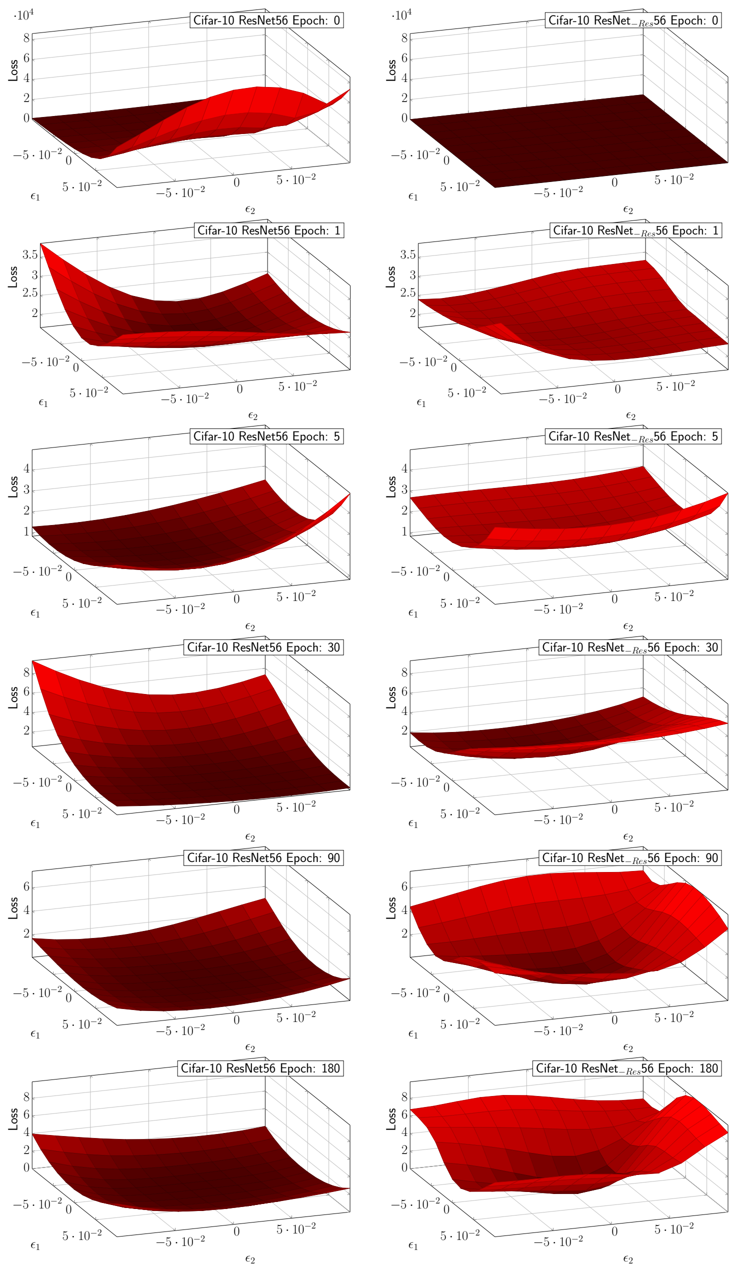

Figure E.17: Loss landscape of ResNet/ResNet$_{-Res}$ 56 on Cifar-10 with batch size 4096 by perturbing the parameters along the first two dominant eigenvectors of the Hessian. Note that the z-axis of ResNet56 at epoch 0 has different range than all the others. The loss landscape of ResNet$_{-Res}$ 56 is indeed sharper than that of ResNet 56, which is align with the trace plot in Figure 6.2 and the Hessian ESD plot in Figure E.10.

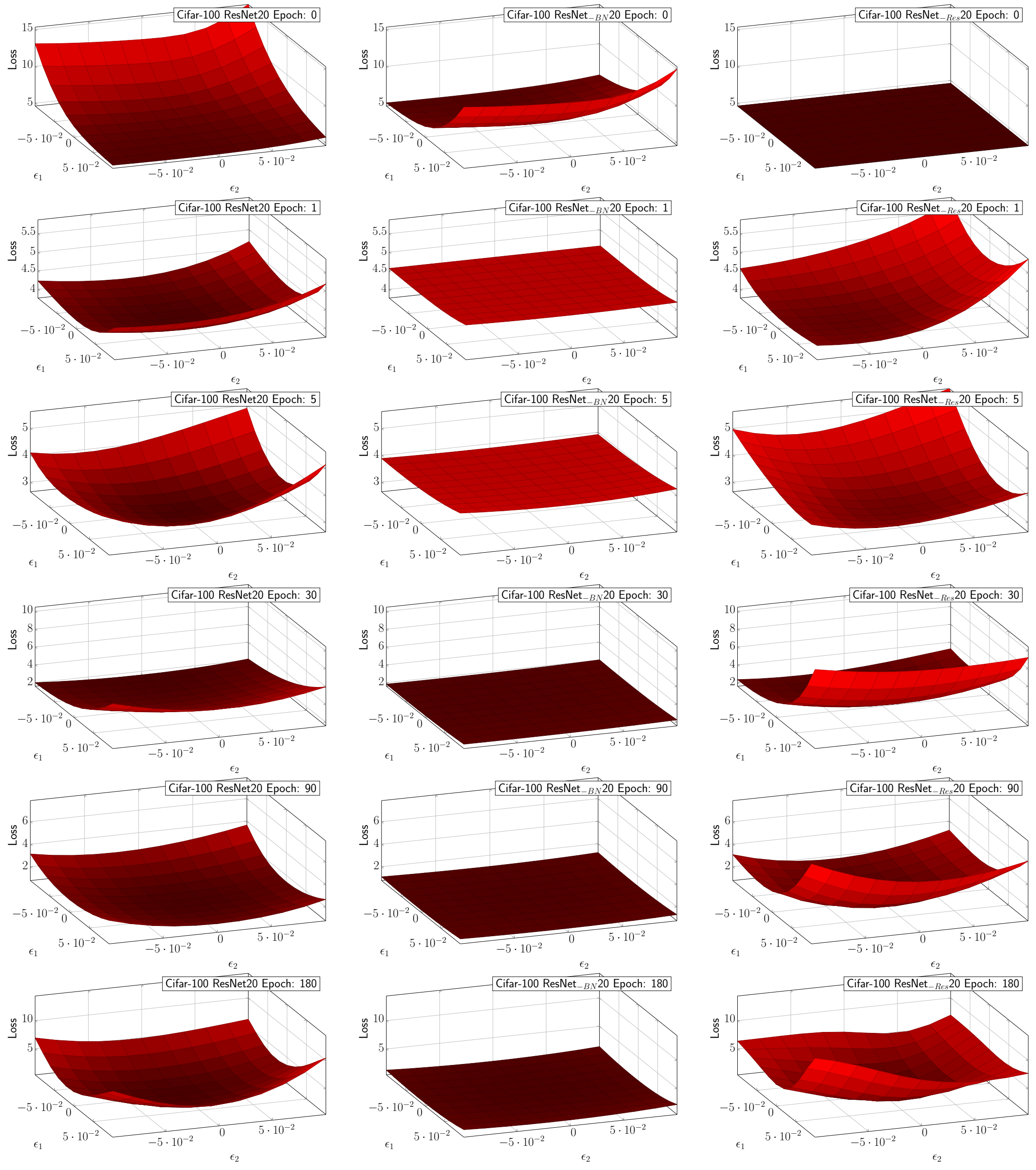

Figure E.18: Loss landscape of ResNet/ResNet$_{-BN}$/ResNet$_{-Res}$ 20 on Cifar-100 with batch size 4096 by perturbing the parameters along the first two dominant eigenvectors of the Hessian. The loss landscape of ResNet$_{-BN}$ 20 (ResNet$_{-Res}$ 20) is indeed smoother (sharper) than that of ResNet 20, which is align with the trace plot in Figure E.5 and the Hessian ESD plot in Figure E.11.

