Towards Practical Second-Order Optimizers in Deep Learning: Insights from Fisher Information Analysis

Damien Martins Gomes

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Computer Science) at

Concordia University

Montréal, Québec, Canada

April 2025

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Damien Martins Gomes**

Entitled: **Towards Practical Second-Order Optimizers in Deep Learning: Insights from Fisher Information Analysis**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Mirco Ravanelli

_____ Examiner
Dr. Elvis Dohmatob

_____ Examiner
Dr. Mirco Ravanelli

_____ Supervisor
Dr. Mahdi S. Hosseini

Approved by _____
Joey Paquet, Chair
Department of Computer Science and Software Engineering

_____ 2025

_____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Towards Practical Second-Order Optimizers in Deep Learning: Insights from Fisher Information Analysis

Damien Martins Gomes

First-order optimization methods remain the standard for training deep neural networks (DNNs). Optimizers like Adam incorporate limited curvature information by preconditioning the stochastic gradient with a diagonal matrix. Despite the widespread adoption of first-order methods, second-order optimization algorithms often exhibit superior convergence compared to methods like Adam and SGD. However, their practicality in training DNNs is still limited by a significantly higher per-iteration computational cost compared to first-order methods. In this thesis, we present *AdaFisher*, a novel adaptive second-order optimizer that leverages a *diagonal block-Kronecker approximation* of the Fisher information matrix to adaptively precondition gradients. AdaFisher aims to bridge the gap between the improved *convergence* and *generalization* of second-order methods and the computational efficiency needed for training DNNs. Despite the traditionally slower speed of second-order optimizers, AdaFisher is effective for tasks such as image classification and language modeling, exhibiting remarkable stability and robustness during hyperparameter tuning. We demonstrate that AdaFisher **outperforms state-of-the-art optimizers** in both accuracy and convergence speed. The Code is available from https://github.com/AtlasAnalyticsLab/AdaFisher.

Acknowledgments

I would like to express my deepest gratitude to Dr. Mahdi S. Hosseini, my supervisor, for his invaluable guidance, extensive support, and the countless hours he dedicated to helping me prepare and submit the AdaFisher paper to the ICLR 2025 conference. His mentorship has been instrumental in this journey.

A special and heartfelt thanks to PhD. Yanley (Kaly) Zhang, the second author of the AdaFisher paper and co-author for the BDN paper, for his relentless efforts, time, and dedication. His contributions to the experiments and countless meetings, even during late nights, have been crucial to the success of this work.

I am also immensely grateful to Atlas Analytics Lab for their support throughout this journey. I benefited greatly from the advice and assistance of my colleagues Cassandre, Ali, Hailey, Amirhossein, Thomas, Denisha, Vasudev, and Sina. Your support has been invaluable.

Merci aux Fonds de Recherche du Québec - Nature et Technologies, pour leur soutien et leur volonté de soutenir la recherche académique québécoise.

I extend my deepest gratitude to my family and friends. To Alexis, my roommate, for the engaging discussions about research, fieldwork, and One Piece. Your intellectual companionship has been invaluable. Thank you to my mother and sibling for their unwavering encouragement and support across the Atlantic.

Merci à mon école en France, IPSA Toulouse, de m'avoir permis de compléter mon parcours universitaire par une expérience de recherche inoubliable. Special thanks to Dr. Lorenzo Ortega for accompanying me as my supervisor in France.

And finally, to all the people who have accompanied me on this research journey, both from France and Canada, merci infiniment pour votre soutien.

Contribution of Authors

Damien Martins Gomes originated and developed the AdaFisher optimizer, addressing the critical challenge of achieving rapid convergence and enhanced generalization while preserving the computational efficiency typical of first-order methods. Drawing on an extensive literature review and extensive discussions with his supervisor, Dr. Mahdi S. Hosseini, Damien designed the theoretical framework for AdaFisher and validated its performance through large-scale experiments. His work was pivotal not only in the conceptual development and empirical evaluation of AdaFisher but also in the composition of the conference paper and the rigorous rebuttal process for ICLR 2025 submission.

Dr. Yanlei Zhang, a Post-Doctoral researcher at Mila and Université de Montréal, joined the project to further strengthen the convergence analysis and support the experimental investigations of AdaFisher. His contributions were crucial in refining the theoretical analysis and ensuring robust experimental outcomes. Dr. Zhang also played an active role in drafting the manuscript and participated in numerous collaborative meetings, thereby enhancing the overall quality of the work.

All authors have reviewed and approved the final manuscript.

Contents

List of Figures

List of Tables

Glossary

AI: Artificial Intelligence

BN: BatchNorm Layer

CNNs: Convolutional Neural Networks

CV: Computer Vision

DL: Deep Learning

DNNs: Deep Neural Networks

EFIM: Empirical Fisher Information Matrix

EMA: Exponential Moving Average

FFT: Fast Fourier Transform

FIM: Fisher Information Matrix

HP: Hyper-Parameter

KF: Kronecker Factor

LLM: Large Language Model

MLP: Multi-Layer Perceptron

NGD: Natural Gradient Descent

NLP: Natural Language Processing

NN: Neural Network

PCA: Principal Component Analysis

PPL: Perplexity

PTB: Penn TreeBank

SGD: Stochastic Gradient Descent

SNRs: Signal-to-Noise Ratios

SOTA: State-Of-The-Art

ViTs: Vision Transformers

VRAM: Video RAM

WCT: Wall-Clock-Time

Chapter 1

Introduction

Deep Neural Networks (DNNs) have revolutionized a wide array of applications, from Computer Vision (CV) and Natural Language Processing (NLP) to reinforcement learning and beyond. Despite their success, training these models remains a challenging task due to the complexity of the loss landscapes they must navigate. In particular, DNN optimization often struggles with the dual challenges of achieving fast convergence and robust generalization across diverse architectures and complex data distributions.

DNNs have demonstrated remarkable success across a wide range of applications, yet their training remains a computationally intensive, time-consuming process that often requires massive amounts of data (Brown et al., 2020). This raises a fundamental motivating question: **how can we train Neural Networks (NN) more effectively**? Efforts to address this challenge have emerged from many directions, including improved optimization algorithms (Martens & Grosse, 2015a; Martens et al., 2010), specialized hardware designs (Misra & Saha, 2010), more data-efficient NN architectures (Snell, Swersky, & Zemel, 2017), and dedicated Deep Learning (DL) compilers (T. Chen et al., 2018; Rotem et al., 2018). However, each of these approaches underscores the need for a deeper understanding of the factors that govern NN performance. Modifications to network architectures or optimization strategies can have profound impacts, as can more subtle changes such as reduced precision training (S. Gupta, Agrawal, Gopalakrishnan, & Narayanan, 2015). Yet, finding a cohesive set of tools to comprehend and harness these effects remains a long-standing challenge in DL optimization.

This thesis promotes a holistic approach to DL optimization, exploring both practical implementations and theoretical insights to illuminate the structure of loss landscapes and the dynamics of training. Central to our perspective is the observation that DL models exhibit a surprisingly rich structure in their loss landscapes, a property that not only facilitates acceleration in optimization but also helps explain the overall success of these models. By adopting loss landscape geometry as a framework, we reveal how various components—from curvature-aware updates to adaptive optimization techniques—contribute to effective neural network training. Connecting these pieces provides a clearer perspective on how the interplay between network architecture, optimization dynamics, and loss landscape geometry can be harnessed to design optimizers that balance convergence speed, generalization, and computational efficiency (Nocedal & Wright, 1999).

In the following chapters, we build on these ideas to introduce *AdaFisher*, an adaptive second-order optimizer that leverages a refined **diagonal block-Kronecker approximation** of the Fisher Information Matrix. Through theoretical development, extensive empirical validation, and comprehensive ablation studies, our work aims to advance the state of DL optimization, providing both practical solutions and new insights for the research community.

1.1 Problem Statement

At the core of DNN training is the minimization of a highly non-convex loss function $\mathcal{L}(\boldsymbol{\theta})$ by updating model parameters θ according to an iterative scheme

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha\,(\mathcal{G}^{(t)})^{-1}\nabla\mathcal{L}(\boldsymbol{\theta}^{(t)}),$$

where α is the learning rate and $\mathcal{G}^{(t)}$ encapsulates curvature information. For first-order methods like Stochastic Gradient Descent (SGD), $\mathcal{G}^{(t)}$ is simply the identity matrix, which makes these methods computationally efficient but often blind to the underlying geometry of the loss surface. In contrast, second-order methods employ richer curvature information—via the Hessian or the Fisher Information Matrix (FIM)—to rescale and orient gradients in a manner that can accelerate convergence and guide the optimizer toward flatter, more generalizable minima. However, these benefits come at a steep computational cost, as calculating and inverting full curvature matrices

is often intractable for large-scale networks. This thesis addresses the need for an optimizer that achieves a balance between rapid convergence, strong generalization, and computational efficiency.

1.2 Challenges Associated with the Problem

The existing literature reveals a fundamental trade-off in DNN optimization. On one hand, methods such as Adam (Kingma & Ba, 2015) and its variants (e.g., AdamP (Heo et al., 2021), AdaInject (Dubey, Basha, Singh, & Chaudhuri, 2022), and AdaBelief (Zhuang et al., 2020)) rely on diagonal approximations of the FIM to remain computationally efficient. While effective in many settings, these approximations can lead to suboptimal convergence and poorer generalization because they fail to capture the full curvature of the loss landscape. On the other hand, advanced second-order approaches like AdaHessian (Yao et al., 2021), Shampoo (V. Gupta et al., 2018), and K-FAC (Grosse & Martens, 2016) attempt to incorporate richer curvature information to improve optimization performance, but they are often hampered by high computational overhead, extensive hyper-parameter tuning requirements, and scalability issues on commodity hardware (Martens, 2020). This imbalance between the richness of curvature information and computational feasibility represents a critical challenge for the field, as an effective solution would greatly enhance training dynamics and yield better-performing models across various domains. Hence, we list three research questions that we will try to talk throughout this thesis: How can the Fisher Information Matrix's structure be leveraged to design a second-order optimizer that is computationally feasible for deep networks?; Does using a refined FIM-based preconditioning improve convergence speed and generalization compared to first-order methods?; What are the impacts of various design choices (e.g., using square-root of adapting optimization scheme, including Fisher computation for normalization layers) on the optimizer's performance?

1.3 Proposed Methodology

To address this challenge, we introduce *AdaFisher*, a novel adaptive second-order optimization method that leverages a refined diagonal block-Kronecker approximation of the FIM. Inspired by Natural Gradient Descent (NGD) (Amari & Nagaoka, 2000)—which uses the FIM as a *Riemannian*

metric to precondition gradients—AdaFisher replaces the second moment in Adaptive framework with our enhanced FIM approximation. This novel approach effectively combines the computational efficiency of first-order methods with the curvature-awareness of second-order techniques. By capturing the essential curvature information through a diagonal block-Kronecker factorization, AdaFisher accelerates convergence and guides the optimization process toward flatter local minima, thereby improving generalization. As illustrated in Figure 1.1, AdaFisher not only converges more rapidly but also reaches a superior local minimum by effectively navigating through saddle points compared to its counterparts. Further details regarding the visualization can be found in Appendix B.

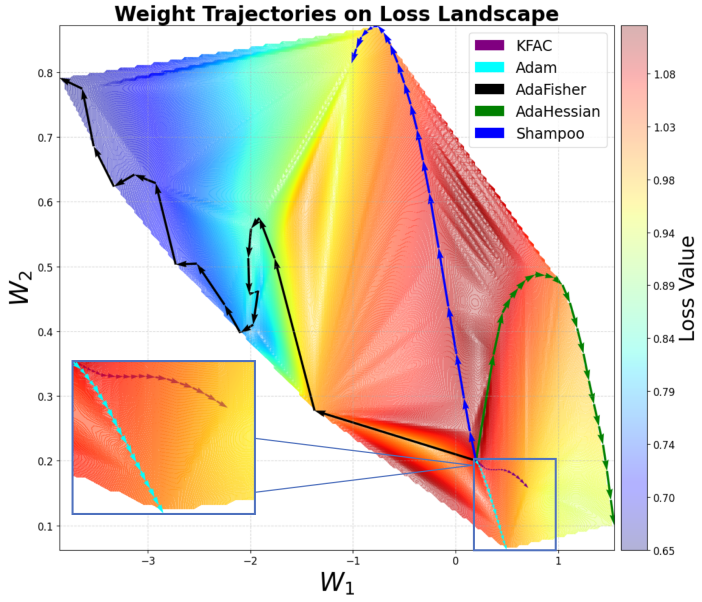

Figure 1.1: Visualizing optimization trajectories for various optimizers overlaid a loss landscape.

1.4 Contributions

The contributions of this thesis are multifaceted and build upon extensive theoretical and empirical insights in DNN optimization. First, we provide a comprehensive literature review that situates our work within the broader context of first-order, second-order, and emerging optimization methods, highlighting the limitations of current approaches. Second, we empirically demonstrate that the energy of the Kronecker Factors (KF) is predominantly concentrated along the diagonal, offering

fresh insights into the structure of the FIM in deep learning. Third, we introduce a novel diagonal block-Kronecker approximation of the FIM, which is applicable across various network layers, including normalization layers, thereby enhancing model adaptability while maintaining computational efficiency. Fourth, we establish the robustness and stability of AdaFisher through extensive experiments across diverse training settings, proving its superior convergence and generalization performance. Fifth, we showcase AdaFisher's empirical performance against state-of-the-art optimizers in both image classification and language modeling tasks. Lastly, we develop innovative visualization techniques that map optimizer trajectories in the loss landscape and introduce an explainable FIM measure, facilitating comparative analysis of optimizer behavior.

1.5 Outline

This thesis is organized to provide a cohesive narrative that bridges theoretical developments with practical implementations. Chapter 2 offers a detailed literature review, surveying existing optimization techniques and identifying the challenges that motivate our approach. Chapter 3 delves into a new approximation of the FIM which enables its computation for large scale DNNs while preserving the main curvature information. Then in Chapter 4 we introduce AdaFisher optimizer, explaining its theoretical foundations, algorithmic structure, and distributed implementation. In Chapter 5, we present extensive experiments in both computer vision and natural language processing, demonstrating the effectiveness of AdaFisher relative to other state-of-the-art optimizers. Chapter 6 provides an in-depth ablation and stability analysis, dissecting the contributions of individual components within AdaFisher. Finally, Chapter 7 concludes the thesis with a discussion of the implications of our findings and potential directions for future research.

By addressing the inherent trade-offs in DNN optimization, this thesis contributes a new, scalable approach that combines the efficiency of first-order methods with the precision of second-order curvature information, paving the way for more robust and generalizable deep learning models.

Chapter 2

Literature Review

Deep Learning has transformed numerous fields by achieving breakthrough results in computer vision (K. He, Zhang, Ren, & Sun, 2016; Krizhevsky, Sutskever, & Hinton, 2012; Z. Liu et al., 2021), natural language processing (Brown et al., 2020; Devlin, 2018; Vaswani et al., 2017), and beyond. The extraordinary performance of deep networks is not solely a consequence of their expressive architectures but also of the intricate dynamics that govern their training. Understanding these dynamics is crucial for several reasons. First, it sheds light on the non-convex optimization challenges inherent to deep models. Second, it helps to reveal why and how over-parameterized networks generalize well despite their capacity to fit highly complex datasets. Third, it informs the design of optimization algorithms that balance rapid convergence with robust generalization. While empirical successes abound, theoretical understanding lags behind practice, with critical open questions persisting about: 1) how optimization trajectories navigate high-dimensional non-convex landscapes (Zaheer, Reddi, Sachan, Kale, & Kumar, 2018), 2) why certain algorithms generalize better despite equivalent training performance (Allen-Zhu, Li, & Liang, 2019), and 3) how computational constraints shape method design (Hu & Huang, 2023). This chapter synthesizes recent advances across three interconnected themes:

First, we will introduce the preliminaries notations and concepts in Section 2.1, then Section 2.2 analyzes DL dynamics through the lens of phase transitions, loss geometry, and implicit regularization. Section 2.3 then delineates fundamental obstacles in NN optimization, from saddle point proliferation to memory-time tradeoffs. Section 2.4 systematically evaluates optimization paradigms,

contrasting zeroth-, first-, and second-order methods through their theoretical guarantees and empirical behavior. Finally, Section 2.5 discusses emerging directions that contextualize the design space for modern optimizers.

2.1 Preliminaries

2.1.1 Standard Notation

We include an overview of our notation within this section in addition to covering some fundamental material that we use throughout the thesis, and additional notations will be defined as needed. We write $\mathbb{R}$ for the space of real numbers and $\mathbb{N}$ for the space of natural numbers. Scalars are typically represented in lower-case, e.g. $x \in \mathbb{R}$. We present vectors in lower-case boldface font, e.g. $\mathbf{x} \in \mathbb{R}^d$. Matrices are written in upper-case font, e.g. $A \in \mathbb{R}^{d \times d}$. We use subscript to denote the entries of vectors and matrices. For example, v_i denotes the i^{th} entry of the vector $\mathbf{v}$, and A_{ij} denotes the entry of A at position (i, j).