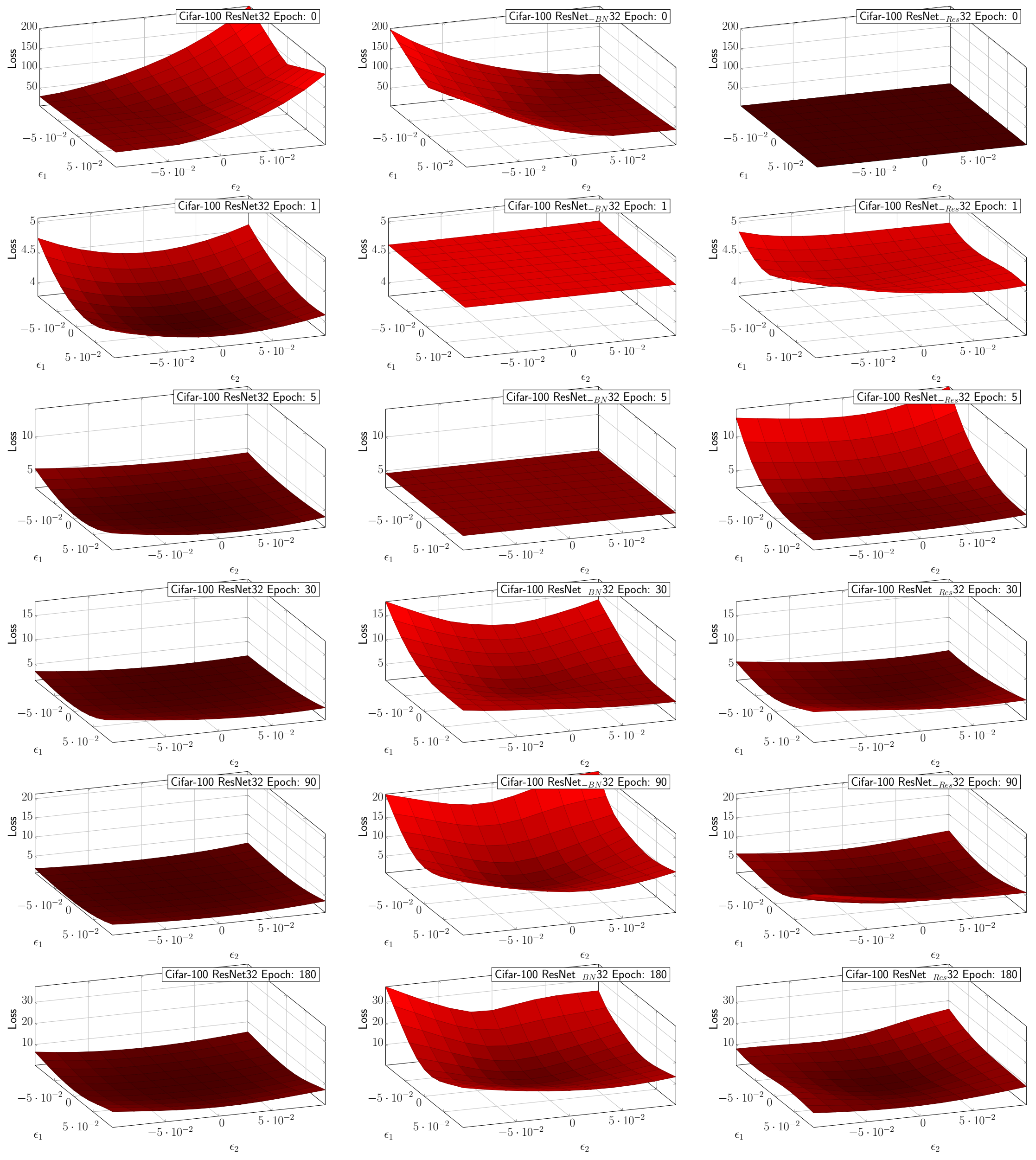

Figure E.19: Loss landscape of ResNet/ResNet$_{-BN}$/ResNet$_{-Res}$ 32 on Cifar-100 with batch size 4096 by perturbing the parameters along the first two dominant eigenvectors of the Hessian. The loss landscape of ResNet$_{-BN}$ 32/ResNet$_{-Res}$ 32 is indeed sharper than that of ResNet 32, which is align with the trace plot in Figure E.5 and the Hessian ESD plot in Figure E.12.

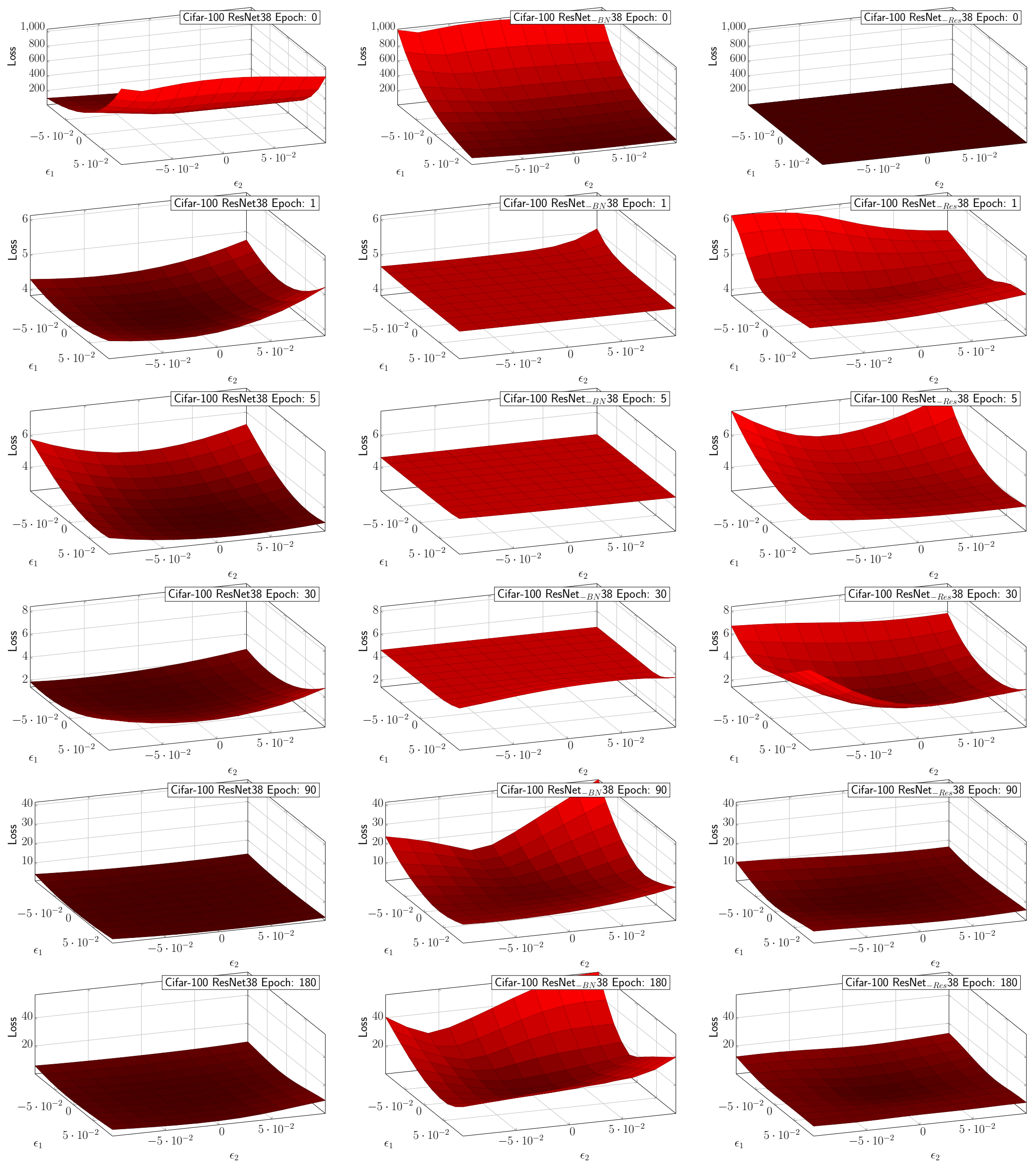

Figure E.20: Loss landscape of ResNet/ResNet$_{-BN}$/ResNet$_{-Res}$ 38 on Cifar-100 with batch size 4096 by perturbing the parameters along the first two dominant eigenvectors of the Hessian. The loss landscape of ResNet$_{-BN}$ 38/ResNet$_{-Res}$ 38 is indeed sharper than that of ResNet 38, which is align with the trace plot in Figure E.5 and the Hessian ESD plot in Figure E.13.

Appendix F

Appendix for Chapter 7

F.1 Deployment Frameworks

A number of frameworks [122, 50, 1, 210, 178, 97, 231, 51] have been developed for deep learning. Many [122, 50, 1, 178] offer a dataflow DAG abstraction for specifying NN workloads and provide optimization support for inference as well as training with automatic differentiation. These frameworks significantly reduce development cycles for deep learning algorithms and thus facilitate innovations in deep learning. However, a majority of these frameworks [122, 50, 178] adopt a library-based approach that maps the NN operations to hardware through existing high-performance libraries, such as cuDNN [53] for GPUs, and GEMMLOWP [119] and NNPACK [74] for CPUs. These libraries currently do not support low-precision inference (INT4), and since they are not open source we could not add that functionality. As such, for our analysis we adopted to use TVM [51], which provides a general graph and a tensor expression intermediate representation (IR) to support automatic code transformation and generation. TVM also equips a QNN dialect [121] to compile the quantization-specific operators of a quantized model. We choose TVM as our deployment framework for several reasons including: (i) its extensive support in the frontend high-level frameworks and the backend hardware platforms; and (ii) its decoupled IR abstraction that separates the algorithm specifications and the scheduling decisions. Augmenting TVM with our mixed-precision quantization support allows this optimization to be used by NNs written in different frameworks as well as for various target hardware platforms. In addition, the decoupled IR design in TVM allows the mixed-precision quantization optimization to be applied without affecting the specification of algorithms.

F.2 Quantization Method

Symmetric and Asymmetric Quantization. For uniform quantization, the scaling factor S is chosen to equally partition the range of real values r for a given bit width:

$$S = \frac{r_{max} - r_{min}}{2^b - 1},$$

where r_{max}, r_{min} denotes the max/min value of the real values, and b is the quantization bit width. This approach is referred to as *asymmetric quantization*. It is also possible to use a *symmetric quantization* scheme where $S = 2\max(|r_{max}|, |r_{min}|)/(2^b - 1)$ and $Z = 0$ (since zero will be exactly represented). As such, the quantization mapping can be simplified as:

$$Q(r) = \text{Int}\left(\frac{r}{S}\right). \tag{F.1}$$

Conversely, the real values r could be recovered from the quantized values $Q(r)$ as follows:

$$\tilde{r} = S\ Q(r). \tag{F.2}$$

Note that the recovered real values $\tilde{r}$ will not exactly match r due to the rounding operation. For HAWQ-V3, we use symmetric quantization for weights and asymmetric quantization for the activations.

Static and Dynamic Quantization. The scaling factor S depends on r_{max} and r_{min}. These can be precomputed for weights. However, for activations, each input will have a different range of values across the NN layers. In dynamic quantization, this range and the corresponding scaling factor is computed for each activation map during runtime. However, computing these values during inference has high overhead. This can be addressed with static quantization, in which this range is pre-calculated during the quantization phase and made independent of the input data, by analyzing the range of activations for different batches. We use static quantization for all of the experiments with HAWQ-V3. With these definitions, we next discuss how quantized inference is performed.

F.3 Fake Quantization for Convolution

In simulated quantization (also referred to as fake quantization in literature), all the calculations happen in FP32, which is different from the approach we used in Section 7.3.1. Similar to Section 7.3.1, suppose that the hidden activation is $h = S_h q_h$ and weight tensor is $W = S_w q_w$. In fake quantization, the output is calculated as:

$$a = (S_w q_w) * (S_h q_h). \tag{F.3}$$

That is the weight and activation are first represented back to FP32 precision, and then the calculation is performed. This result is then requantized and sent to the next layer as follows:

$$q_a = \text{Int}\left(\frac{a}{S_a}\right), \tag{F.4}$$

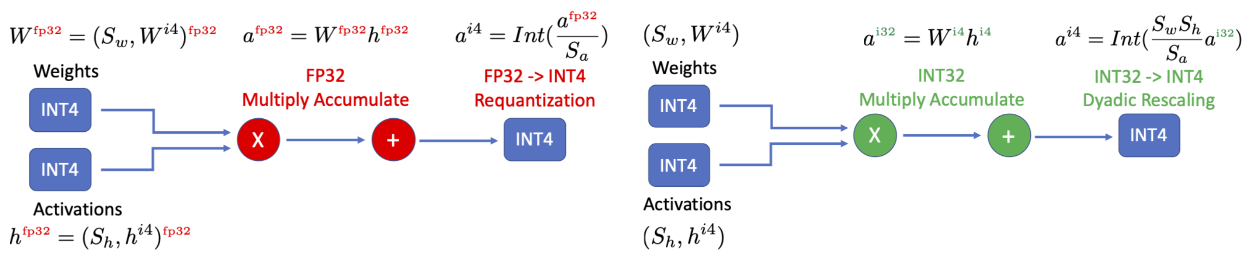

Figure F.1: Illustration of fake vs true quantization for a convolution (fully-connected) layer. (Left) In the simulated quantization (aka fake quantization), weights and activations are simulated as integers with floating point representation, and all the multiplication and accumulation happens in FP32 precision. However, with this approach, one cannot benefit from low-precision ALUs. (Right) An illustration of the integer-only pipeline with integer-only quantization. Note that with this approach, all the weights and activations are stored in integer format, and all the multiplications are performed with INT4 and accumulated in INT32 precision. Finally, the accumulated result is requantized to INT4 with dyadic scaling (denoted by $(\frac{S_w S_h}{S_a})$). Importantly, no floating point or even integer division is performed.

where S_a is the pre-calculated scale factor for the output activation. However, notice that here the requantization operation requires FP32 arithmetic (division by S_a), which is different from HAWQ-V3's Dyadic arithmetic that only uses integer operations. Figure F.1 shows the illustration of fake vs true quantization for a convolution (fully-connected) layer, without the BN layer. We also showed the corresponding illustration when BN is used in Figure 7.1.

F.4 Batch Normalization Fusion

During inference, the mean and standard deviation used in the BN layer are the running statistics (denoted as μ and σ). Therefore, the BN operation can be fused into the previous convolutional layer. That is to say, we can combine BN and CONV into one operator as,

$$\begin{aligned} \text{CONV\_BN}(h) &= \beta\frac{Wh-\mu}{\sigma}+\gamma \\ &= \frac{\beta W}{\sigma}h + (\gamma - \frac{\beta\mu}{\sigma}) \equiv \bar{W}h+\bar{b}, \end{aligned} \tag{F.5}$$

where W is the weight parameter of the convolution layer and h is the input feature map. In HAWQ-V3, we use the fused BN and CONV layer and quantize $\bar{W}$ to 4-bit or 8-bit based on the setting, and quantize the bias term, $\bar{b}$ to 32-bit. More importantly, suppose the scaling factor of h is S_h and the scaling factor of $\bar{W}$ is $S_{\bar{W}}$. The scaling factor of $\bar{b}$ is enforced to be

$$S_{\bar{b}} = S_h S_{\bar{W}}. \tag{F.6}$$

So that the integer components of $\bar{W}h$ and $\bar{b}$ can be directly added during inference.

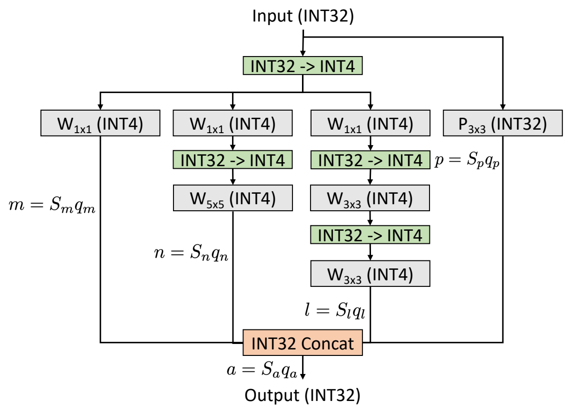

Figure F.2: Illustration of HAWQ-V3 for an inception module. Input feature map is given in INT32 precision, which is requantized to INT4 precision (green boxes) before being passed to the three convolutional branches. The pooling layer, however, is performed on the original input feature map in INT32. This is important since performing pooling on 4-bit data can result in significant information loss. The outputs for all the branches are scaled and requantized before being concatenated.

F.5 Concatenation Layer

The concatenation operation in Inception is an important component, which needs to be quantized carefully to avoid significant accuracy degradation. Concatenation layers are often used in the presence of pooling layers and other convolutions (a good example is the inception family of NNs). In HAWQ-V3, we use INT32 for the pooling layer since performing pooling on 4-bit can result in significant information loss. Furthermore, we perform separate dyadic arithmetic for the following concatenation operator in the inception module. Suppose the input of a concatenation block is denoted as $h = S_h q_h$, the output of the three convolutional branches are $m = S_m q_m$, $n = S_n q_n$, and $l = S_l q_l$, the output of the pooling branch is $p = S_p q_p$, and the final output is $a = S_a q_a$.

The pooling branch directly takes h as input, and the rest of the three convolutional branches take the quantized 4-bit tensor as input. After the computation of four separate branches, the output q_a is calculated with four DN operators:

$$q_a = \sum_{i \in \{m,n,l\}} \mathrm{DN}\left(\frac{S_i}{S_a}\right) q_i + \mathrm{DN}\left(\frac{S_p}{S_a}\right) q_p. \tag{F.7}$$

This scheme is represented in Figure F.2.

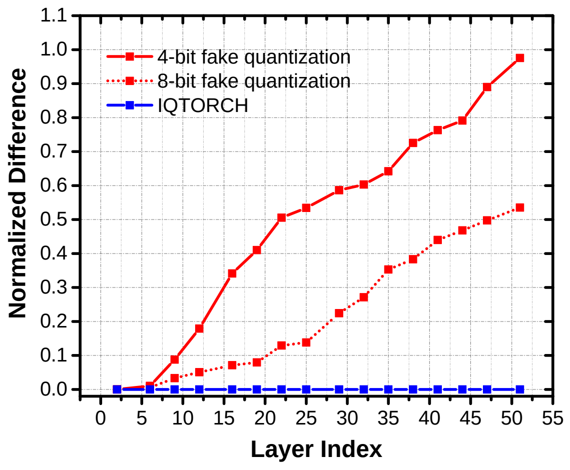

Figure F.3: The normalized difference between activation tensors in TVM and activation tensors in PyTorch during inference. The normalized difference is the L_2 norm of the difference between two activation counterparts divided by the L_2 norm of the TVM activation tensor.

F.6 Fake Quantization for Residual Connection

Similar to Section 7.3.3, Let us denote the activation passing through the residual connection as $r = S_r q_r$. the activation of the main branch before residual addition as $m = S_m q_m$. the final output after residual accumulation as $a = S_a q_a$. In fake quantization, the output a is calculated in FP32 as,

$$a = S_r q_r + S_m q_m. \tag{F.8}$$

Afterwards, requantization is performed,

$$q_a = \mathrm{Int}(\frac{S_r q_r + S_m q_m}{S_a}), \tag{F.9}$$

where the Int operator requires FP32 multiplication.