2.1.2 Probabilities

Throughout this thesis, we rely on fundamental tools from probability theory. While a careful, thorough review is out of scope, we present here some key ideas that we need later. To denote the probability of an event A occurring, we write $\mathbb{P}[A]$. We frequently work with random variables, which are defined as a function from a sample space to a measurable space. We focus primarily on real-valued random variables, such that the measurable space is a subset of $\mathbb{R}$. Our random variables can typically be described via a *probability density function* (PDF), $p(x)$. That is, given a random variable X,

$$\mathbb{P}[X \in R] = \int_R p(x)dx,$$

where $R \in \mathbb{R}$. Given two random variables X and Y, we define the conditional distribution of X given Y by the PDF,

$$p(x|y) = \frac{p(x,y)}{p(y)}$$

where $p(x,y)$ denotes the joint PDF.

Expected value. We define the expected value of a real-valued random variable with PDF $p(x)$ as,

$$\mathbb{E}[X] = \int_{\mathbb{R}} xp(x)dx$$

This definition can be readily extended to vector-valued random variables by integrating element-wise. In the case where we have several samples of a random variable, $x_1, \ldots, x_n$, we write the empirical mean as,

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Covariance Matrices. Given a vector-valued random variable $x \in \mathbb{R}^d$, its covariance is defined by,

$$\text{Cov}(\mathbf{x}) = \mathbb{E}\left[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^\top\right].$$

Given, observations $\mathbf{x}_i \in \mathbb{R}^{d\times n}$, for $i = 1, \ldots, n$ of the r.v $\mathbf{x}$, we define the empirical covariance matrix as,

$$\hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \bar{\mathbf{x}})^\top = \frac{1}{n}(X - \bar{X})(X - \bar{X})^\top,$$

where the design matrix $X \in \mathbb{R}^{d\times n}$ contains $\mathbf{x}_i$ in its i^{th} column, and $\bar{X} \in \mathbb{R}^{d\times n}$has $\bar{\mathbf{x}}$ in each column. From this, we can see that the empirical covariance matrix is symmetric and positive semidefinite–the latter meaning that all eigenvalues of $\hat{\Sigma}$ are non-negative.

Latent Variable Models. In Chapter 3, we present a new, efficient approximation of the Fisher Information Matrix—defined below—which naturally arises in the context of latent variable models that require an underlying statistical framework. These are probabilistic graphical models (Koller, 2009) where latent variables[1] capture features of observed variables. In the simplest setting, the latent variables, $\mathbf{z} \in \mathbb{R}^k$, and the observed variables $\mathbf{x} \in \mathbb{R}^d$ have their joint PDF defined in the following way,

$$p(x, z) = p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p(\mathbf{z}),$$

where $p(\mathbf{z})$ represents a *prior distribution* over the latent variables, and $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ represents the conditional distribution parameterized by some $\boldsymbol{\theta}$ to be learned from observed data. Given some i.i.d. observed data, $\{\mathbf{x}_i\}_{i=1}^n$, we seek to maximize the log marginal likelihood,

$$\sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(\mathbf{x}_i) = \sum_{i=1}^{n} \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{z}|\mathbf{x}_i)}.$$

Optimizing the parameters, $\boldsymbol{\theta}$, typically requires the computation of (or samples from) the posterior distribution. But this is not generally available in closed form. Several methods exist that allow us to perform inference in this model regardless.

To formalize the notion of the FIM, consider a parametric model $p_{\boldsymbol{\theta}}(\mathbf{x})$ describing observed data $\mathbf{x} \in \mathbb{R}^d$. The FIM with respect to $\boldsymbol{\theta}$ is given by

$$F(\boldsymbol{\theta}) \;=\; \mathbb{E}_{\mathbf{x}\sim p_{\boldsymbol{\theta}}(\mathbf{x})}\Big[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x})\; \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x})^{\top}\Big], \tag{1}$$

i.e., it measures the expected curvature of the log-likelihood and captures how sensitively the model's probability distribution depends on its parameters. Further discussions about the importance of the FIM and its relevance in optimization scenarios will follow later.

[1] Often referred to as *unobserved* or *hidden* variables

2.1.3 Multivariate Calculus

Throughout this thesis, we extensively employ multivariate calculus and carefully outline the notations and underlying concepts. Given a function $f : \mathbb{R}^n \to \mathbb{R}^m$, we can write its Taylor series expansion about a point $\boldsymbol{\theta}^{(0)} \in \mathbb{R}^n$ as,

$$f(\boldsymbol{\theta}) = f(\boldsymbol{\theta}^{(0)}) + J(\boldsymbol{\theta}^{(0)})(\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)}) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)})^\top H(\boldsymbol{\theta}^{(0)})(\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)}) + o(||\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)}||_2^2),$$

where $J(\boldsymbol{\theta}^{(0)}) \in \mathbb{R}^{m \times n}$ denotes the *Jacobian matrix* evaluated at $\boldsymbol{\theta}^{(0)}$ and $H(\boldsymbol{\theta}^{(0)}) \in \mathbb{R}^{n \times m \times n}$ is the *Hessian tensor* evaluated at $\boldsymbol{\theta}^{(0)}$. We use the notation,

$$[(\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)})^\top H(\boldsymbol{\theta}^{(0)})(\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)})]_j = \sum_{i=1}^{n}\sum_{k=1}^{n}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)})_i H(\boldsymbol{\theta}^{(0)})_{ijk}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)})_k,$$

that is, the vector-tensor product is applied on the outer indices. The Jacobian matrix can be defined as the collection of first-order derivatives between all input-output pairs of the function. Explicitly, writing $f = (f_1, \ldots, f_m)$ and $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$ the ij^{th} entry of J is given by,

$$J(\boldsymbol{\theta}_0)_{ij} = \frac{\partial f_i}{\partial \theta_j}\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(0)}}.$$

The Jacobian matrix is a linear operator that determines the rate of change of the function along some direction. The Hessian tensor can be defined similarly,

$$H(\boldsymbol{\theta}_0)_{ijk} = \frac{\partial^2 f_j}{\partial \theta_i \partial \theta_k}\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(0)}}.$$

The Hessian determines the curvature of the function and is a particularly useful object for optimization. For instance, the Hessian can be used to classify stationary points as saddles, minima, or maxima and measures the conditioning of these stationary points as the ratio of the largest and smallest singular values.

2.1.4 Neural Networks

This thesis is ultimately concerned with understanding the properties of NN and how we can leverage this knowledge to develop a methodology for enhancing the convergence/generalization. Our investigation of NNs ranges from closed-form theoretical analysis to experimental study. In this section, we briefly introduce the notation that we use to describe NNs throughout this thesis — with an emphasis on the NNs that we focus on within our theoretical analysis.

We consider a supervised learning framework with a dataset $\mathbf{D}$ containing N i.i.d samples, $\mathbf{D} \coloneqq \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$ where $\mathbf{x}_n \in \mathbb{R}^d$ and $\mathbf{y}_n \in \mathbb{R}^C$. Loosely speaking, a NN is a parametric function, $f_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}^C$, parametrized by $\boldsymbol{\theta}$ where $\theta_i = \text{concat}(W_i, \mathbf{b}_i) \in \mathbb{R}^{P_i}$, and $P_i = P_i^{out} \times (P_i^{in} + 1)$. Let $\mathcal{L} : \mathbb{R}^C \times \mathbb{R}^C \to \mathbb{R}$ be the loss function defined by the Negative Log-Likelihood (NLL), i.e. $\mathcal{L}(\mathbf{y}, f_{\boldsymbol{\theta}}(\mathbf{x})) \coloneqq -\log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$ where $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$ is the likelihood of the NN f_θ. The network computes its output $h_L = f_{\boldsymbol{\theta}}(\mathbf{x})$ according to

$$\mathbf{a}_i = \boldsymbol{\theta}_i \bar{\mathbf{h}}_{i-1}, \;\; \mathbf{h}_i = \phi_i(\mathbf{a}_i), \;\; \forall\, i \in \{1, \dots, L\} \mid \mathbf{h}_0 = \mathbf{x},$$

where $\bar{\mathbf{h}}_{i-1} = [\mathbf{h}_{i-1}^\top, \mathbf{1}]^\top \in \mathbb{R}^{P_{i-1}^{in}+1}$, and ϕ_i is an element-wise nonlinearity applied at layer i. For a given input target pair $(\mathbf{x}, \mathbf{y})$, the gradient of the loss $\mathcal{L}(\mathbf{y}, f_{\boldsymbol{\theta}}(\mathbf{x}))$ concerning the weights are computed by the backpropagation algorithm (Lecun, 2001). For convenience, we adopt the special symbol $\mathbf{s}_i = \nabla_{\mathbf{a}_i}\mathcal{L}$ for the pre-activation derivative. Starting from $\nabla_{\mathbf{h}_L}\mathcal{L} = \partial_{\mathbf{h}_L}\mathcal{L}(\mathbf{y}, \mathbf{h}_L)$, we perform

$$\mathbf{s}_i \coloneqq \nabla_{\mathbf{a}_i}\mathcal{L} = \nabla_{\mathbf{h}_i}\mathcal{L} \odot \phi_i'(\mathbf{a}_i), \; \nabla_{\theta_i}\mathcal{L} = \mathbf{s}_i \bar{\mathbf{h}}_{i-1}^\top, \; \nabla_{\bar{\mathbf{h}}_{i-1}}\mathcal{L} = \boldsymbol{\theta}_i^\top \mathbf{s}_i \quad \mid \forall i \in \{L, \dots, 1\},$$

where $\odot$ denotes the element-wise product. Finally, the gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ is retrieved by: $\nabla_{\boldsymbol{\theta}}\mathcal{L} = [\text{vec}(\nabla_{\theta_1}\mathcal{L})^\top, \text{vec}(\nabla_{\theta_2}\mathcal{L})^\top, \dots, \text{vec}(\nabla_{\theta_L}\mathcal{L})^\top]^\top$; $\text{vec}(\cdot)$ denotes the Kronecker vectorization operator which stacks the columns of a matrix into a vector. Throughout this thesis we will use the notation $\mathbf{g}^{(t)} = \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}^{(t)})$.

Network Jacobian. Network Jacobian in Section 2.1.3 we defined the Jacobian matrix. Here we write NNs as functions that depend on some parameters $\boldsymbol{\theta}$ and operate on some data $\mathbf{x}$. We typically consider the Jacobian matrix of NNs with respect to their parameters $\boldsymbol{\theta}$, and they are thus written as $J_{\boldsymbol{\theta}}(\mathbf{x})$.

2.1.5 Optimization

In this section, we review the background material that is relevant to the optimization components of this thesis. We will formalize the optimization problem considered when training a NN.

Setting. The optimization problems that we consider can be represented via an objective function, $\mathcal{L}$ (e.g., NLL defined in Section 2.1.4). In the settings that we study in this thesis, the objective function is evaluated for a given set of model parameters and observations. In the most general case, we use $\boldsymbol{\theta}$ to denote the model parameters. Note that in this thesis we will consider a supervised learning framework. Typically, the objective function can be decomposed as the average of several independent loss terms,

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \ell(\boldsymbol{\theta}; \mathbf{x}_n, \mathbf{y}_n)$$

Our goal is to minimize this objective function with respect to the model parameters. In doing so successfully, we recover a global minimum,

$$\boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$$

Notice the use of $\in$ instead of $=$ operator, this is because of the nature of the problem, which in the case of NN optimization, is highly non-convex. In the typical applications that we care about, such as deep learning optimization, n is far too large to evaluate $\mathcal{L}$ efficiently. In this case, we often resort to stochastic optimization where we obtain statistics of the objective function by evaluating it using only a randomly chosen subset of the observations.

Gradient Descent. A ubiquitous approach to minimizing $\mathcal{L}$ is to start with some initial guess of the optimal parameters, $\boldsymbol{\theta}_0$, and follow the gradient from $\boldsymbol{\theta}_0$ towards an optimum. This algorithm is known as gradient descent, and can be represented by the following recursive computation,

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha^{(t)}\mathbf{g}^{(t)}, \tag{2}$$

where $\alpha^{(t)}$ denotes the learning rate. This algorithm (and extensions of it) still form the basis of modern deep learning optimization. To better understand it, we will now provide a brief review of convex optimization and the behaviour of gradient descent under convex objectives.

Convex Optimization. In convex optimization, we restrict our attention to the class of convex objective functions.

Definition 2.1.1. *A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if and only if the following holds. For all $t \in [0, 1]$, and all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$,*

$$f(t\mathbf{x} + (1-t)\mathbf{y}) \leq tf(\mathbf{x}) + (1-t)f(\mathbf{y}).$$

In general, a convex optimization problem may include a set of constraints such that the feasible set of solutions is a convex set. Unless stated otherwise, we restrict our attention to unconstrained convex optimization.

Perhaps the simplest non-trivial convex function is the *quadratic function*, which (without loss of generality (WLG)) takes the form,

$$f(\mathbf{x}) = (\mathbf{x} - \mathbf{x}^*)^\top A(\mathbf{x} - \mathbf{x}^*), \tag{3}$$

where A is a symmetric positive semidefinite matrix. Convex quadratic optimization problems form the cornerstone of second-order optimization algorithms, since at each iteration the loss function is approximated by a convex quadratic via a second-order Taylor expansion, thereby making the resulting subproblem significantly easier to solve.

Note that throughout this thesis we consider *unconstrained composite convex optimization problems* which are significantly simpler than their constrained counterparts.

Convergence on convex objectives. The following properties of (unconstrained) convex optimization are key to our success in optimization (Boyd, 2004):

- Every locally optimal point of a convex objective function is globally optimal.
- If f is differentiable, then a point $\mathbf{x}$ is optimal if and only if $\nabla_{\mathbf{x}} f = 0$.

Importantly, the convergence rate of gradient descent at an optimum of an unconstrained convex optimization problem depends on the *condition number* of the Hessian matrix. That is the ratio of the largest eigenvalue to the smallest eigenvalue of the Hessian matrix. The larger the condition number, the slower the convergence.

Consider a quadratic objective with κ denoting the condition number of the Hessian matrix ($H = A$) in Eq. (3). Then the convergence rate of gradient descent (with optimal learning rate) is given by,

$$f(\mathbf{x}^{(t)}) - f(\mathbf{x}^*) = \left(1 - \frac{1}{\kappa}\right)^t (f(\mathbf{x}_0) - f(\mathbf{x}^*)),$$

where x_0 is the initial point. One commonly used method to improve convergence under ill-conditioned objectives is by utilizing acceleration. These methods typically reduce the dependency on the condition number from $1/\kappa$ to $1/\sqrt{\kappa}$.

2.2 Deep Learning Dynamics

The dynamics of deep learning encompass the evolution of network parameters as a function of training time and the interplay between the optimization method, loss landscape, and model architecture. These dynamics are central to understanding how deep networks transition between different regimes of learning, how they navigate highly non-convex landscapes, and how they converge to solutions with desirable generalization properties. In this section, we review the state-of-the-art (SOTA) in deep learning dynamics with a focus on three aspects: training dynamics and phase

transitions, loss landscape geometry, and implicit regularization.

2.2.1 Training Dynamics and Phase Transitions

Neural Tangent Kernel (NTK) Theory. A powerful theoretical lens on the training behavior of overparameterized NN is provided by the *Neural Tangent Kernel (NTK)* framework (Jacot, Gabriel, & Hongler, 2018). Consider a NN $f_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}$ (WLG, assume a scalar output) parameterized by $\boldsymbol{\theta} \in \mathbb{R}^p$. At any parameter setting $\boldsymbol{\theta}$, the *NTK* between two inputs $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$ is defined as

$$\mathcal{K}_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}') \;=\; \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{x})^\top \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{x}')\,.$$

In the *NTK regime*—often associated with very wide (infinite-width) NNs—the evolution of network parameters under gradient descent can be approximated by linearizing $f_{\boldsymbol{\theta}}(\mathbf{x})$ around its initialization $\boldsymbol{\theta}^{(0)}$. Concretely, writing $\boldsymbol{\theta}^{(t)}$ for the parameters at iteration t,

$$f_{\boldsymbol{\theta}^{(t)}}(\mathbf{x}) \;\approx\; f_{\boldsymbol{\theta}^{(0)}}(\mathbf{x}) \;+\; \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}^{(0)}}(\mathbf{x})^\top \big(\boldsymbol{\theta}^{(t)} \;-\; \boldsymbol{\theta}^{(0)}\big).$$

Under this approximation, the kernel $\mathcal{K}_{\boldsymbol{\theta}^{(0)}}(\mathbf{x}, \mathbf{x}')$ remains nearly constant throughout training. As a result, the training dynamics reduce to a linearized model whose parameters evolve in a predictable manner. While this perspective sheds light on many salient features of training—including the speed of convergence and the capacity for perfect interpolation in overparameterized networks—it only partially captures the highly non-linear aspects observed in real-world scenarios where finite width, adaptive optimizers, and more complex architectures induce additional effects.

Phase Transitions in Training. The training process of deep networks is characterized by complex temporal behavior that often manifests as distinct phases (Fort et al., 2020). Early in training, the network typically undergoes rapid changes, while later stages are marked by slower, diffusion-like dynamics. Recent studies have systematically mapped out these phases by examining how Hyper-Parameters (HP) such as learning rate, depth, and width influence convergence behavior (Erhan, Courville, Bengio, & Vincent, 2010; Xie, Wang, Zhang, Sato, & Sugiyama, 2022). For example, in Kalra and Barkeshli (2023), the authors present a phase diagram that distinguishes regimes

of fast initial descent from later phases where learning slows, thereby highlighting critical transition points that affect both convergence and generalization. In (Jastrzebski et al., 2020), the authors analyze the importance of the early phase of training and its critical impact on final performance, arguing that key properties of the loss surface are strongly influenced by initial learning dynamics.