Similarly, fake quantization for concatenation layer is calculated as (see Appendix F.5 for notations):

$$q_a = \mathrm{Int}(\frac{m + n + l + p}{S_a}). \tag{F.10}$$

F.7 Error Accumulation of Fake Quantization

There has been a common misunderstanding that using fake quantization is acceptable since one can use FP32 precision to perform Integer operations exactly. First, this is only true if the matrix multiplications only use integer numbers, without using very large numbers. The latter is the case in most ML applications. However, the problem is that many quantization approaches use fake quantization in a way that is different than the above argument.

For example, keeping the BN parameters in FP32 and not quantizing them is a major problem. It is not possible to simply ignore that and deploy a quantized model with FP32 BN parameters on integer-only hardware. This difference was discussed and illustrated in Figure 7.1.

Another very important subtle issue is how the residual connection is treated. As discussed in the previous section, the fake quantization approaches use FP32 arithmetic to perform the residual addition. The common (but incorrect) the argument here again is that the INT arithmetic can be performed without error with FP32 logic. However, this is not the problem, since there is a subtle difference in how requantization is performed. In fake quantization, the results are first accumulated in FP32 and then requantized. However, it is not possible to perform such an operation on integer-only hardware. In integer-only hardware, the results are always quantized and then accumulated. This difference can actually lead to O(1) error.

For example consider the following case: assume $S_a = 1$, $r = 2.4$, $m = 4.4$ (see definition in Appendix F.6), and the requantization operator (Int) uses the "round to the nearest integer". Then using fake quantization, the output q_a is

$$q_a = \mathrm{Int}(4.4 + 2.4) = 7. \tag{F.11}$$

However for true quantization, the output q_a is

$$q_a = \mathrm{Int}(4.4) + \mathrm{Int}(2.4) = 6. \tag{F.12}$$

This is an O(1) error that will propagate throughout the network. Also note that the problem will be much worse for low precision error. This is because an O(1) error for INT8 quantization is equivalent to a constant times (1/256), while for INT4 quantization it will be a constant times (1/16).

We also performed a realistic example on ResNet50 for the uniform quantization case. We perform fake quantization in PyTorch for fine-tuning and then deploy the model in TVM using integer-only arithmetic. Afterward, we calculate the error between the feature map between PyTorch (fake quantization) and TVM (integer-only). In particular, we measure the normalized difference using L_2 norm:

$$\mathrm{Normalized\_Difference} = \frac{\|x_1 - x_2\|}{\|x_1\|}, \tag{F.13}$$

where x_1, x_2 are the feature maps with fake quantization and the corresponding values calculated in hardware with integer-only arithmetic. In Figure F.3 we show the normalized difference between activation tensors in TVM and activation tensors in PyTorch during inference. As one can see, the numerical differences of the first layers are relatively small. However, this error accumulates throughout the layers and becomes quite significant in the last layers. Particularly, for uniform 4-bit quantization, the final difference becomes $> 95\%$.

F.8 Implementation Details

Models. All the empirical results are performed using pretrained models from PyTorchCV [187] library. In particular, we do not make any architectural changes to the models, even though doing so might lead to better accuracy. We consider three NN models, ResNet18, ResNet50, and InceptionV3, trained on the ImageNet dataset [65]. For all the NNs, we perform BN folding to speed up the inference. All the calculations during inference are performed using dyadic arithmetic (i.e., integer addition, multiplication, and bit shifting), with no floating point or integer division anywhere in the network, including requantization stages.

Training details. We use PyTorch (version 1.6) for quantizing models with HAWQ-V3. For all the quantization results, we follow the standard practice of keeping the first and last layer in 8-bit (note that input data is encoded with 8-bits for the RGB channels, which is quantized with symmetric quantization). We only use uniform quantization along with channel-wise symmetric quantization for weights, and we use layer-wise asymmetric quantization for activations. In order to perform static quantization, we set our momentum factor of quantization range (i.e., minimum and maximum) of activations to be 0.99 during training. Although further hyperparameter tuning may achieve better accuracy, for uniformity, all our experiments are conducted using learning rate 1e-4, weight decay 1e-4, and batch size 128.

Distillation. As pointed out previously [185], for extra-low bit quantization (in our case uniform 4 bit and mixed 4/8 bit quantization), distillation may alleviate the performance degradation from quantization. Therefore, in addition to our basic results, we also present results with distillation (denoted with HAWQ-V3+DIST). Among other things, we do confirm the findings of previous work [185] that distillation can boost the accuracy of quantized models. For all different models, we apply ResNet101 [104] as the teacher, and the quantized model as the student. For simplicity, we directly use the naive distillation method proposed in [107]. (More aggressive distillation or fine-tuning with hyperparameter may lead to better results.)

Latency Measurement. We use TVM to deploy and tune the latency of the quantized models using Google Cloud Platform virtual machines with Tesla T4 GPUs and CUDA 10.2. We build the same NN models in TVM and tune the layerwise performance by using the autotuner. Once we have the tuned models, we run the end-to-end inference multiple times to measure the average latency. For the accuracy test, we load the parameters trained from PyTorch and preprocess it to the corresponding data layout that TVM requires. Then, we do inference in TVM and verify that the final accuracy matches the results in PyTorch.

Mixed-precision configuration. For mixed-precision configuration, we first compute the trace of each layer [69] using PyHessian [254], and then solve the ILP problem using PULP [198]. Our mixed-precision ILP problem can find the right bit-precision configuration with orders of magnitude faster run time, as compared to the RL based method [235, 242]. For instance, the entire trace computation can be finished within 30 minutes for all layers of ResNet50/InceptionV3 with only 4 RTX 6000 GPUs. Afterward, the ILP problem can be solved in **less than a second** (on a 15 inch MacBook Pro), as compared to more than

10/50 hours searching using RL [235] with 4 RTX 6000 GPUs.

F.9 ILP Result Interpolation

We plot the bit-precision setting for each layer of ResNet18 that the ILP solver finds for different latency constraints, as shown in Figure F.4. Additionally, we also plot the sensitivity (Ω_i in (7.9)) and the corresponding speed up for each layer computed by quantizing the respective layer in INT8 quantization versus INT4. As can be seen, the bit configuration chosen by the ILP solver is highly intuitive based on the latency speed-up and the sensitivity. Particularly, when the mixed-precision model is constrained by the High-Latency setting (the first row of Figure F.4), only relatively insensitive layers, along with those that enjoy high INT4 speed-up, are quantized (i.e., layers 9, 14, and 19). However, for the more strict Low-Latency setting (last row of Figure F.4), only very sensitive layers are kept at INT8 precision (layer 1, 2, 3, 5, and 7).[1]

[1]Note that here layer 7 is the downsampling layer along with layer 5, so it is in the same bit setting as layer 5 even though the latency gain of layer 7 is limited.