A complementary perspective is provided by theoretical frameworks such as the NTK described above. Although the NTK regime simplifies the analysis by approximating the parameter evolution as quasi-linear, it captures only a fraction of the full non-linear behavior observed in practice. In more realistic settings—where non-linear activations, finite widths, and adaptive optimization methods are prevalent—the training dynamics exhibit richer behavior, including abrupt shifts (or phase transitions) that mark the onset of distinct learning regimes.

The inherent stochasticity in optimization methods like Stochastic Gradient Descent (SGD) (Marcus, Santorini, & Marcinkiewicz, 1993) introduces noise that can help escape poor local minima and saddle points. For instance, Jin, Ge, Netrapalli, Kakade, and Jordan (2017) demonstrated that by adding noise at each step, gradient descent can escape saddle points in polynomial time. Moreover, Xie, Sato, and Sugiyama (2021) proposes a diffusion theory for deep learning dynamics in which the stochastic fluctuations inherent in SGD introduce an additional regularization effect, steering the optimization trajectory toward broader, flatter minima. This bias toward flat solutions is argued to underpin the superior generalization performance observed in deep networks, as flatter minima are typically more robust to perturbations and less prone to overfitting (Frankle & Carbin, 2019).

2.2.2 Loss Landscape Geometry

What is a loss landscape? A loss landscape is determined by the output of the loss function over some subset of the optimization parameters. The loss landscape describes the space explored by an optimization algorithm and ultimately determines the optimizer's trajectory. Figure 2.1 displays a 2D slice of the loss landscape of a fully-connected network.

Geometry Intuitions. The geometry of the loss landscape is a key determinant of both the ease of optimization and the generalization ability of a DNN. Empirical investigations have revealed that

Figure 2.1: A 2D slice of a loss landscape. This is a 3D rendering of a 2D slice of the loss landscape of a fully-connected network. The slice itself was chosen via gradient descent, using an objective that encouraged this 2D slice to match a target image. Full details are given by Lucas (2022).

the loss surfaces of deep models are *highly non-convex* and characterized by a multitude of local minima and saddle points. Yet, intriguingly, many of these minima are connected via *low-loss pathways*, suggesting that the landscape possesses an underlying structure that facilitates optimization (Garipov, Izmailov, Podoprikhin, Vetrov, & Wilson, 2018; yeh Chiang et al., 2023). One influential concept in this context is the *Goldilocks zone*, which refers to regions in the parameter space where the curvature is balanced—not too steep and not too flat. Studies such as Fort and Scherlis (2019) and Vysogorets, Dawid, and Kempe (2024) have shown that minima in these zones tend to be flat, and flatness has been empirically associated with better generalization performance. The spectral properties of the Hessian matrix have often been used to characterize these regions: minima with a concentration of small eigenvalues (i.e., flat minima) are typically more robust to perturbations, whereas sharp minima, characterized by large eigenvalues, are more sensitive and prone to overfitting (Dinh, Pascanu, Bengio, & Bengio, 2017; Keskar, Mudigere, Nocedal, Smelyanskiy, &

Tang, 2017). Several works have further illuminated the interplay between optimization dynamics and loss landscape geometry. For instance, Chaudhari et al. (2019) introduce Entropy-SGD, which augments the loss function with an entropy term $-\beta^{-1}\,\mathrm{H}(\boldsymbol{\theta})$ to the loss:

$$\tilde{\mathcal{L}}(\boldsymbol{\theta}) \;=\; \mathcal{L}(\boldsymbol{\theta}) \;-\; \beta^{-1}\,\mathrm{H}(\boldsymbol{\theta}), \quad \mathrm{H}(\boldsymbol{\theta}) \;=\; -\int p(\boldsymbol{\phi}|\boldsymbol{\theta})\,\ln p(\boldsymbol{\phi}|\boldsymbol{\theta})\,d\boldsymbol{\phi},$$

where β is a temperature-like parameter and $p(\boldsymbol{\phi}|\boldsymbol{\theta})$ is a local Gibbs distribution around $\boldsymbol{\theta}$. Similarly, Xie et al. (2021) develops a diffusion theory for deep learning dynamics, modeling the update of parameters $\boldsymbol{\theta}^{(t)}$ under SGD as

$$\boldsymbol{\theta}^{(t+1)} \;=\; \boldsymbol{\theta}^{(t)} \;-\; \alpha\,\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}^{(t)}) \;+\; \sqrt{2\,\alpha\,T_{\text{eff}}}\,\xi^{(t)},$$

where $\xi^{(t)}$ is an isotropic noise term and T_{eff} is an effective temperature capturing batch-size and learning-rate dependence. Their analysis shows that the resulting parameter distribution is biased exponentially toward regions where $H(\boldsymbol{\theta})$ has smaller eigenvalues, thus promoting flatter minima. Their results quantitatively show that SGD exponentially favors flat minima, offering a theoretical underpinning for the empirically observed link between flatness and generalization. The concept of flatness is further enriched by the study of asymmetric valleys. H. He, Huang, and Yuan (2019) observe that the local minima in deep networks are often asymmetric: the loss increases abruptly in some directions while rising more gradually in others. Under mild assumptions, they prove that solutions biased toward the flat side of an asymmetric valley generalize better than the exact empirical minimizers. This finding is supported by weight-averaging techniques such as Stochastic Weight Averaging (SWA) (Izmailov, Podoprikhin, Garipov, Vetrov, & Wilson, 2018), which implicitly guide the model parameters toward flatter regions. Finally, optimization methods that explicitly target flatter regions—such as Sharpness-Aware Minimization (SAM) (Foret, Kleiner, Mobahi, & Neyshabur, 2021)—modify the training objective to explicitly penalize local sharpness by modifying the training objective to

$$\mathcal{L}_{\text{SAM}}(\boldsymbol{\theta}) \;=\; \min_{\boldsymbol{\theta}} \max_{\|\delta\|\le\rho} \mathcal{L}(\boldsymbol{\theta}+\delta), \tag{4}$$

with a radius parameter $\rho > 0$. By ensuring robust performance in a neighborhood around $\boldsymbol{\theta}$, SAM systematically favors flatter basins of the loss landscape.

Consequently, these studies underscore that a comprehensive understanding of loss landscape geometry—including factors such as curvature, asymmetry, and the connectivity of low-loss regions—is crucial not only for explaining the dynamics of SGD but also for designing algorithms that enhance generalization in deep networks.

2.2.3 Implicit Regularization

Implicit regularization refers to the phenomenon by which the training algorithm itself biases the optimization process towards solutions with desirable generalization properties, even in the absence of explicit regularization terms. A growing body of evidence suggests that the dynamics of methods such as SGD inherently favor flat minima, which are linked to improved robustness and generalization Frankle and Carbin (2019); Kalra and Barkeshli (2023). This effect arises partly because the noise injected by stochastic sampling encourages exploration of the loss landscape, while the iterative nature of gradient descent effectively smooths out sharp curvatures.

The interplay between implicit regularization and loss landscape geometry is complex. For instance, weight decay—commonly applied as an explicit regularizer—has been shown to interact with the gradient dynamics in ways that further promote the selection of flatter minima Xie, zhiqiang xu, Zhang, Sato, and Sugiyama (2023). Moreover, while adaptive optimization methods (e.g., Adam (Kingma & Ba, 2015), RAdam (L. Liu et al., 2020)) often accelerate convergence, they may sometimes undermine the implicit bias towards flatness, leading to a trade-off between optimization speed and generalization quality (L. Liu et al., 2020). Recent studies have also drawn connections between implicit regularization and the lottery ticket hypothesis (Frankle & Carbin, 2019), which posits that *sparse subnetworks* within over-parameterized models can perform comparably to the full network when appropriately trained. Such observations highlight the multifaceted role of implicit regularization in deep learning.

In a nutshell, these three facets—training dynamics and phase transitions, loss landscape geometry, and implicit regularization—provide a framework for understanding how deep networks learn and generalize. The following sections will build on this foundation to discuss the key challenges

in optimizing deep networks and survey a range of optimization methods that have been developed to tackle these issues.

2.3 Key Challenges in Optimizing Deep Networks

The optimization of deep networks presents several interrelated challenges that stem from the inherent complexity of the models and the high-dimensional, non-convex nature of their loss landscapes. In this section, we discuss three core challenges: the difficulties posed by non-convex optimization, the delicate balance between achieving generalization and avoiding overfitting, and the computational constraints that arise when scaling optimization algorithms to large networks.

2.3.1 Non-Convex Optimization

Deep networks are trained by minimizing loss functions that are highly non-convex, characterized by a profusion of local minima, saddle points, and flat regions as illustrated in Figure 2.2. This non-convexity complicates the task of finding globally optimal solutions and often necessitates reliance on local search methods. SGD and its variants have emerged as the workhorses for deep learning partly because the inherent noise in SGD helps in escaping shallow local minima and saddle points (Y. N. Dauphin et al., 2014; Kalra & Barkeshli, 2023). Moreover, several studies have shown that the structure of the loss landscape is such that many local minima are of comparable quality in terms of generalization (Choromanska, Henaff, Mathieu, Arous, & LeCun, 2015; yeh Chiang et al., 2023). This observation implies that even if the global optimum is not reached, the solution found by SGD can still generalize well. To further navigate the complex curvature of the loss surface, second-order information is often exploited. Approaches such as Kronecker-Factored Approximate Curvature (K-FAC) (Martens & Grosse, 2015a) and its efficient variants (Benzing, 2022; Mozaffari, Li, Zhang, & Dehnavi, 2023; Tang et al., 2021; L. Zhang, Shi, & Li, 2023) seek to provide accurate curvature approximations that help steer the optimization process. However, their increased computational overhead remains a significant challenge, especially in large-scale settings. In parallel, classical optimization techniques such as the Limited-memory BFGS algorithm (L-BFGS) (D. C. Liu & Nocedal, 1989) have also influenced modern algorithmic developments. While

L-BFGS is capable of providing precise curvature information, its direct application in stochastic, high-dimensional environments typical of deep learning is hindered by sensitivity to noise and scalability issues. As a result, recent work has focused on hybrid strategies that combine the efficiency of first-order methods with selective second-order insights, aiming to better exploit the geometry of the loss surface without incurring prohibitive computational costs.

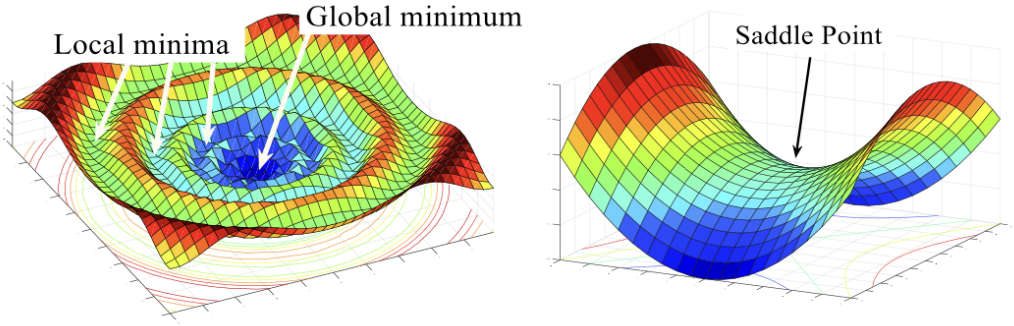

Figure 2.2: Three types of stationary points in non-convex optimization landscapes: local minima, global minima, and saddle points.

2.3.2 Generalization vs. Overfitting

A central paradox in deep learning is that over-parameterized models can achieve excellent generalization performance despite their capacity to overfit. The challenge lies in steering the optimization process toward solutions that generalize well rather than merely fitting the training data. Implicit regularization, inherent in methods like SGD, plays a critical role by biasing training towards flatter minima—regions in the loss landscape that are less sensitive to perturbations and are empirically linked to improved generalization (Foret et al., 2021; Soudry, Hoffer, Nacson, Gunasekar, & Srebro, 2018; Xie et al., 2023). Explicit regularization techniques such as weight decay and dropout have traditionally been employed to mitigate overfitting. However, recent research suggests that the choice of optimization algorithm itself can exert a significant regularizing effect. For instance, adaptive methods like Adam and its variants—including AdamP and AdaBelief (Zhuang et al., 2020)—are known to accelerate convergence; yet, they may sometimes sacrifice the

implicit bias toward flat minima that is beneficial for generalization (Wilson, Roelofs, Stern, Srebro, & Recht, 2017). Moreover, margin-based analyses have provided theoretical insights into how the implicit regularization of gradient-based methods can yield better generalization bounds even in over-parameterized settings (Bartlett, Foster, & Telgarsky, 2017). The ongoing investigation into the interplay between these dynamics is further enriched by studies on the lottery ticket hypothesis (Frankle & Carbin, 2019), which posit that certain sparse subnetworks are inherently predisposed to generalize well. Collectively, these insights continue to drive the development of optimization strategies that aim to balance fitting accuracy with robust generalization.

2.3.3 Computational Constraints

The high computational cost associated with training deep networks is a persistent challenge, particularly as models continue to scale in size and complexity. While second-order methods, which utilize curvature information, can in principle lead to faster convergence, their direct application is often hindered by prohibitive memory and time requirements. Methods such as Shampoo (V. Gupta et al., 2018) and its variants attempt to incorporate second-order information through Kronecker decompositions, yet these approaches can be computationally intensive, especially for large-scale models.

To address these issues, researchers have proposed a variety of efficient approximations and hybrid methods. For example, efficient implementations of K-FAC (Botev, Ritter, & Barber, 2017; Frantar, Kurtic, & Alistarh, 2021; George, Laurent, Bouthillier, Ballas, & Vincent, 2018) and recent proposals such as FOOF (Benzing, 2022) and M-FAC (Frantar et al., 2021) aim to strike a balance between leveraging curvature information and maintaining computational feasibility. Additionally, first-order methods, including adaptive optimizers like RAdam and AdaFactor (Shazeer & Stern, 2018), continue to be refined to reduce overhead while preserving convergence quality. The constant drive to lower computational burdens has not only spurred algorithmic innovations but also influenced hardware-aware optimizations, ensuring that deep network training remains tractable on modern computational platforms.

Hence, the challenges of non-convex optimization, balancing generalization with overfitting,

and managing computational constraints define the current landscape of deep network optimization. Addressing these challenges continues to be a major focus of both theoretical investigations and practical algorithm design, as evidenced by the ongoing contributions to the machine learning community.

2.4 Optimization Methods

Optimization methods for deep networks can be categorized into four main groups: zeroth-order, first-order, second-order, and hybrid methods. Each category leverages different types of information and offers distinct trade-offs in computational cost, convergence behavior, and memory requirements. In this section, we provide detailed mathematical descriptions of these methods and refer to prominent works published in top conferences.

2.4.1 Zeroth-Order Methods

Zeroth-Order (ZO) methods, also known as *derivative-free* optimization techniques, form a class of algorithms that rely exclusively on function evaluations rather than on explicit gradient information. These approaches prove particularly useful in settings where gradients are unavailable, unreliable, or prohibitively expensive to compute. Applications span a wide range of areas, including black-box optimization, reinforcement learning, adversarial attacks, and other non-differentiable scenarios.

Zeroth-Order Optimization in NNs. ZO methods address the parameter optimization problem

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^P} \mathcal{L}(\boldsymbol{\theta}) \tag{5}$$

for an L-layer NN $f_{\boldsymbol{\theta}}$ *without* using backpropagation. Here, $P = \sum_{i=1}^{L} P_i$ is the total number of parameters. Instead of computing the parameter gradient via automatic differentiation, these methods approximate $\nabla_\theta \mathcal{L}(\boldsymbol{\theta})$ by sampling *function evaluations* of the loss landscape. When the

network is L-smooth, meaning there exists a constant $L > 0$ such that

$$\|\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) \;-\; \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}')\| \;\leq\; L\|\boldsymbol{\theta} \;-\; \boldsymbol{\theta}'\|,$$

a classical *coordinate-wise* finite-difference estimator takes the form

$$\widehat{\partial_{\theta_i}\mathcal{L}}(\boldsymbol{\theta}) \;=\; \frac{\mathcal{L}(\boldsymbol{\theta} + h\,\mathbf{e_i}) \;-\; \mathcal{L}(\boldsymbol{\theta})}{h},$$

where $\mathbf{e_i}$ is the i^{th} canonical basis vector in $\mathbb{R}^P$, and $h > 0$ is a small finite-difference step size (or *perturbation scale*). This method requires $\mathcal{O}(P)$ function evaluations per update, which can be prohibitive for large P (Golovin et al., 2020).