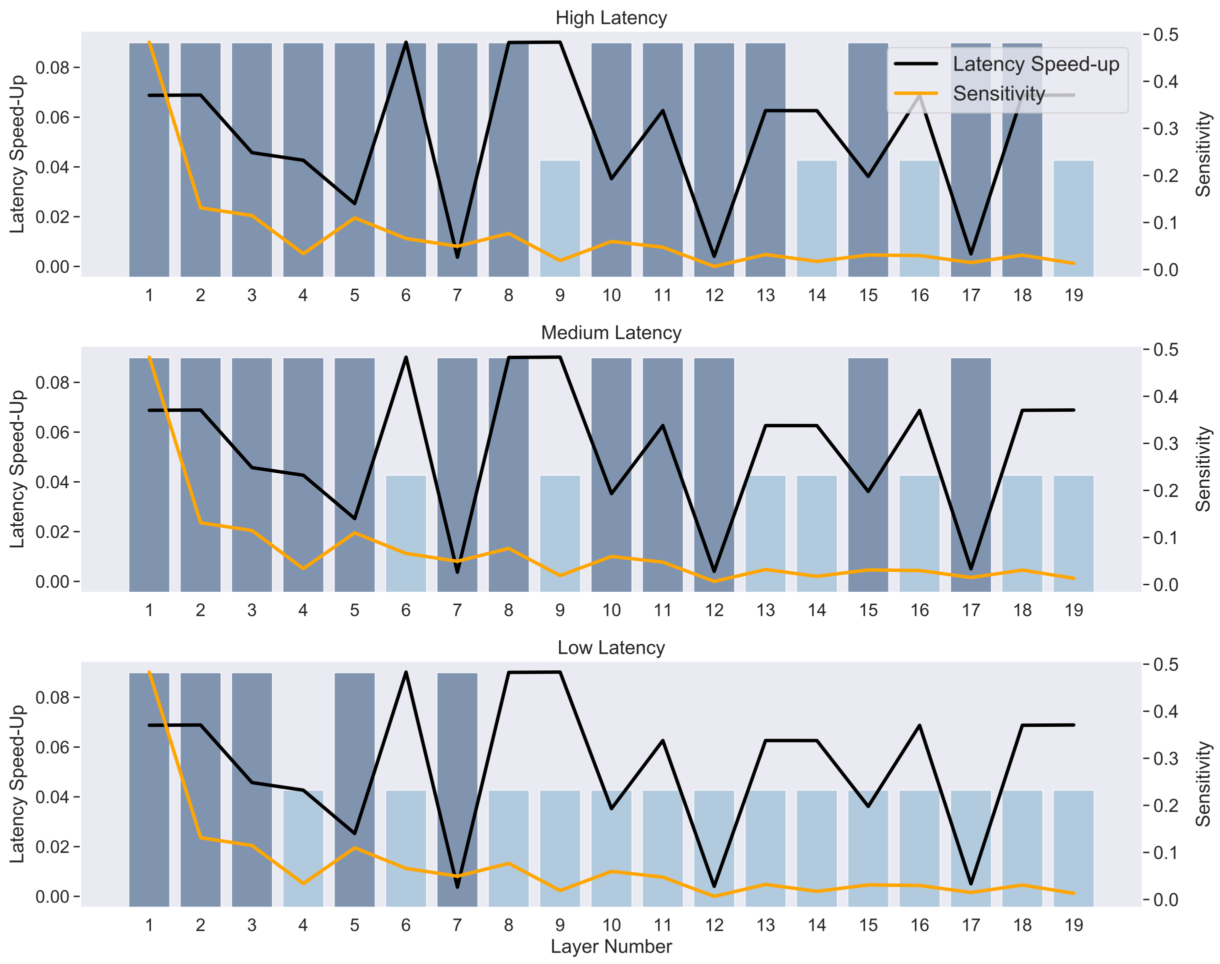

Figure F.4: Illustration of the final model specification that the ILP solver finds for ResNet18 with latency constraint. The black line shows the percentage of latency reduction for a layer executed in INT4 versus INT8, normalized by total inference reduction. Higher values mean higher speedup with INT4. The orange line shows the sensitive difference between INT8 and INT4 quantization using second-order Hessian sensitivity [69]. The bit-precision setting found by ILP is shown in bar plots, with the blue and taller bars denoting INT8, and cyan and shorter bars denoting INT4. Each row corresponds to the three results presented in Table 7.2 with latency constraint. For the low latency constraint, the ILP solver favors assigning INT4 for layers that exhibit large gains in latency when executed in INT4 (i.e., higher values in dark plot) and that have low sensitivity (lower values in the orange plot).

Appendix G

Appendix for Chapter 8

G.1 Importance of Speed: Adversarial Training for large-scale Learning

As discussed in the introduction, one of the motivations of using adversarial attacks is that multiple recent studies have shown superior results when adversarial data is injected during training [211, 205, 251, 253]. However, the overhead of computing an adversarial noise needs to be small, specially for large scale training. Here, the speed with which the adversarial noise is computed becomes important, so as not to slow down the training time significantly. To illustrate the efficiency of TR attacks, we perform adversarial training using the Adaptive Batch Size Adversarial Training (ABSA) method introduced by [251, 253] for large-scale training. We test ResNet-18 on Cifar-10 which achieves a baseline accuracy of **83.50%** (with a batch size of 128 using the setup as in [253]). If we use the TR L_∞ attack, the final accuracy increases to **87.79%** (actually for the hardest with batch size of 16K). If we instead use FGSM, the final accuracy only increases to **84.32%** (for the same batch size of 16K). It is important to note that it is computationally infeasible to use CW in this sort of training, as it is about 15× slower than TR/FGSM. This result shows that a stronger attack that is fast may be very useful for adversarial training. We emphasize that the goal of this test is simply to illustrate the importance of a fast and stronger adversarial attack, which could make TR to be a useful tool for adversarial training research. More thorough testing (left for future work) is required to understand how stronger attacks such as TR could be used in the context of adversarial training.

G.2 Visual Examples

Here we provide more visual examples for different neural networks on ImageNet. We show the original image along with adversarially perturbed ones using the three methods of DeepFool, CW, and ours (TR). We also provide two heat maps for the adversarial perturbations.

The first one is calibrated based on DeepFool's maximum perturbation magnitude, and the second one is calibrated for each of the individual method's maximum perturbation. This allows a better visualization of the adversarial perturbation of each method.