Modern implementations often opt for *random directional* derivatives that perturb all parameters simultaneously. A typical estimator is

$$\widehat{\nabla}_{\boldsymbol{\theta}}\,\mathcal{L}(\boldsymbol{\theta}) \;=\; \frac{\mathcal{L}\big(\boldsymbol{\theta} + h\,\mathbf{u}\big) \;-\; \mathcal{L}(\boldsymbol{\theta})}{h}\,\mathbf{u}, \quad \mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_P),$$

where $\mathbf{u}$ is drawn from a spherically symmetric distribution (e.g., a Gaussian). In expectation, this *directional* scheme optimizes a smoothed version of the loss

$$\mathcal{L}_h(\boldsymbol{\theta}) \;=\; \mathbb{E}_{\mathbf{u}}\big[\mathcal{L}(\boldsymbol{\theta} + h\,\mathbf{u})\big], \quad \mathbb{E}_{\mathbf{u}}\big[\widehat{\nabla}_{\boldsymbol{\theta}}\,\mathcal{L}(\boldsymbol{\theta})\big] \;=\; \nabla_{\boldsymbol{\theta}}\mathcal{L}_h(\boldsymbol{\theta}),$$

which provides a form of regularization in high-dimensional spaces (Ghadimi & Lan, 2013).

The parameter updates mirror those of SGD (see Eq. (2)) but employ estimated gradients

$$\boldsymbol{\theta}^{(t+1)} \;=\; \boldsymbol{\theta}^{(t)} \;-\; \alpha^{(t)}\,\widehat{\nabla}_{\boldsymbol{\theta}}\,\mathcal{L}\big(\boldsymbol{\theta}^{(t)}\big).$$

Under bounded variance, i.e. $\mathbb{E}\big[\|\widehat{\nabla}_{\boldsymbol{\theta}}\,\mathcal{L}(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}}\,\mathcal{L}(\boldsymbol{\theta})\|^2\big] \leq \sigma^2$, and assuming L-smoothness, the average squared gradient norm over T iterations satisfies

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\big[\|\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}^{(t)})\|^2\big] \;\leq\; \mathcal{O}\Big(\frac{L\big(\mathcal{L}\big(\boldsymbol{\theta}^{(1)}\big) - \mathcal{L}^*\big)}{T} \;+\; \frac{P\,L\,\sigma^2}{T}\Big),$$

where $\mathcal{L}^*$ is the global optimum (Ghadimi & Lan, 2013). Consequently, to achieve an ϵ-stationary solution, one needs $\mathcal{O}\big(\frac{P}{\epsilon^2}\big)$ iterations, reflecting the *dimension dependence* in zeroth-order estimates.

Parallelization can partially mitigate this cost. For instance, Lian, Zhang, Hsieh, Huang, and Liu (2016) show that with K parallel workers and communication delay bound τ, the iteration complexity improves to $\mathcal{O}\big(\frac{P}{K\,\epsilon^2}\big)$ under certain conditions, maintaining a near-linear speedup when $K \leq \mathcal{O}\big(T^{1/4}\big)$. Additionally, Al-Dujaili and O'Reilly (2020) propose *sign-based* compression of the updates

$$\boldsymbol{\theta}^{(t+1)} \;=\; \boldsymbol{\theta}^{(t)} \;-\; \alpha^{(t)}\,\mathrm{sign}\Big(\widehat{\nabla}_{\boldsymbol{\theta}}\mathcal{L}\big(\boldsymbol{\theta}^{(t)}\big)\Big),$$

which reduces communication overhead. This approach is further motivated by the finding that $\mathrm{sign}\big(\nabla_{\boldsymbol{\theta}}\mathcal{L}\big)$ often preserves sufficient information about the descent direction, especially when coupled with adaptive step sizes (Y. Zhao et al., 2025). Such sign-based updates highlight the broad flexibility of zeroth-order optimization in large-scale, high-dimensional NN training.

Recent Advances and Practical Successes. Driven by the need to make derivative-free methods more efficient, recent techniques incorporate adaptive sampling and second-order insights. For instance, MeZO (Malladi et al., 2023) circumvents backpropagation by directly estimating gradients through carefully designed perturbations in a *symmetric* manner

$$\widehat{\nabla}_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) \;=\; \frac{\mathcal{L}(\boldsymbol{\theta}+\epsilon\,\mathbf{z}) - \mathcal{L}(\boldsymbol{\theta}-\epsilon\,\mathbf{z})}{2\,\epsilon}\,\mathbf{z},$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_P)$. Under $\mathcal{L} \in C^3$, this estimator yields $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$ up to $\mathcal{O}(\epsilon^2)$ (X. Chen, Liu, Xu, et al., 2019; Malladi et al., 2023), and storing random seeds rather than full vectors $\mathbf{z}$ improves memory usage. Y. Zhao et al. (2025) combine variance reduction and adaptive learning rates for faster convergence in high-dimensional settings, extending the work of Ghadimi and Lan (2013); Lian et al. (2016) to show that under unbiasedness and bounded-variance assumptions, zeroth-order updates can inherit many classical SGD convergence properties. Other influential contributions have broadened the scope of derivative-free optimization. Golovin et al. (2020) highlight the challenges of exponential growth in function evaluations with dimension, emphasizing the importance

of efficient sampling schemes. Rando, Molinari, Rosasco, and Villa (2024) reduce variance via smoothing-based estimators, and Larson, Menickelly, and Wild (2019) address adversarial noise with robust ZO methods. Y. Zhang et al. (2022) propose flexible exploration-exploitation strategies for randomized search. Collectively, these studies underscore that while zeroth-order methods can be a powerful alternative when gradients are unavailable or unreliable, their practical utility depends heavily on *reducing* the number of function evaluations.

Moreover, other methods such as ZO-SVRG (Zeroth-Order Stochastic Variance Reduced Gradient) (S. Liu et al., 2018) employs control variates to reduce variance in the gradient approximation. Specifically, it defines

$$\mathbf{m}^{(t)} = \widehat{\nabla}_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}^{(t)}) - \widehat{\nabla}_{\boldsymbol{\theta}}\mathcal{L}(\tilde{\boldsymbol{\theta}}) + \tilde{\boldsymbol{\mu}}, \quad \tilde{\boldsymbol{\mu}} = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \widehat{\nabla}_{\boldsymbol{\theta}}\mathcal{L}_i(\tilde{\boldsymbol{\theta}}).$$

where $\tilde{\boldsymbol{\theta}}$ is a reference parameter (typically updated periodically), and $\tilde{\boldsymbol{\mu}}$ is computed as the average of the zeroth-order gradient estimates over a mini-batch $\mathcal{B}$. This estimator combines the current gradient approximation $\widehat{\nabla}_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}^{(t)})$ with a correction term that accounts for the discrepancy between the current parameter and the reference parameter, thus reducing the overall variance of the estimate. The update rule for the parameters is then given by

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha\, \mathbf{m}^{(t)},$$

where α is the learning rate. By leveraging both the current and historical gradient information, ZO-SVRG is able to reduce the variance from $\mathcal{O}(d/\sqrt{T})$ to $\mathcal{O}(d/T + 1/b)$, thereby stabilizing convergence even in high-dimensional settings.

Furthermore, the adaptive framework has been also used in ZO methods as demonstrated by ZO-AdaMM (X. Chen, Liu, Xu, et al., 2019) which extends Adam into a zeroth-order framework by iterating

$$\mathbf{m}^{(t+1)} = \beta_1\, \mathbf{m}^{(t)} + (1-\beta_1)\, \widehat{\nabla}_{\boldsymbol{\theta}}\mathcal{L}\big(\boldsymbol{\theta}^{(t+1)}\big), \quad \mathbf{v}^{(t+1)} = \beta_2\, \mathbf{v}^{(t)} + (1-\beta_2)\, \big(\widehat{\nabla}_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}^{(t+1)})\big)^2, \tag{6}$$

the update rule is $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \frac{\mathbf{m}^{(t)}}{\sqrt{\mathbf{v}^{(t)}}+\delta}$, where $\beta_1, \beta_2 \in (0,1)$ and $\delta > 0$ is a regularization constant. This per-parameter adaptation remains valid in a ZO setting provided that the expected magnitude of $(\mathbf{v}^{(t)})^{-1/2}\, \widehat{\nabla}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^{(t)})$ remains bounded.

Integrating second-order free information has also been studied by several works (Bollapragada & Wild, 2023; Y. Zhao et al., 2025), which have explored approximating second-order information without explicit differentiation. In the context of ZO methods, one seeks to recover curvature information through function evaluations rather than analytic derivatives. This is particularly beneficial when gradients or Hessians are either unavailable or too expensive to compute. A common strategy is to estimate Hessian actions via *finite-difference approximations*. For example, a quasi-Newton scheme might estimate the product of the Hessian $H^{(t)}$ with a vector $\mathbf{z}$ by using the relation

$$H^{(t)}\,\mathbf{z} \;\approx\; \frac{\mathcal{L}\big(\boldsymbol{\theta}^{(t)} + h^{(t)}\,\mathbf{z} + h^{(t)}\,\mathbf{u}\big) \;-\; \mathcal{L}\big(\boldsymbol{\theta}^{(t)} + h^{(t)}\,\mathbf{u}\big)}{(h^{(t)})^2},$$

where $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_P)$ serves as a random direction that, when combined with the directional perturbation $\mathbf{z}$, yields an approximation of the second-order behavior along that direction. The key idea is that the change in the loss function, when evaluated at points that are slightly perturbed in both the desired and a random direction, encodes information about the curvature of the loss surface. This approach circumvents the need for explicit Hessian computation while still capturing essential curvature details. Subsequently, one can employ a BFGS update to build an approximation $B^{(t)} \approx (H^{(t)})^{-1}$ of the inverse Hessian. This approximated inverse Hessian is then used to precondition the gradient, leading to a curvature-informed update of the form,

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha^{(t)}\, B^{(t)}\, \widehat{\nabla}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^{(t)}),$$

where $\widehat{\nabla}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^{(t)})$ represents a ZO estimate of the gradient. By adjusting the descent direction based on the local curvature, this method is capable of accelerating convergence, especially in ill-conditioned landscapes where the loss surface may exhibit steep or flat regions. Although these methods are computationally heavier due to the additional function evaluations and the complexity of updating the inverse Hessian approximation, the integration of curvature information can lead to

more robust and efficient optimization. In particular, the ability to modulate the step direction and length based on local curvature often results in improved convergence behavior, making these techniques an attractive alternative in settings where traditional gradient-based methods may struggle (Shu et al., 2023).

Summary and Outlook. In summary, zeroth-order methods offer a derivative-free alternative for optimizing NNs, relying solely on function evaluations instead of explicit gradient computations. This characteristic makes them particularly attractive in scenarios where gradients are unavailable, unreliable, or computationally expensive, such as in black-box optimization, reinforcement learning, and adversarial attacks. By approximating gradients through techniques like coordinate-wise finite differences or random directional derivatives, these methods enable optimization in non-differentiable settings, albeit at the cost of increased function evaluations and inherent dimension dependence. Recent advances, including variance reduction schemes such as ZO-SVRG and adaptive frameworks like ZO-AdaMM, have significantly improved the efficiency and stability of zeroth-order optimization. Furthermore, the incorporation of second-order information via finite-difference approximations and quasi-Newton updates has enhanced convergence in ill-conditioned landscapes by providing curvature-aware steps. While challenges remain—particularly regarding the high computational overhead in high-dimensional spaces—the ongoing development of efficient sampling, parallelization, and adaptive techniques holds promise for expanding the practical applicability of derivative-free methods in large-scale deep learning and beyond.

2.4.2 First-Order Methods

First-order methods remain the most widely used optimization techniques in deep learning, as they rely on gradient information to iteratively update model parameters (Goodfellow, 2016). We consider a function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ to be minimized with respect to some parameters $\boldsymbol{\theta}$. In its simplest form, one solves

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^d} \mathcal{L}(\boldsymbol{\theta}), \quad \mathbf{g}^{(t)} = \nabla \mathcal{L}(\boldsymbol{\theta}) \in \mathbb{R}^d, \tag{7}$$

via the *gradient descent* update defined in Eq. (2). Note that the formulation defined in Eq. (7) is similar to the one defined in Eq. (5), except that we now compute the actual gradient value instead of estimating it. Due to the large scale of modern datasets and models, practitioners typically rely on *mini-batch* SGD where $\mathbf{g}^{(t)}$ is substituted by an unbiased gradient estimate computed on a random mini-batch of training data, i.e. $\widehat{\nabla}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^{(t)})$ in Eq. (2). This stochastic approach not only reduces computation but also helps escape sharp minima and saddle points due to the induced noise the computed gradient (X. Chen, Liu, Sun, & Hong, 2019a; Y. Dauphin, De Vries, & Bengio, 2015; Reddi, Kale, & Kumar, 2018).

SGD-Based and Momentum Methods. A straightforward yet crucial refinement of SGD is to incorporate *momentum*, a technique originally studied by Polyak (1964) and subsequently refined for NNs (Sutskever, Martens, Dahl, & Hinton, 2013a). In its classical form, one initializes $\boldsymbol{\theta}_0$ and $\mathbf{m}_0 = \mathbf{0}$, then iterates

$$\mathbf{m}^{(t+1)} = \beta\, \mathbf{m}^{(t)} - \mathbf{g}^{(t)}, \tag{8}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \alpha^{(t)}\, \mathbf{m}^{(t+1)}, \tag{9}$$

where $\mathbf{m}^{(t)} \in \mathbb{R}^d$ is a *velocity* term, $\beta \in (0, 1)$ is the damping coefficient, and $\alpha^{(t)} > 0$ denotes the learning rate schedule. By accumulating gradients in $\mathbf{m}^{(t)}$, momentum can significantly speed up convergence, particularly if β is well-chosen. However, if β is set too small, progress in low-curvature directions may stall, whereas a large β risks instabilities and oscillations in high-curvature regions.

Nesterov Accelerated Gradient (NAG) (Nesterov, 1983; Sutskever et al., 2013a) further refines classical momentum by evaluating the gradient at a look-ahead position, effectively correcting errors introduced by moving in the direction of the velocity in Eq. (8). One writes

$$\mathbf{m}^{(t+1)} = \beta\, \mathbf{m}^{(t)} - \nabla_{\boldsymbol{\theta}} \mathcal{L}\big(\boldsymbol{\theta}^{(t)} + \alpha^{(t)}\, \beta\, \mathbf{m}^{(t)}\big),$$

and apply $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \mathbf{m}^{(t+1)}$. Nesterov's modification helps stabilize updates in non-convex

optimization and can accelerate convergence in quadratic settings by implicitly adapting to the curvature.

Beyond these foundational momentum schemes, numerous enhancements address high-dimensional or otherwise challenging NN landscapes. For example, Lion method (X. Chen et al., 2024) incorporates a diagonal preconditioner into the momentum update. Formally, one can write the update as

$$\mathbf{m}^{(t+1)} = \beta \mathbf{m}^{(t)} - (1-\beta)\, P \odot \mathbf{g}^{(t)},$$

where P is a diagonal matrix that rescales the gradient along each coordinate and applies Eq. (9). The intuition behind this approach is to adaptively adjust the learning rate in each dimension based on local curvature, thereby enhancing numerical conditioning and promoting a more balanced descent across the parameter space. GaLore method (J. Zhao et al., 2024) adopts a complementary strategy by projecting the gradient into a reduced-dimensional subspace. Let $U \in \mathbb{R}^{d\times k}$ be an orthonormal matrix whose columns form a basis for a subspace with $k \ll d$. The projected gradient is then given by

$$(\mathbf{g}^{(t)})^{\text{proj}} = UU^{\top}\mathbf{g}^{(t)}, \tag{10}$$

and integrating Eq. (10) into Eq. (8), the momentum update becomes

$$\mathbf{m}^{(t+1)} = \beta \mathbf{m}^{(t)} + (1-\beta)\, (\mathbf{g}^{(t)})^{\text{proj}}.$$

This projection effectively retains the most critical descent directions while significantly reducing memory usage. The key concept is that even in a high-dimensional setting, the most informative directions for descent may lie on a low-dimensional manifold, thus allowing the method to discard redundant information without compromising convergence. Moreover, WIN method (Zhou, Xie, & YAN, 2022) fuses weight decay with a Nesterov-like momentum framework. In this method, the

velocity update is modified to incorporate weight regularization directly into the gradient accumulation. i.e.

$$\mathbf{m}^{(t+1)} = \beta \mathbf{m}^{(t)} - (1-\beta)\left(\mathbf{g}^{(t)} + \lambda \boldsymbol{\theta}^{(t)}\right),$$

where λ is the weight decay coefficient. The update rule is similar as Eq. (9). By integrating weight decay into the momentum term, WIN simultaneously enforces regularization and benefits from the anticipatory nature of Nesterov momentum, which results in smoother training trajectories and enhanced convergence behavior.

Despite their diverse implementations, these momentum-oriented methods share a consistent theme: the effective use of a velocity term, which is carefully maintained and updated, can significantly accelerate deep learning optimization. Crucially, the success of these approaches hinges on the appropriate tuning of damping factors and step sizes to balance acceleration with stability.