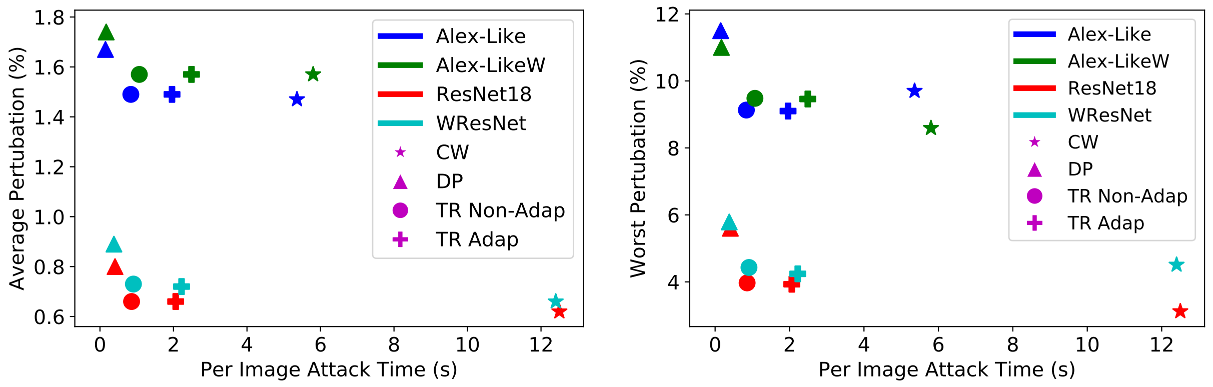

Figure G.1: The figures show various neural networks, the corresponding time to compute the adversarial attack (x-axis) and the perturbations a particular attack method needs to fool an image (detailed results in Table 8.1). The results are obtained for the Cifar-10 dataset (for ImageNet results please see Figure 8.4). On the left we report the average perturbation and on the right we report the worst perturbation. In particular, different colors represent different models, and different markers illustrate the different attack methods. Notice that, our TR and TR Adap achieve similar perturbations as CW but with significantly less time (up to **14.5**×).

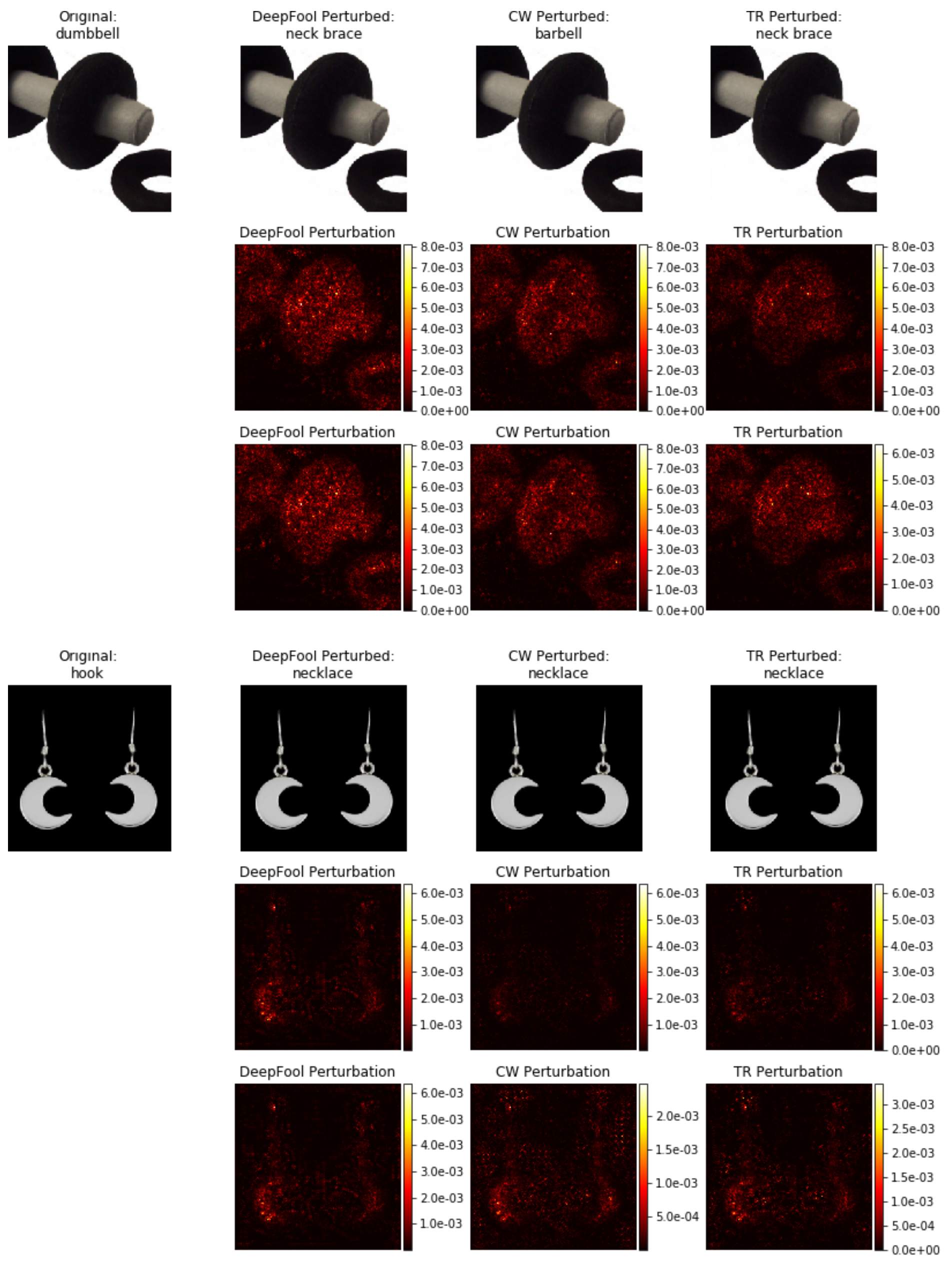

Figure G.2: ResNet-50 adversarial examples; row 1: original and adversarial images with ImageNet labels as titles; row 2: perturbation heat map with same colorbar scale; row 3: perturbations with color bar scale adjusted to each method for better visualization. TR attack obtains similar perturbation as CW, but **26.4**× faster Figure 8.4.