Adaptive Adam-Type Approaches. While momentum methods amplify or dampen the *direction* of the raw gradient signal, a complementary branch of first-order optimizers focuses on *adaptively* scaling the *magnitude* of each parameter's update. Early pioneers in this category include Adagrad (Duchi, Hazan, & Singer, 2011) and RMSProp (Hinton, Srivastava, & Swersky, 2012). Adagrad accumulates the squared gradients over time, thus adjusting the learning rate based on the cumulative variance of each parameter's gradient. Formally, the parameter update then becomes

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \frac{\mathbf{g}^{(t)}}{\sqrt{\sum_{\tau=1}^{t} (\mathbf{g}^{(\tau)})^2} + \epsilon},$$

where $\epsilon > 0$ avoids division by zero. Although Adagrad excels at handling sparse features by boosting the effective learning rate for infrequently updated parameters, its global learning rate can diminish excessively over time. RMSProp addresses this by introducing an exponential moving average of the squared gradients rather than a strict sum

$$\mathbf{v}^{(t+1)} = \beta\, \mathbf{v}^{(t)} + (1-\beta)\, (\mathbf{g}^{(t)})^2, \quad \boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \frac{\mathbf{g}^{(t)}}{\sqrt{\mathbf{v}^{(t+1)}} + \epsilon},$$

where $\beta \in (0, 1)$. By continuously discounting older gradients, RMSProp better balances fast adaptation with sustained learning. Adam can be viewed as combining RMSProp's second-moment tracking with a momentum-like first-moment term, i.e.

$$\mathbf{m}^{(t+1)} = \beta_1 \mathbf{m}^{(t)} + (1 - \beta_1)\, \mathbf{g}^{(t)}, \quad \mathbf{v}^{(t+1)} = \beta_2 \mathbf{v}^{(t)} + (1 - \beta_2)\, \left(\mathbf{g}^{(t)}\right)^2, \tag{11}$$

where $\beta_1, \beta_2 \in (0, 1)$. Note that Eq. (6) is the same as Eq. (11) except that the gradient is estimated rather than explicitly computer. After correcting for initialization bias, the update becomes

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \frac{\mathbf{m}^{(t+1)}/(1 - \beta_1^{(t+1)})}{\sqrt{\mathbf{v}^{(t+1)}/(1 - \beta_2^{(t+1)})} + \epsilon}.$$

This per-parameter scaling often yields rapid convergence and robustness to HP misconfiguration, making Adam and its variants among the most widely adopted optimizers in deep learning. This per-parameter adaptation often yields faster convergence and greater numerical stability. Building on Adam's popularity, several adaptive methods have been proposed to mitigate specific challenges in large-scale NN optimization. AdaFactor addresses memory constraints by factorizing the second-moment estimates. For a matrix-valued parameter $\Theta^{(t)} \in \mathbb{R}^{m \times n}$ with gradient $G^{(t)}$ (here $\Theta^{(t)} = \text{vec}^{-1}(\boldsymbol{\theta}^{(t)})$ and $G^{(t)} = \text{vec}^{-1}(g^{(t)})$), instead of storing a full second-moment tensor, AdaFactor computes two accumulators—a row vector $\mathbf{r}^{(t)} \in \mathbb{R}^m$ and a column vector $\mathbf{c}^{(t)} \in \mathbb{R}^n$—updated as

$$\mathbf{r}_i^{(t+1)} = \beta_2\, \mathbf{r}_i^{(t)} + (1 - \beta_2) \sum_{j=1}^{n} \left(G_{ij}^{(t+1)}\right)^2, \quad \mathbf{c}_j^{(t+1)} = \beta_2\, \mathbf{c}_j^{(t)} + (1 - \beta_2) \sum_{i=1}^{m} \left(G_{ij}^{(t+1)}\right)^2.$$

The effective second moment for each entry is then approximated via a factorization of these statistics,

$$V_{ij}^{(t+1)} = \frac{\mathbf{r}_i^{(t+1)}\, \mathbf{c}_j^{(t+1)}}{\sum_{j=1}^{n} \mathbf{c}_j^{(t+1)}}.$$

Given this approximation, the update rule for the parameter matrix becomes

$$\Theta^{(t+1)} = \Theta^{(t)} - \alpha \frac{G^{(t+1)}}{\sqrt{V^{(t+1)}} + \epsilon}. \tag{12}$$

This update mechanism allows AdaFactor to dramatically reduce memory overhead while still capturing essential second-order information through the factorized approximation, making it particularly attractive for large-scale models such as Transformers (Vaswani et al., 2017). To continue, RAdam aims to improve Adam's instability during early training by rectifying the variance of the adaptive learning rate. It first computes a statistic

$$\rho^{(t)} = \rho_\infty - \frac{2t\beta_2^{(t)}}{1-\beta_2^{(t)}}, \qquad \text{with} \qquad \rho_\infty = \frac{2}{1-\beta_2} - 1,$$

which indicates whether the variance of the second-moment estimate has stabilized. When $\rho^{(t)}$ is sufficiently large, a rectification factor $\phi(\rho^{(t)})$ is applied to the update

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \frac{\phi(\rho^{(t+1)})\, \mathbf{m}^{(t+1)}}{\sqrt{\mathbf{v}^{(t+1)}} + \epsilon},$$

where $\phi(\rho^{(t)})$ is defined as

$$\phi(\rho^{(t)}) = \begin{cases} \sqrt{\frac{(\rho^{(t)}-4)(\rho^{(t)}-2)\rho_\infty}{(\rho_\infty-4)(\rho_\infty-2)\rho^{(t)}}}, & \text{if } \rho^{(t)} > 4, \\ 1, & \text{otherwise.} \end{cases}$$

This rectification ensures that, during the early stages of training, the update resembles SGD with momentum, and as the second-moment estimate stabilizes, the optimizer gradually transitions to a fully adaptive behavior. Moreover, AdaBound (Luo, Xiong, & Liu, 2019) mitigates extreme early behavior by dynamically bounding the effective learning rate. In this method, the adaptive learning rate is clipped between two time-dependent bounds $\alpha_{\text{low}}(t)$ and $\alpha_{\text{high}}(t)$, leading to the modified step size

$$\hat{\alpha}^{(t)} = \text{clip}\left(\frac{\alpha}{\sqrt{\mathbf{v}^{(t)}} + \epsilon},\, \alpha_{\text{low}}(t),\, \alpha_{\text{high}}(t)\right),$$

with the update rule given by $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \hat{\alpha}^{(t+1)}\, \mathbf{m}^{(t+1)}$. Here, α denotes the base learning rate, which sets the overall scale of the update before the adaptive scaling and clipping are applied. As the bounds tighten over time, the algorithm increasingly resembles vanilla SGD, which can yield improved generalization in the later stages of training. Then, AdaBelief modifies Adam's second-moment estimation by tracking the deviation of the observed gradient from its exponential moving average. The second-moment accumulator is updated as

$$\mathbf{v}^{(t+1)} = \beta_2 \mathbf{v}^{(t)} + (1-\beta_2)(\mathbf{g}^{(t+1)} - \mathbf{m}^{(t+1)})^2.$$

The parameter update follows the familiar form $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \frac{\mathbf{m}^{(t+1)}}{\sqrt{\mathbf{v}^{(t+1)}}+\epsilon}$. By emphasizing the "belief" in the current gradient direction (i.e., penalizing unexpected deviations), AdaBelief tends to yield a more stable and reliable convergence. AdamW (Loshchilov & Hutter, 2019) resolves a conceptual inconsistency in Adam by decoupling weight decay from the gradient-based update. Instead of mixing L2 regularization into the gradient, AdamW applies weight decay as a separate multiplicative factor

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)}(1-\lambda) - \alpha \frac{\mathbf{m}^{(t+1)}}{\sqrt{\mathbf{v}^{(t+1)}}+\epsilon}.$$

where λ is the weight decay HP. This decoupling clarifies the role of regularization and has shown empirical benefits, especially in large-scale language models. Furthermore, NaDam (Dozat, 2016) is a variant of the Adam optimizer that integrates Nesterov momentum into the adaptive framework. In standard momentum-based methods, the momentum term $\mathbf{m}^{(t+1)}$, is updated as in Eq. (11). In NaDam, rather than using the momentum term alone to update the parameters, the update rule incorporates a weighted combination of both the momentum and the current gradient

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \left(\beta_1 \mathbf{m}^{(t+1)} + (1-\beta_1)\mathbf{g}^{(t+1)}\right).$$

This formulation mimics the essence of NAG where NaDam approximates this look-ahead evaluation. The intuition behind this approach is that the weighted sum $\beta_1 \mathbf{m}^{(t+1)} + (1-\beta_1)\mathbf{g}^{(t+1)}$ serves

as an effective surrogate for the gradient computed at a future point. This anticipatory update direction can lead to a more informed descent step, potentially improving convergence rates by better aligning the parameter update with the long-term trajectory of the optimization. AdaInject (Dubey et al., 2022) represents a further evolution by incorporating approximations of second-order information into an Adam-like update. In addition to the standard first and second-moment estimates, AdaInject computes a curvature-aware term $\mathbf{s}^{(t)}$ that modulates the adaptive learning rate

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \frac{\mathbf{m}^{(t+1)}}{\sqrt{\mathbf{v}^{(t+1)} + \gamma \mathbf{s}^{(t+1)}} + \epsilon},$$

where γ governs the influence of the injected second-order statistics. This integration of partial curvature information aims to combine the simplicity of first-order methods with the robustness of second-order approaches, potentially leading to more effective convergence.

Sharpness-Aware Extensions. While most first-order methods emphasize convergence speed or adaptive scaling, sharpness-aware approaches directly target the geometry of the loss surface by controlling the *sharpness* of local minima. SAM reformulates the standard objective into a minimax problem defined in Eq. (4). This formulation forces the optimization to favor parameter regions where the loss remains stable under small adversarial perturbations, effectively promoting flatter minima that are empirically associated with better generalization. In practice, SAM first computes a tentative update by taking a gradient step from θ and then re-evaluates the loss in a local neighborhood around this point, ensuring that the descent direction leads to a region of the loss surface with reduced sharpness.

Several variants extend SAM by integrating adaptive scaling or refined geometric insights. AdaSAM (Sun et al., 2024) merges the neighborhood-based minimax framework of SAM with Adam's per-parameter scaling, yielding an approach that leverages local curvature information while retaining the benefits of adaptive learning rates. ASAM (Sun et al., 2024) further refines the method by dynamically adjusting the perturbation radius ρ based on the magnitudes of the gradients, thereby fine-tuning the sensitivity of the sharpness measure to the local loss landscape. Meanwhile, GSAM (Zhuang et al., 2022) incorporates the top eigenvalue of the local Hessian into

the sharpness estimation, offering a more detailed account of the loss curvature. Collectively, these sharpness-aware extensions demonstrate that by incorporating geometric considerations into the optimization process, purely first-order methods can be enhanced to locate flatter regions of the loss surface, potentially leading to models with superior generalization.

Summary and Outlook. In summary, first-order optimizers in deep learning branch into several intertwined families. Momentum-based methods augment raw SGD to accelerate convergence and reduce erratic updates, while adaptive Adam-type approaches further modulate step sizes on a per-parameter basis, often leading to faster training and improved stability. More recently, sharpness-aware optimizers use local curvature information to search for flatter minima, reflecting broader interest in explaining why certain solutions generalize better. Despite their diversity, these methods all share a reliance on gradient information, underscoring the central role of first-order techniques in modern deep learning optimization.

2.4.3 Second-Order Methods

Second-order methods incorporate curvature information into the optimization process by leveraging the Hessian matrix or its approximations. This additional information enables more informed parameter updates compared to first-order techniques, often leading to faster convergence and improved performance, especially in regions where the loss landscape exhibits significant curvature. Consider an objective function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ that we wish to minimize with respect to the parameters $\boldsymbol{\theta}$. The classical formulation of the problem is similar to Eq. (7), but with the integration of second order information, is defined as

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \mathcal{L}(\boldsymbol{\theta}), \quad \mathbf{g}^{(t)} = \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) \in \mathbb{R}^d, \quad H(\boldsymbol{\theta}) = \nabla^2_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}). \tag{13}$$

The prototypical second-order method is Newton-Raphson's method (Dedieu, 2015), which updates the parameters according to

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - H(\boldsymbol{\theta}^{(t)})^{-1}\, \mathbf{g}(\boldsymbol{\theta}^{(t)}). \tag{14}$$

While Newton's method can achieve rapid convergence, the direct computation and inversion of the Hessian matrix $H(\boldsymbol{\theta}^{(t)})$ becomes prohibitively expensive in high-dimensional settings, such as DNNs. To overcome these limitations, a variety of approximations and alternative strategies have been developed. These methods aim to retain essential curvature information while alleviating the computational burden associated with Hessian inversion. In what follows, we categorize these second-order methods based on the techniques they employ—ranging from quasi-Newton approximations and Hessian-free methods to structured or low-rank representations of the Hessian. This taxonomy highlights the trade-offs between computational efficiency and the fidelity of curvature information, paving the way for more robust and scalable optimization strategies in large-scale deep learning.

Hessian-Based Diagonal and Quasi-Newton Methods. Many second-order optimizers aim to capture curvature information without incurring the full cost of computing and inverting the Hessian $H(\boldsymbol{\theta})$. A common strategy is to approximate the Hessian by either its diagonal or a structured factorization, thereby striking a balance between extracting useful curvature cues and maintaining computational efficiency. For instance, methods such as AdaHessian (Yao et al., 2021) and Apollo (Ma, 2020) focus on approximating the diagonal elements of the Hessian. In these approaches, for each parameter θ_i a scalar curvature estimate is obtained via

$$D_i \approx \frac{\partial^2 \mathcal{L}(\boldsymbol{\theta})}{\partial \theta_i^2},$$

which is then used to rescale the gradient in the update

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \alpha \frac{m_i^{(t)}}{\sqrt{D_i^{(t)}} + \epsilon}.$$

This rescaling captures directional curvature, allowing the optimizer to take larger steps in flat regions and more cautious updates in steep directions, thereby improving convergence behavior. However, while diagonal approximations are computationally efficient compared to the full Hessian, they inherently neglect cross-parameter interactions that can be critical in highly non-isotropic loss landscapes.

Other methods, such as Sophia (H. Liu, Li, Hall, Liang, & Ma, 2024) and INNA-Prop (Bolte, Boustany, Pauwels, & Purica, 2024), also leverage diagonal Hessian approximations or use incremental updates to refine the gradient-based steps. These techniques are often embedded within Adam-like frameworks, where the adaptive scaling is enriched by curvature estimates, thus enabling more nuanced adjustments that better reflect the underlying loss surface geometry. In contrast, M-FAC focuses on efficiently computing inverse Hessian-vector products. Instead of explicitly forming or inverting $H(\boldsymbol{\theta})$, M-FAC approximates its action on a vector by representing the Hessian as a low-rank or structured factorization,

$$H(\boldsymbol{\theta}) \approx Q \Lambda Q^{\top},$$

where Q contains dominant eigenvectors and Λ is a diagonal matrix of eigenvalues. The inverse Hessian is then approximated by

$$H^{-1}(\boldsymbol{\theta}) \approx Q \Lambda^{-1} Q^{\top},$$

which is used in the update defined in Eq. (14). This approach allows the optimizer to incorporate global curvature information while avoiding the full cost of Hessian inversion.

Classic quasi-Newton methods, such as L-BFGS, remain influential by iteratively constructing an approximation of the inverse Hessian from successive gradient differences. In L-BFGS, one collects updates $\mathbf{s}^{(t)} = \boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}$ and $\mathbf{y}^{(t)} = \mathbf{g}^{(t+1)} - \mathbf{g}^{(t)}$ to update the inverse Hessian estimate $B^{(t)}$ through formulas such as

$$B^{(t+1)} = (V^{(t)})^{\top} B^{(t)} V^{(t)} + \boldsymbol{\rho}^{(t)} \mathbf{s}^{(t)} (\mathbf{s}^{(t)})^{\top}, \tag{15}$$

with $\boldsymbol{\rho}^{(t)} = \frac{1}{(\mathbf{y}^{(t)})^{\top} \mathbf{s}^{(t)}}$ and $V^{(t)} = I - \boldsymbol{\rho}^{(t)} \mathbf{y}^{(t)} (\mathbf{s}^{(t)})^{\top}$. This quasi-Newton strategy efficiently captures curvature information from recent iterations, but its performance in deep learning is often limited by the high dimensionality of the parameter space and the stochasticity introduced by mini-batch training.

The central intuition behind these Hessian-based methods is to allow the optimizer to adjust step

sizes based on the local geometry of the loss landscape. By rescaling the gradient using curvature information, these methods enable larger updates in flat regions and more conservative moves in steep directions, which can lead to faster convergence and improved stability. Nevertheless, challenges remain. Diagonal approximations may oversimplify the curvature by ignoring off-diagonal interactions, while low-rank or quasi-Newton approximations can be sensitive to noise and may incur non-negligible computational overhead. Balancing the fidelity of curvature information with computational tractability is thus a critical design consideration in the development of effective second-order methods for deep learning.

Generalized Gauss-Newton (GGN) and Hybrid Approaches. A further category of second-order techniques leverages the GGN matrix or other surrogates for the Hessian, providing a more tractable means of incorporating curvature information into the optimization process. In many NN settings, the Hessian is both expensive to compute and can be indefinite. The GGN offers an attractive alternative by approximating the Hessian using the model's Jacobian $J(\boldsymbol{\theta})$ and the Hessian of the loss with respect to the network outputs. Specifically, the GGN is defined as

$$G(\boldsymbol{\theta}) = J(\boldsymbol{\theta})^\top \nabla^2_{f(\boldsymbol{\theta})} \mathcal{L}\big(f(\boldsymbol{\theta})\big)\, J(\boldsymbol{\theta}),$$

For many losses that are locally quadratic, such as least-squares losses, this Hessian is *positive semidefinite*, ensuring that the curvature information provided by $G(\boldsymbol{\theta})$ is both meaningful and stable.

Building on this idea, methods such as SOFO (Yu, Xia, Ma, Lengyel, & Hennequin, 2024) approximate curvature via the GGN to precondition the gradient efficiently. For instance, one might update parameters as

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \left(G(\boldsymbol{\theta}^{(t)}) + \lambda I\right)^{-1} \mathbf{g}(\boldsymbol{\theta}^{(t)}),$$

where $\lambda > 0$ is a damping factor that ensures numerical stability and I is the identity matrix. This update captures essential curvature while sidestepping the prohibitive cost of a full Hessian inversion.

Hybrid approaches further integrate second-order insights into first-order frameworks to enhance both convergence speed and generalization. LocoProp (Amid, Anil, & Warmuth, 2022) exemplifies this strategy by introducing a local second-order correction within a predominantly first-order update scheme. Rather than computing a full curvature matrix, LocoProp performs a localized approximation around the current iterate, adjusting the gradient step based on this curvature estimate. Similarly, FOSI (Sivan, Gabel, & Schuster, 2024) combines standard gradient information with a curvature correction term. A representative update rule in such hybrid methods is

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \left(\mathbf{g}(\boldsymbol{\theta}^{(t)}) + \beta \, \Delta\boldsymbol{\theta}_{\text{curv}} \right),$$

where $\Delta\boldsymbol{\theta}_{\text{curv}}$ embodies the second-order correction derived from partial curvature information and β modulates its influence. This blend of first- and second-order cues enables the optimizer to make more informed updates without incurring the full cost of second-order methods.

The intuition behind these GGN and hybrid approaches is to harness curvature information to achieve a more adaptive step size—allowing for larger steps in flat regions and more cautious moves in steep directions—thus accelerating convergence and potentially improving generalization. Nevertheless, challenges remain. While the GGN matrix is more computationally tractable than the full Hessian, its computation still incurs significant overhead in very large models, and the quality of the curvature approximation can be sensitive to the network architecture and the loss function. Moreover, hybrid methods require a delicate balance between first- and second-order contributions; an overly aggressive curvature correction may lead to instability, whereas insufficient correction may fail to realize the benefits of second-order information. Overall, these methods reflect a growing trend towards embedding partial curvature insights into existing optimization frameworks, striving for an optimal trade-off between computational cost and the accuracy of curvature information.

K-FAC and Its Extensions. *Natural gradient descent* (NGD), introduced by Amari Amari and Nagaoka (2000), leverages the FIM as a *Riemannian metric* to perform parameter updates that are invariant to reparameterization, thereby often leading to more efficient convergence than standard

gradient descent. In this framework, the update is given by

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \, (F^{(t)})^{-1} \mathbf{g}^{(t)}, \tag{16}$$

where F is the FIM. However, directly computing and inverting F is generally intractable for high-dimensional models. To address this issue, K-FAC approximates the FIM by exploiting the layer-wise structure of deep networks. Specifically, using Eq. (1) the FIM is given by

$$F = \sum_{n=1}^{N} \mathbb{E}_{\mathbf{y} \sim p(\mathbf{y}|f_{\boldsymbol{\theta}}(\mathbf{x}_n))} \left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}_n) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}_n)^\top \right] = \mathbb{E}\left[\nabla_{\boldsymbol{\theta}} \mathcal{L} \, (\nabla_{\boldsymbol{\theta}} \mathcal{L})^\top \right] = \mathbb{E}[\mathbf{g}\mathbf{g}^\top],$$

where F quantifies the expected information that an observable $\mathbf{y}$ conveys about the parameter $\boldsymbol{\theta}$. For simplicity, we write $\mathbb{E}$ in place of the full expectation $\mathbb{E}_{\mathbf{y} \sim p(\mathbf{y}|f_{\boldsymbol{\theta}}(\mathbf{x}_n))}$. K-FAC further simplifies FIM computation by adopting a block-diagonal approximation tailored to deep neural networks. By decomposing F into layer-specific blocks and employing a Kronecker-factorization based on the identity $\text{vec}(uv^\top) = v \otimes u$, K-FAC efficiently approximates each block, thereby enabling scalable natural gradient updates. K-FAC capitalizes on the structure of DNNs by simplifying the computation of the FIM via a block-diagonal approximation. In this framework, the full FIM is viewed as a block matrix with $L \times L$ blocks (for a network with L layers). The (i, j)th block is defined by

$$F_{i,j} = \mathbb{E}\Big[\text{vec}(g_i) \, \text{vec}(g_j)^\top \Big],$$

where g_i denotes the gradient with respect to the weights of the i^{th} layer. Using the Kronecker-vectorization identity, $\text{vec}(\mathbf{u}\mathbf{v}^\top) = \mathbf{v} \otimes \mathbf{u}$ (see, e.g., (**?**)), we can express the gradient for layer i as

$$g_i = \mathbf{s}_i \, \bar{\mathbf{h}}_{i-1}^\top \quad \Longrightarrow \quad \text{vec}(g_i) = \bar{\mathbf{h}}_{i-1} \otimes \mathbf{s}_i, \tag{17}$$

where $\bar{\mathbf{h}}_{i-1}$ represents the activations (augmented with a bias term) from the previous layer, and $\mathbf{s}_i$ represents the sensitivity or pre-activation derivatives at layer i.

Segmenting the FIM into layer-specific blocks, we have

$$\hat{F}_{i,j} = \mathbb{E}\Big[\text{vec}(g_i)\,\text{vec}(g_j)^\top\Big] = \mathbb{E}\Big[\bar{\mathbf{h}}_{i-1}\bar{\mathbf{h}}_{j-1}^\top \otimes \mathbf{s}_i\mathbf{s}_j^\top\Big].$$

Under the assumption that the activations $\bar{\mathbf{h}}_{i-1}$ and the pre-activation derivatives $\mathbf{s}_i$ are statistically independent, this expectation can be approximated as the Kronecker product of individual expectations

$$\hat{F}_{i,j} \approx \mathbb{E}\Big[\bar{\mathbf{h}}_{i-1}\bar{\mathbf{h}}_{j-1}^\top\Big] \otimes \mathbb{E}\Big[\mathbf{s}_i\mathbf{s}_j^\top\Big] \;=\; \mathcal{H}_{i-1,j-1} \otimes \mathcal{S}_{i,j},$$

Where $\mathcal{H}_{i-1,j-1} = \mathbb{E}\Big[\bar{\mathbf{h}}_{i-1}\bar{\mathbf{h}}_{j-1}^\top\Big]$ and $\mathcal{S}_{i,j} = \mathbb{E}\Big[\mathbf{s}_i\mathbf{s}_j^\top\Big]$ are the Kronecker Factors (KF). In practice, a further simplification is often made by assuming that the weight derivatives for different layers are uncorrelated. This leads to a block-diagonal approximation of the FIM, denoted as the Empirical FIM (EFIM)

$$\hat{F} = \text{diag}\left(\hat{F}_{1,1}, \ldots, \hat{F}_{L,L}\right) = \begin{bmatrix} \hat{F}_1 & & & \\ & \hat{F}_2 & & \\ & & \ddots & \\ & & & \hat{F}_L \end{bmatrix}.$$

This Kronecker-factorization significantly reduces the computational burden: instead of inverting a huge matrix of size proportional to the total number of parameters, one only needs to invert the much smaller matrices $\mathcal{H}_{i-1,i-1}$ and $\mathcal{S}_{i,i}$ for each layer. The resulting efficient inversion is based on the property $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$. Consequently, for a given layer i, the parameter update rule of K-FAC is expressed as

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \alpha \left((\mathcal{H}_{i-1,i-1}^{(t)})^{-1} \otimes (\mathcal{S}_{i,i}^{(t)})^{-1}\right) \nabla_{\theta_i}\mathcal{L}(\theta_i^{(t)}).$$

Figure 2.3 illustrates the computation of the EFIM via K-FAC for a given layer i.

Building upon the foundational K-FAC framework, several extensions have been proposed to address its computational and approximation challenges. For example, S-KFAC (Tang et al., 2021)

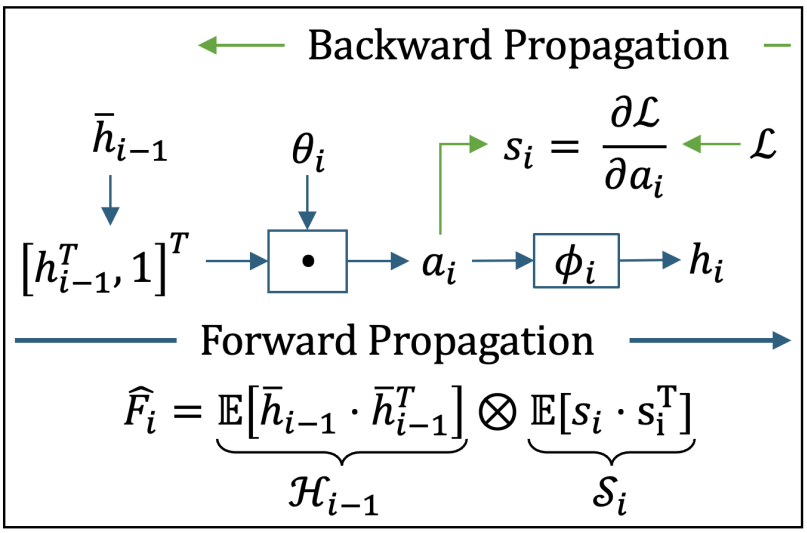

Figure 2.3: Illustration of EFIM computation using K-FAC for a given layer i.

introduces strategies to further reduce computational overhead by incorporating sparsity and more aggressive damping, modifying the update to

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \alpha \left((\mathcal{H}_{i-1,i-1}^{(t)} + \lambda I)^{-1} \otimes (\mathcal{S}_{i,i}^{(t)} + \lambda I)^{-1} \right) \nabla_{\theta_i} \mathcal{L}(\theta_i^{(t)}).$$

where λ is a damping parameter that enhances numerical stability. Eva (L. Zhang et al., 2023) refines the estimation of the covariance matrices $\mathcal{H}$ and $\mathcal{S}$ by using adaptive averaging techniques, which leads to a more accurate curvature approximation and, consequently, to faster convergence and improved generalization.

Other methods, such as FOOF and MKOR (Mozaffari et al., 2023), adopt a block-diagonal strategy that partitions the parameter space into smaller, more manageable sub-blocks, thereby reducing memory requirements and computational complexity while still capturing essential second-order information within each block. Quasi-Newton extensions such as K-BFGS (Goldfarb, Ren, & Bahamou, 2020) and KFRA (Botev et al., 2017) integrate the Kronecker-factorized approach with iterative updates reminiscent of the BFGS algorithm. In these methods, one maintains an estimate of the inverse curvature matrix $B^{(t)}$ and updates it using the standard quasi-Newton formula defined in Eq. (15). The novelty in K-BFGS and KFRA lies in organizing the updates so that the Kronecker structure is preserved, which allows these methods to capture curvature more precisely while still

benefiting from the efficiency of factorization. EKFAC (George et al., 2018) further extends the K-FAC paradigm by incorporating an eigenbasis decomposition of the covariance matrices:

$$\mathcal{H} = Q_{\mathcal{H}} \Lambda_{\mathcal{H}} Q_{\mathcal{H}}^{\top} \quad \text{and} \quad \mathcal{S} = Q_{\mathcal{S}} \Lambda_{\mathcal{S}} Q_{\mathcal{S}}^{\top}.$$

In this framework, the approximated FIM is expressed as

$$\hat{F} \approx (Q_{\mathcal{H}} \otimes Q_{\mathcal{S}})(\Lambda_{\mathcal{H}} \otimes \Lambda_{\mathcal{S}})(Q_{\mathcal{H}} \otimes Q_{\mathcal{S}})^{\top},$$

and its inversion is straightforward since the Kronecker product of the eigenvalue matrices is diagonal. This eigenbasis approach offers a diagonalized structure that balances fidelity to the true Hessian with the need for scalability in large models.

Collectively, these K-FAC extensions and related methods embody a common goal: to efficiently incorporate curvature information into the training of DNNS. By leveraging the inherent layer-wise structure and exploiting the properties of the Kronecker product, they enable second-order updates that are both computationally tractable and effective at capturing the nuanced geometry of the loss landscape. Despite their advances, challenges remain in terms of the assumptions required (e.g., independence of activations and pre-activation derivatives) and the computational overhead involved in maintaining and updating the necessary covariance estimates. Nevertheless, these methods continue to evolve, offering promising directions for scalable second-order optimization in deep learning.

Shampoo-Based Approaches. Shampoo is an influential second-order optimizer that constructs layer-wise preconditioners by leveraging the tensor structure of weight matrices. Unlike methods that restrict themselves to diagonal approximations of the curvature, Shampoo collects full second-moment statistics along different dimensions of a weight tensor. Shampoo computes two second-moment matrices,

$$G_r = \sum_t \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^{(t)}) \, \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^{(t)})^{\top} \quad \text{and} \quad G_c = \sum_t \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^{(t)})^{\top} \, \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^{(t)}),$$

which capture the curvature information along the row and column dimensions, respectively. The core idea is to form a preconditioner as a Kronecker product of the inverses of fractional powers of these matrices, leveraging the similar property that K-FAC leverages, i.e. $(G_r \otimes G_c)^{-1} = G_r^{-1} \otimes G_c^{-1}$. In practice, Shampoo typically employs a fractional exponent (often $-\frac{1}{4}$) for numerical stability, leading to the update rule

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha\, G_r^{-1/4}\, \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^{(t)})\, G_c^{-1/4},$$

This update effectively preconditions the gradient by incorporating a full-matrix approximation of the curvature, thereby allowing for more adaptive step sizes that account for the local geometry of the loss surface.

Building on the Shampoo framework, subsequent approaches have sought to integrate additional first-order insights and further reduce computational overhead. For instance, SOAP (Vyas et al., 2025) merges Shampoo's second-order preconditioning with adaptive mechanisms similar to those found in Adam, blending the robustness of curvature information with the flexibility of adaptive learning rates. Meanwhile, a 4-bit variant of Shampoo (Wang, Zhou, Li, & Huang, 2024) addresses memory limitations by quantizing both the parameters and the preconditioners. This quantization reduces the memory footprint dramatically, making it feasible to apply second-order updates to very large models where storage and computational resources are at a premium.

The key intuition behind Shampoo and its extensions is to capture richer, multidimensional curvature information than what is available through simple diagonal approximations. By preconditioning the gradient with matrices that reflect the full second-moment structure along different dimensions, these methods enable more nuanced adjustments: taking larger steps in flatter directions and smaller, more cautious steps in regions of high curvature. Although this enhanced curvature awareness can lead to faster convergence and better adaptation to complex loss landscapes, it comes at the cost of increased computational and memory requirements. Extensions like SOAP and the 4-bit variant are motivated by the need to mitigate these challenges, offering scalable implementations that make the advantages of second-order information accessible even in massive deep learning models.

Summary and Outlook. Overall, second-order methods aim to utilize curvature information—via the Hessian, Fisher Information Matrix, or generalized approximations—to guide more informed and often more stable parameter updates. Approaches like K-FAC and Shampoo factorize large matrices to preserve crucial curvature cues without incurring prohibitive computational costs. Meanwhile, diagonal or block-diagonal Hessian approximations enable partial second-order adaptation that is more practical for massive models. Although these techniques have evolved significantly, challenges remain in scaling them to the largest networks and reconciling second-order insights with the stochasticity inherent in modern deep learning. Ongoing research continues to push the boundaries, as evidenced by the growing variety of second-order and hybrid optimizers, each offering a unique balance between computational feasibility and the richness of curvature information.

2.5 Emerging Directions

The field of deep network optimization continues to evolve rapidly, with novel methods emerging to tackle both longstanding theoretical challenges and pressing practical constraints. Recent work has pushed the boundaries in several key areas, opening up exciting new avenues for research and application.

Integration of Curvature and Adaptive Strategies. A prominent trend in recent research is the seamless blending of adaptive first-order techniques with more refined curvature approximations. New hybrid methods, such as INNA-Prop and FOSI, aim to combine the low computational overhead and robustness of first-order updates with the enhanced directionality offered by partial second-order information. These methods typically incorporate curvature-aware preconditioners into adaptive frameworks, effectively adjusting the step sizes based not only on gradient magnitudes but also on local curvature estimates. The overarching goal is to achieve a more balanced optimization process that is both efficient and sensitive to the underlying loss landscape. While early results are promising, challenges remain in accurately estimating curvature in noisy, high-dimensional settings and in determining the optimal balance between adaptive scaling and curvature corrections.

Memory-Efficient Architectures. As NN models continue to grow in size, memory efficiency becomes an increasingly critical factor. Recent innovations such as 4-bit Shampoo and GaLore exemplify this trend by employing quantization and low-dimensional projection techniques to compress both parameters and the auxiliary statistics required for second-order updates. These methods make it feasible to deploy complex second-order optimizers even in resource-constrained environments or when dealing with models that contain hundreds of millions or even billions of parameters. By reducing the memory footprint without severely compromising the quality of curvature estimates, such techniques promise to bridge the gap between theoretical advances and practical deployment on modern hardware accelerators.