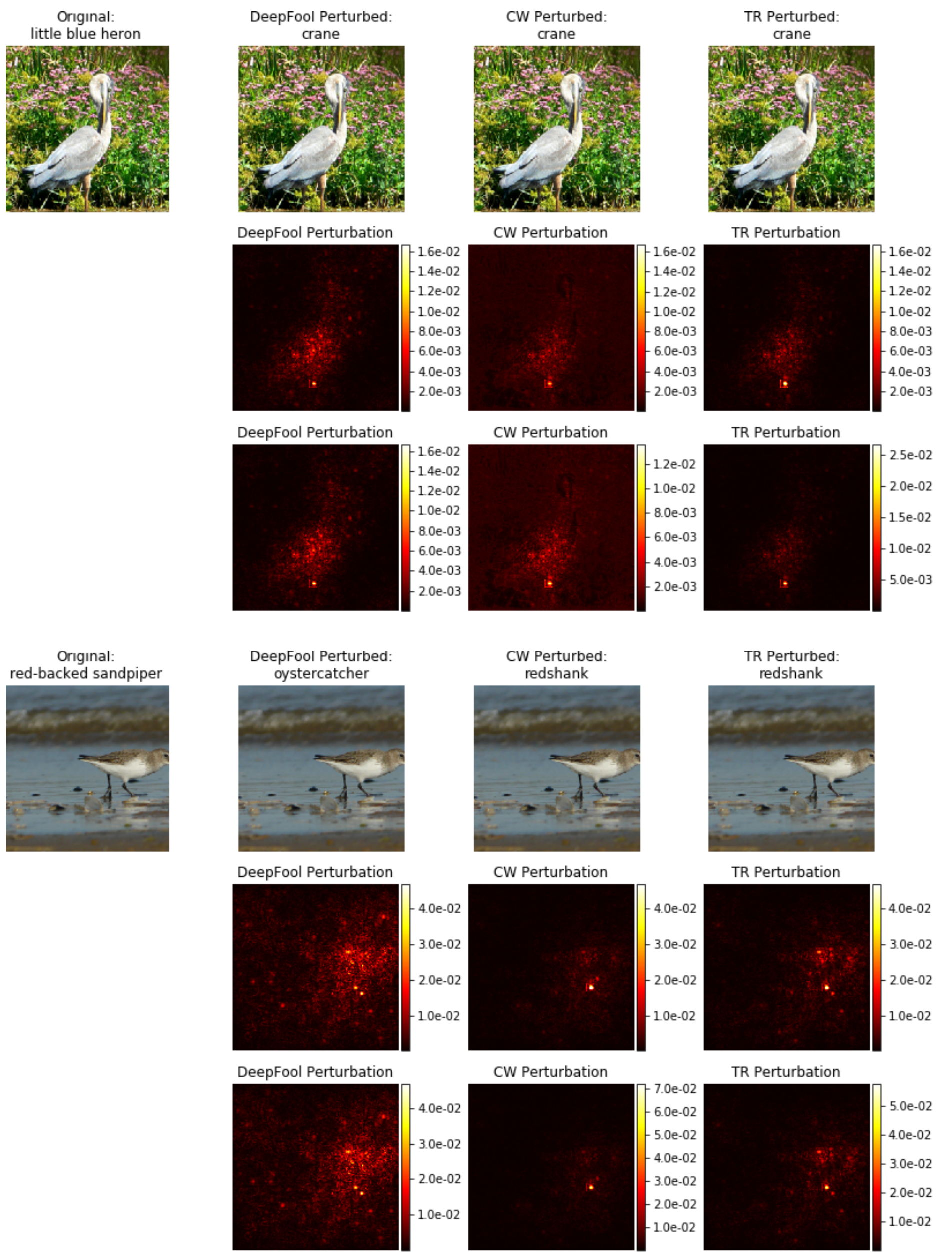

Figure G.3: AlexNet adversarial examples; row 1: original and adversarial images with ImageNet labels as titles; row 2: perturbation heat map with same colorbar scale; row 3: perturbations with color bar scale adjusted to each method for better visualization. TR attack obtains similar perturbation as CW, but **15**× faster Figure 8.4.

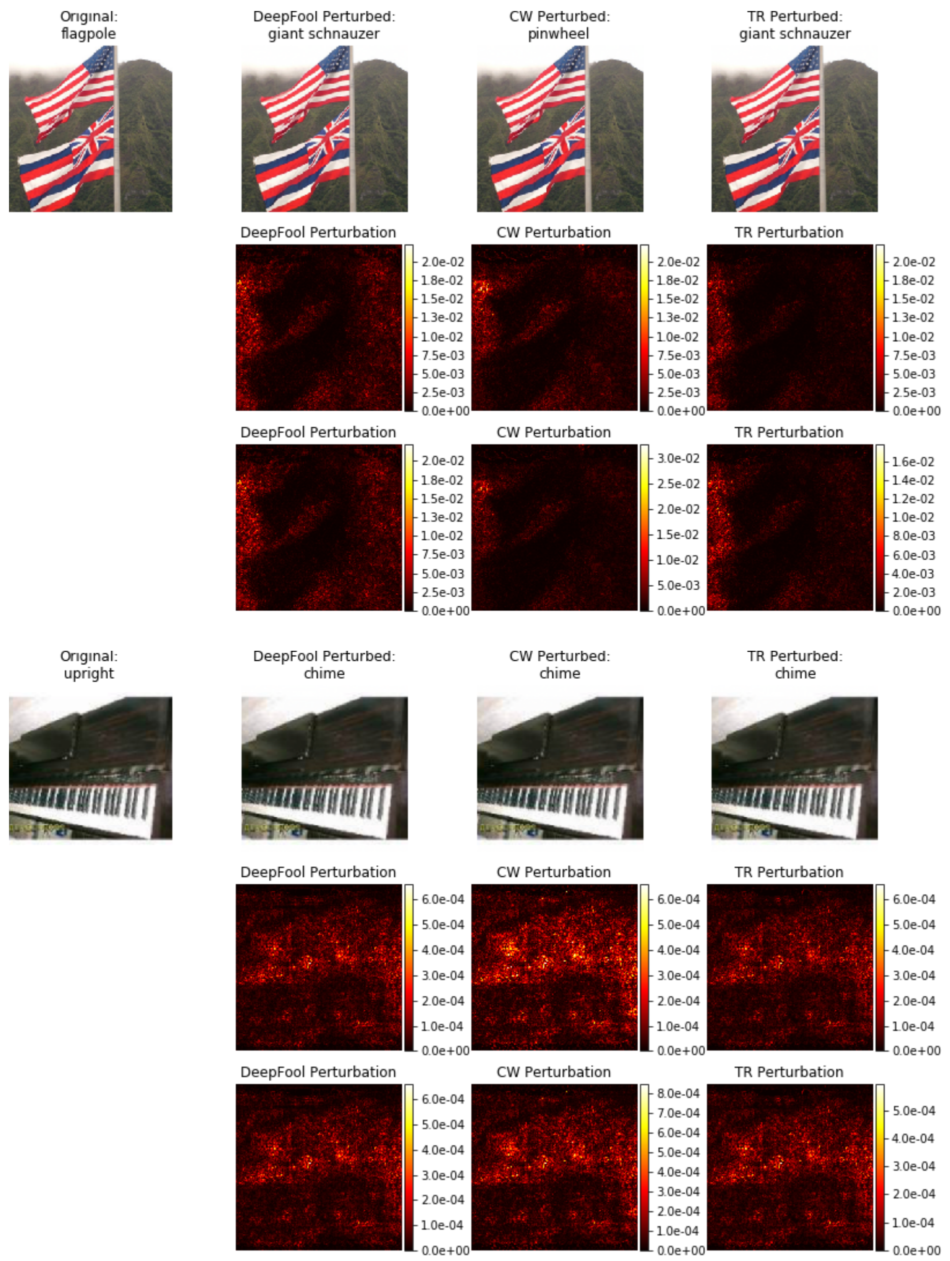

Figure G.4: DenseNet-121 adversarial examples; row 1: original and adversarial images with ImageNet labels as titles; row 2: perturbation heat map with same colorbar scale; row 3: perturbations with color bar scale adjusted to each method for better visualization. TR attack obtains similar perturbation as CW, but **26.8**× faster Figure 8.4.