Robustness and Generalization in the Context of Large Models. Generalization remains a central challenge as models become larger and more complex. Insights from studies on the loss landscape, including works like yeh Chiang et al. (2023), Fort and Scherlis (2019), and Vysogorets et al. (2024), have underscored the importance of flat minima in achieving robust generalization. Building on these insights, methods such as SAM and its derivatives (e.g., AdaSAM) are being refined to not only accelerate convergence but also to guide the optimizer towards regions of the parameter space that generalize well. Recent research has further explored how phase transitions in deep learning dynamics (as discussed in Kalra and Barkeshli (2023)) might be harnessed to design optimizers that are inherently more robust to overfitting while still maintaining computational efficiency.

Optimization for Fine-Tuning Large Language Models. The rapid rise of Large Language Models (LLM) has introduced unique optimization challenges, particularly in the fine-tuning stage. New methods, such as HiZOO, are designed to leverage Hessian information to capture fine-grained local curvature properties. This is crucial for stable fine-tuning, where the risk of drastic parameter changes can lead to overfitting or catastrophic forgetting. These approaches often integrate second-order insights into scalable frameworks that can handle the massive parameter counts typical of transformer architectures. Future research in this area is expected to further refine these techniques, ensuring that fine-tuning processes are both robust and efficient.

Theoretical Advances and Practical Algorithms. There is a growing effort to reconcile theoretical optimization guarantees with the practical requirements of deep learning. Recent studies have investigated the implicit regularization effects of SGD Frankle and Carbin (2019) and the role of curvature in shaping generalization properties Xie et al. (2023). These investigations are guiding the development of new algorithms that seek to balance exploration—via stochasticity and adaptive updates—with exploitation—through careful curvature estimation. By grounding practical algorithms in strong theoretical foundations, researchers aim to develop optimizers that not only converge quickly but also offer provable guarantees on generalization performance.

Summary and Outlook. In summary, the emerging directions in deep network optimization are multifaceted and dynamic, addressing issues of efficiency, robustness, scalability, and theoretical rigor. By integrating advanced curvature approximations with adaptive strategies, embracing memory-efficient architectures, and targeting both robustness and fine-tuning challenges in large models, the field is moving toward a new generation of optimizers. These innovations promise to make deep learning training more efficient and reliable, even as model sizes and dataset complexities continue to increase.

2.6 Concluding Remarks

In this chapter, we have provided a comprehensive survey of the literature on deep network optimization, weaving together theoretical insights, practical algorithms, and emerging research directions. Our review has spanned the full spectrum of techniques and challenges that define the SOTA in deep learning optimization. We began by establishing the foundational concepts and notation in Section 2.1, setting the stage for a rigorous exploration of deep learning dynamics. In Section 2.2, we examined the evolution of training processes, including phase transitions and the critical role of loss landscape geometry. These elements, along with implicit regularization mechanisms, have been shown to play pivotal roles in guiding training toward solutions with strong generalization properties.

The discussion then transitioned to the practical challenges of optimizing deep networks in Section 2.3. Here, we highlighted the inherent difficulties posed by non-convex optimization, the

trade-offs between overfitting and generalization, and the significant computational burdens that arise as models grow in scale and complexity. Our in-depth review of optimization methods in Section 2.4 revealed a rich and diverse landscape. We explored:

- **Zeroth-Order Methods** (2.4.1), which rely solely on function evaluations and finite-difference approximations to operate in gradient-free settings.
- **First-Order Methods** (2.4.2), such as SGD and adaptive variants like Adam and RAdam, which leverage gradient information to form the backbone of deep network training.
- **Second-Order Methods** (2.4.3), including Hessian- and Fisher-based approaches like K-FAC and Shampoo, which incorporate curvature information to achieve more informed and potentially faster convergence, albeit at a higher computational cost.

Furthermore, Section 2.5 outlined emerging directions that promise to reshape the field. Researchers are increasingly integrating adaptive strategies with refined curvature approximations, developing memory-efficient architectures tailored to the demands of massive models, and designing specialized algorithms for fine-tuning large-scale transformers and language models. These trends are driven not only by the quest for enhanced performance but also by the need for robust, scalable, and theoretically grounded optimization techniques.

Overall, the insights gathered throughout this chapter not only highlight the remarkable progress achieved in deep network optimization but also underscore the exciting opportunities that lie ahead. By bridging theoretical foundations with practical implementations, the methods reviewed here pave the way for next-generation optimizers that can handle the complexities of modern NNs. As research continues to push the boundaries, future work will likely focus on further integrating adaptive and curvature-aware strategies, enhancing computational efficiency, and developing algorithms that deliver both strong convergence guarantees and superior generalization. In sum, the landscape of deep network optimization is rich with innovation and promise. The diverse approaches surveyed in this chapter provide a robust framework for understanding the multifaceted challenges of modern deep learning, and they set the stage for continued advancements in the field.

Bridging the Literature Review with AdaFisher

The comprehensive survey presented in this chapter has illuminated the rich tapestry of approaches in deep network optimization—from the efficiency of first-order adaptive methods to the nuanced curvature approximations offered by second-order techniques. Methods such as Adam and RAdam have set the benchmark for adaptive scaling, while curvature-based strategies like K-FAC and Shampoo have demonstrated the power of leveraging the FIM and its approximations to guide optimization. These approaches underscore a central theme: integrating adaptive behavior with curvature-aware preconditioning can significantly enhance both convergence speed and generalization performance.

AdaFisher (GOMES, Zhang, Belilovsky, Wolf, & Hosseini, 2025) optimization method builds directly on these insights. By fusing adaptive strategies with Fisher-based curvature information, AdaFisher seeks to overcome several limitations identified in the literature. Specifically, it addresses the challenges of computational scalability and stability by dynamically constructing efficient, layer-wise preconditioners derived from the FIM. In doing so, AdaFisher not only inherits the robustness of adaptive optimizers but also benefits from the precise curvature estimates provided by second-order approximations.

In the subsequent Chapters, we detail the theoretical underpinnings of AdaFisher, outline its algorithmic framework, and present empirical evaluations that demonstrate its superior performance on large-scale deep learning tasks. This work represents a natural evolution of the ideas surveyed in this chapter, aiming to bridge the gap between first-order adaptability and second-order precision in a single, scalable optimization method.

Chapter 3

Efficient Approximation of the FIM

The FIM is fundamental to statistical estimation and natural gradient descent, providing critical curvature information about the loss landscape in DL. However, directly computing the FIM is infeasible for modern deep networks due to its high dimensionality and complexity. This challenge has spurred efficient approximations—most notably, the K-FAC method—which decomposes the FIM into tractable Kronecker product factors. In this chapter, we investigate the structural properties of the FIM and its approximations. We first review the supervised learning framework and the role of the FIM in natural gradient descent, then examine K-FAC and its extensions across various layer types (fully connected, convolutional, and normalization layers). Empirical analyses reveal that the Kronecker factors exhibit a strong diagonal concentration even under noise, which motivates our novel diagonal approximation of the FIM that significantly reduces computational overhead while preserving essential curvature information.

The chapter is organized as follows. Section 3.1 provides the necessary theoretical background. Section 3.2 presents an in-depth study of the diagonal concentration of Kronecker factors, and Section 3.3 outlines our efficient computation method for the FIM. Together, these contributions form a scalable, curvature-aware optimization framework for DL.

3.1 Background

In this section, we briefly review the core concepts and notation that form the foundation of our methodology for efficient FIM approximation. For a more comprehensive treatment, we strongly recommend first reading Section 2.1 and Section 2.4.3, where these ideas are explored in depth. The discussion below assumes familiarity with the detailed exposition provided earlier. We consider a supervised learning framework with a dataset $\mathbf{D} \coloneqq \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$, where $\mathbf{x}_n \in \mathbb{R}^d$ and $\mathbf{y}_n \in \mathbb{R}^C$. Let $f_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}^C$ be an L-layer NN parameterized by $\boldsymbol{\theta} = \text{vec}\left(\{\theta_i\}_{i=1}^L\right)$, where $\theta_i = \text{concat}(W_i, \mathbf{b}_i) \in \mathbb{R}^{P_i}$, where $P_i = P_i^{out} \times (P_i^{in} + 1)$. The network computes its output through successive layers via

$$\mathbf{a}_i = \theta_i \bar{\mathbf{h}}_{i-1}, \quad \mathbf{h}_i = \phi_i(\mathbf{a}_i), \quad \text{with } \mathbf{h}_0 = \mathbf{x}_n \text{ and } \bar{\mathbf{h}}_{i-1} = [\mathbf{h}_{i-1}^\top, \mathbf{1}]^\top.$$

For a given input-target pair $(\mathbf{x}, \mathbf{y})$, the loss is defined as the negative log-likelihood, $\mathcal{L}(\mathbf{y}, f_{\boldsymbol{\theta}}(\mathbf{x})) \coloneqq -\log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$, and the gradients are computed via backpropagation. We denote the pre-activation derivatives by $\mathbf{s}_i = \nabla_{\mathbf{a}_i} \mathcal{L}$, and the gradient with respect to the weights of layer i is given by $\nabla_{\theta_i} \mathcal{L} = \mathbf{s}_i \bar{\mathbf{h}}_{i-1}^\top$. To capture curvature information, we use the FIM as an approximation to the Hessian

$$F = \mathbb{E}\left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})^\top\right] = \mathbb{E}[\mathbf{g}\mathbf{g}^\top], \quad \mathbf{g} = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$$

The K-FAC approach further simplifies the FIM by exploiting the layer-wise structure of deep networks. Viewing F as a block matrix with blocks$F_{i,j} = \mathbb{E}[\text{vec}(g_i)\,\text{vec}(g_j)^\top]$, and using the Kronecker-vectorization identity $\text{vec}(\mathbf{u}\mathbf{v}^\top) = \mathbf{v} \otimes \mathbf{u}$, we express $\text{vec}(g_i) = \bar{h}_{i-1} \otimes s_i$. Assuming that activations and pre-activation derivatives are mutually independent, each block is approximated as

$$\hat{F}_{i,j} \approx \mathbb{E}[\bar{\mathbf{h}}_{i-1} \bar{\mathbf{h}}_{j-1}^\top] \otimes \mathbb{E}[\mathbf{s}_i \mathbf{s}_j^\top] = \mathcal{H}_{i-1,j-1} \otimes \mathcal{S}_{i,j}.$$

By further assuming that weight derivatives across distinct layers are uncorrelated, the FIM is approximated by its block-diagonal form: $\hat{F} = \text{diag}\left(\hat{F}_1, \ldots, \hat{F}_L\right), \quad \text{with} \quad \hat{F}_i \approx \mathcal{H}_{i-1,i-1} \otimes \mathcal{S}_{i,i}$.

3.2 Diagonal Concentration of KFs

Inspired by the Gershgorin circle theorem (Horn & Johnson, 2012), we empirically study the diagonal concentration of Kronecker Product Factors (KFs) within DNNs. Our focus centers on the eigenvalue distribution and perturbation behavior of weight matrices, with a particular emphasis on the 37th layer of ResNet-18 (K. He et al., 2016) after 50 epochs of training on the CIFAR-10 dataset (Krizhevsky, Hinton, et al., 2009). Figure 3.1 illustrates the eigenvalue spectrum (red crosses) alongside Gershgorin discs (black circles), revealing that the eigenvalues cluster near the diagonal elements of the matrix. This observation underscores a pronounced diagonal dominance, which the Gershgorin circle theorem formalizes by asserting that every eigenvalue λ of a complex square matrix $\mathcal{A}$ resides within at least one Gershgorin disc $D(a_{ii}, R_i)$, where $R_i = \sum_{j \neq i} |a_{ij}|$ quantifies the row-based off-diagonal magnitude. To investigate the stability of this diagonal dominance under stochastic disturbances, we inject Gaussian noise $\mathcal{N}(0, \sigma^2)$ with $\sigma = 10^{-3}$ into the off-diagonal elements. The perturbed matrix $\hat{\mathcal{M}}$ is then given by

$$\hat{\mathcal{M}} = \mathcal{A} + \mathcal{E}, \quad \text{where } \mathcal{E} = [e_{ij}],\ e_{ij} \sim \mathcal{N}(0, \sigma^2) \text{ for } i \neq j.$$

Our perturbation experiments show that the principal eigenvalues satisfying the Kaiser criterion are minimally affected (Braeken & Van Assen, 2017), implying that the bulk of the matrix's energy is concentrated along the diagonal. Both the Gershgorin disc visualization and the eigenvalue perturbation analysis confirm a high degree of **diagonal dominance**.

Extending Gershgorin Analysis to Linear Layers. This diagonal concentration is not confined to convolutional layers alone. We observe an analogous pattern in the 41st (linear) layer of ResNet-18, as shown in Figure 3.2. Once again, the eigenvalues (red crosses) lie predominantly within the Gershgorin discs, centered along the diagonal (black circles), underscoring the pervasiveness of this diagonal dominance across diverse layer types.

Spectral Insights via FFT. Moving beyond eigenvalue-based diagnostics, we also perform a frequency-domain examination using the Fast Fourier Transform (FFT) to study how noise injection

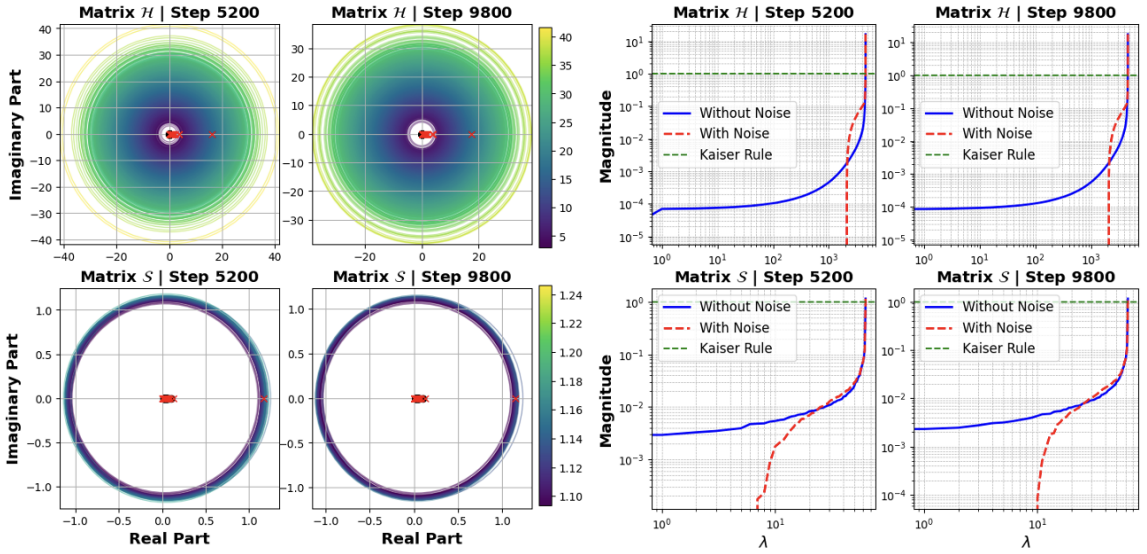

Figure 3.1: Gershgorin discs and eigenvalue perturbations for the 37th convolutional layer of ResNet-18 at steps 5200 (middle of training) and 9800 (end of training). Left: Gershgorin discs in the complex plane; Right: Eigenvalue spectrum with and without Gaussian noise added to off-diagonal entries.

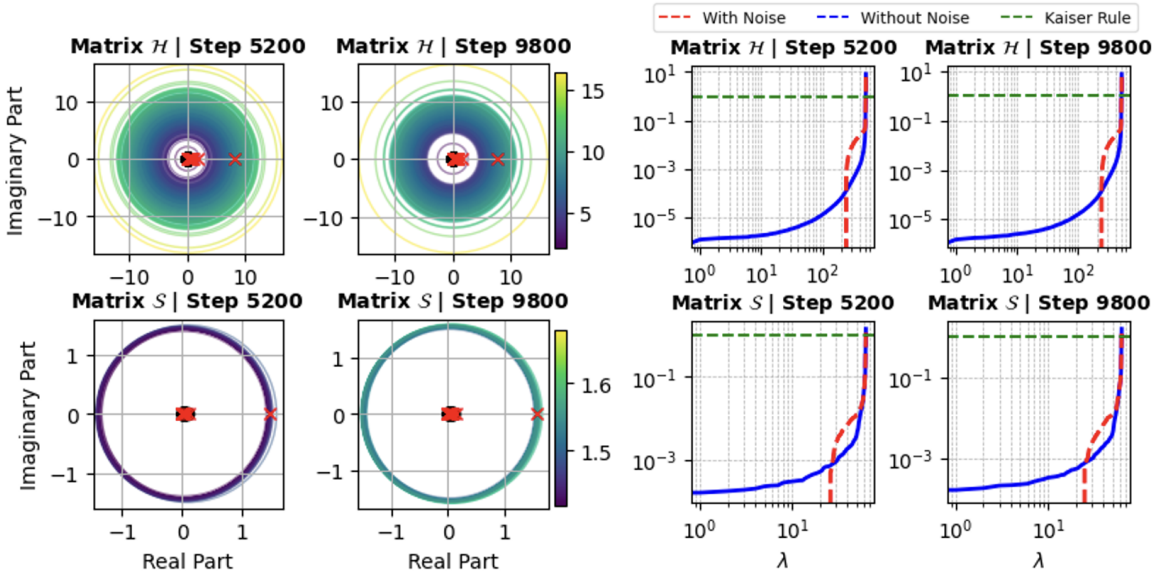

Figure 3.2: Gershgorin discs and eigenvalue perturbation analysis for matrices $\mathcal{H}$ and $\mathcal{S}$ at training steps 5200 and 9800 in the linear (41st) layer of ResNet-18. The left panel displays Gershgorin discs in the complex plane, while the right panel depicts the eigenvalue spectra with and without Gaussian noise.

impacts the KFs $\mathcal{H}$ and $\mathcal{S}$. Let $A \in \mathbb{C}^{m \times n}$ be a matrix, whose FFT is defined by

$$\mathcal{F}(A)_{kl} = \sum_{p=0}^{m-1} \sum_{q=0}^{n-1} A_{pq} \cdot e^{-2\pi i \left(\frac{pk}{m} + \frac{ql}{n}\right)}.$$

Figure 3.3 showcases the FFT magnitude plots for both noise-free and noisy conditions at training steps 5200 (middle of training) and 9800 (end of training). Even under Gaussian perturbations, the primary diagonal structure of the KFs remains conspicuous in the frequency domain, signifying that the principal information is predominantly concentrated along the diagonal. In addition, we

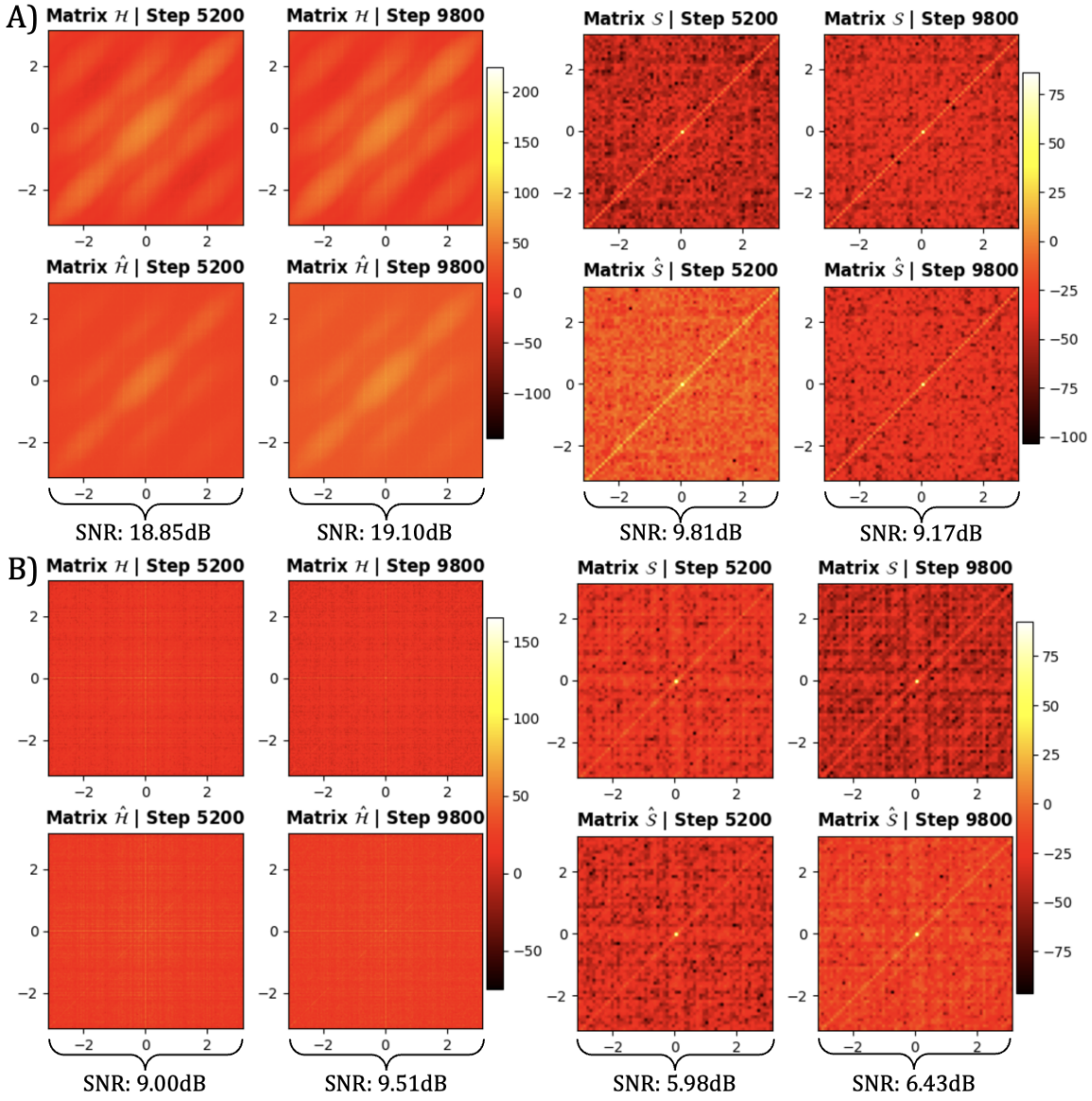

Figure 3.3: FFT-based spectral analysis of KFs $\mathcal{H}$ and $\mathcal{S}$ for convolutional (37th layer) and linear (41st layer) segments of a ResNet-18 at iterations 5200 and 9800. (A) Noise-free (top) vs. noisy (bottom) FFT results for $\mathcal{H}$ from the convolutional layer; (B) analogous results for $\mathcal{S}$ in the linear layer.

compute the Signal-to-Noise Ratio (SNR) to quantify the effect of noise on off-diagonal entries.

For a matrix $\mathcal{M}$ and its perturbed version $\hat{\mathcal{M}}$, the SNR is defined by

$$\text{SNR} = 10 \cdot \log_{10} \left(\frac{\sum_{i=1}^{N} |\mathcal{M}_{ii}|^2}{\sum_{j>i}^{N} |\hat{\mathcal{M}}_{ij}|^2} \right).$$

We observe that the SNR remains sufficiently high across training steps, implying minimal eigenvalue displacement and reinforcing our diagonal-dominance hypothesis.

Matrix Visualization and Diagonal Energy. Figure 3.4 provides direct visualizations of the KFs $\mathcal{H}$ and $\mathcal{S}$ at steps 5200 and 9800 for both the convolutional (37th) and linear (41st) layers. Each subplot reveals a pronounced diagonal band, further highlighting the structural consistency of these factors over the course of training. The resilience of this diagonal concentration against off-diagonal noise underscores a form of robust feature extraction, where the primary informational energy resides along the main diagonal.

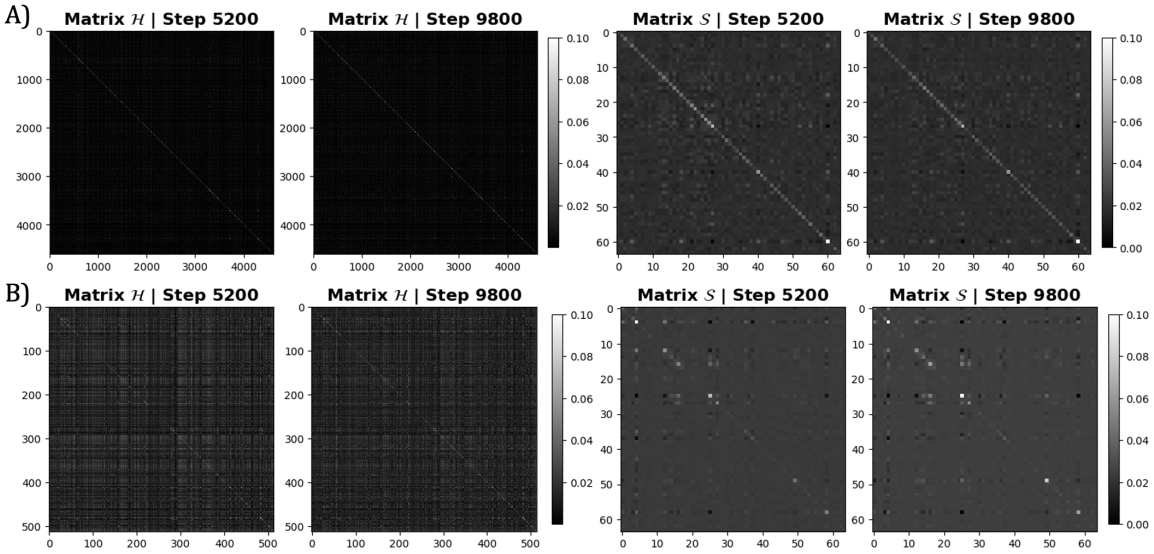

Figure 3.4: Visualization of KFs $\mathcal{H}$ and $\mathcal{S}$ in convolutional (A) and linear (B) layers of ResNet-18 at different iteration steps (5200 and 9800). The first two plots in (A) depict factor $\mathcal{H}$ at the two training stages, while the next two plots illustrate factor $\mathcal{S}$. Analogously, (B) shows the evolution of $\mathcal{H}$ and $\mathcal{S}$ in the linear layer. The diagonal salience persists throughout training, attesting to a stable, diagonal-centric structure.

Conclusion. Both Gershgorin disc analysis and perturbation experiments converge on the finding that the KFs in DNNs, particularly in ResNet-18 layers, exhibit strong diagonal dominance. Noise

introduced into off-diagonal elements has a minimal impact on the eigenvalues—especially those surpassing the Kaiser criterion—indicating that essential information is heavily concentrated along the diagonal. This property remains stable over successive training iterations, further corroborating the structural resilience of these matrices. Our Fourier-domain investigation reinforces these observations, revealing that even in the frequency spectrum, the pivotal data of the KFs is localized near the diagonal, thereby reinforcing the robustness and tenacity of Kronecker structures under noisy conditions.

3.3 Efficient Computation of the FIM

In the realm of optimization, NGD offers a geometrically nuanced adaptation of the classical steepest descent approach (in Euclidean space), transitioning the focus from parameter space to the model's distribution space underpinned by the adoption of a *Riemannian metric*, Amari and Nagaoka (2000). Using Eq. (16), the formulation of the *preconditioned* gradient $\bar{g}^{(t)}$ given the NGD method is articulated as

$$\bar{\mathbf{g}}^{(t)} = (F^{(t)})^{-1}\mathbf{g}^{(t)}, \tag{18}$$

One of the distinguishing features of NGD within this framework is its re-parametrization invariance, a direct consequence of leveraging the model's distribution properties rather than its parameters. Nevertheless, the direct FIM computation is highly demanding, and we solve this by adopting the diagonal approximation of the KFs as supported by our analyses from Section 3.2. In addition, a critical component of modern DNNs is known to be the normalization of the layers by introducing scale and shift parameters (e.g. batch-normalization (Ioffe, 2015), layer-normalization (Lei Ba, Kiros, & Hinton, 2016)). This is to adjust the network's dynamics (e.g., reducing covariance shift) in a non-trivial way L. Huang et al. (2023) where the lack of FIM approximation on such normalization layers can lead to suboptimal preconditioning. Therefore, we introduce a method for calculating the KFs for normalization layers detailed in Proposition 3.3.1.

Proposition 3.3.1 (EFIM for normalization layer)**.** *Let $(\boldsymbol{\nu}_i, \boldsymbol{\beta}_i) \in \mathbb{R}^{C_i}$ be the scale and shift parameters of a normalization layer i. The empirical KFs for the FIM approximation are*

$$\mathcal{H}_{i-1}\Big|_{\nu_i} = \frac{1}{|\mathcal{T}_i|}\sum_{x\in\mathcal{T}_i}\mathbf{h}_{i-1,x}\mathbf{h}_{i-1,x}^\top,\ \mathcal{H}_{i-1}\Big|_{\beta_i} = \mathbf{1}\mathbf{1}^\top, \quad \mathcal{S}_i = \frac{1}{|\mathcal{T}_i|}\sum_{x\in\mathcal{T}_i}\mathbf{s}_{i,x}\mathbf{s}_{i,x}^\top,$$

where $\mathbf{h}_{i-1}, \mathbf{s}_i \in \mathbb{R}^{C_i\times|\mathcal{T}_i|}$ represent the pre-normalized activations and gradients, respectively. Here, $\mathcal{T}_i$ is the set of dimensions over which normalization statistics are computed, and C_i is the channels/features size.

The proof of this proposition is given in Appendix A. The KFs for other type of layers are computed following methodologies similar to those described in Grosse and Martens (2016); Martens and Grosse (2015b). This section revisits the key equations used for this computation. For a given layer i in a NN, the empirical KFs are computed as follows:

- For **fully connected layers**, the KFs are:

$$\mathcal{H}_{D_{i-1}} = \text{diag}(\bar{\mathbf{h}}_{i-1}\bar{\mathbf{h}}_{i-1}^\top), \quad \mathcal{S}_{D_i} = \text{diag}(\mathbf{s}_i\mathbf{s}_i^\top);$$

- For **convolutional layers**, the computation accounts for the spatial positions within the layer, denoted as $\mathcal{T}$:

$$\mathcal{H}_{D_{i-1}} = \text{diag}\left(\frac{[\![\bar{\mathbf{h}}_{i-1}]\!][\![\bar{\mathbf{h}}_{i-1}]\!]^\top}{|\mathcal{T}|}\right), \quad \mathcal{S}_{D_i} = \text{diag}\left(\frac{\mathbf{s}_i\mathbf{s}_i^\top}{|\mathcal{T}|}\right);$$

  The algorithm employs the expansion operation denoted by $[\![\cdot]\!]$ (Grosse & Martens, 2016). *This operation essentially takes the patches surrounding spatial locations, stretches them into vectors, and compiles these vectors into a matrix.*

- For **Normalization layers** (BatchNorm & LayerNorm) refer to Proposition. 3.3.1

- For all **other type of layers** the KFs are:

$$\mathcal{H}_{D_{i-1}} = \mathbf{I}_{P_{i-1}^{out}+1}, \quad \mathcal{S}_{D_i} = \mathbf{I}_{P_i^{out}};$$

Note that in the context of online and stochastic optimization, the KFs for a given layer i can be estimated using an Exponentially Moving Average (EMA) scheme across batches defined by

$$\mathcal{H}_{i-1}^{(t)} = \gamma \mathcal{H}_{i-1}^{(t)} + (1-\gamma)\mathcal{H}_{i-1}^{(t-1)}, \; \mathcal{S}_{i}^{(t)} = \gamma \mathcal{S}_{i}^{(t)} + (1-\gamma)\mathcal{S}_{i}^{(t-1)}, \tag{19}$$

where $0 < \gamma \leq 1$ is the exponential decay factor at step time t, $\mathcal{H}_{i-1}^{(t)}$ and $\mathcal{S}_{i}^{(t)}$ in the right-hand side are new KFs calculated during each forward and backward pass computation. This EMA scheme is commonly used in methods involving diagonal or block-diagonal approximations to the curvature matrix (e.g. LeCun, Bottou, Orr, and Müller (2012); Park, Amari, and Fukumizu (2000); Schaul, Zhang, and LeCun (2013)). Such schemes have the desirable property that they allow the curvature estimation to depend on much more data than what can be reasonably processed in a single mini-batch.

Our study from Section 3.2 suggests that the FIM's critical information predominantly resides along its diagonal. Building upon this, we propose a novel approximation for the FIM, described in Proposition 3.3.2, that conceptualizes the KFs as diagonal matrices denoted as $\tilde{F}_{D_i}$ for layer i.

Proposition 3.3.2 (Efficient EFIM)**.** *Assume that $\mathcal{H}_{i-1}$ and $\mathcal{S}_i$ can be closely approximated by diagonal matrices, denoted by $\mathcal{H}_{D_{i-1}}$ and $\mathcal{S}_{D_i}$ respectively at layer i, such that $\mathcal{H}_{D_{i-1}} = Diag(\mathcal{H}_{i-1})$, $\mathcal{S}_{D_i} = Diag(\mathcal{S}_i)$ where Diag denote the diagonal of a matrix. Accordingly, the Empirical FIM is defined by*

$$\tilde{F}_{D_i} \triangleq \mathcal{H}'_{D_{i-1}} \otimes \mathcal{S}'_{D_i} + \lambda \mathbf{I}, \tag{20}$$

where $\mathcal{H}'_{D_{i-1}}$ and $\mathcal{S}'_{D_i}$ denote the Min-Max normalization of $\mathcal{H}_{D_{i-1}}$ and $\mathcal{S}_{D_i}$ (Patro & Sahu, 2015) and λ is a regularization parameter.

The proof of this proposition is given in Appendix A. This approximation strikes a balance between computational time and space complexity and the accuracy of performance, as discussed in Chapter 5. We set the regularization parameter $\lambda = 0.001$, which acts as a damping factor following the Tikhonov regularization principle Martens and Grosse (2015b), enhancing computational stability and conditioning of the FIM. The closed-form solution for the preconditioned gradient $\bar{\mathbf{g}}^{(t)}$

is derived from the diagonal approximation of the FIM, given by

$$\tilde{\mathbf{g}}^{(t)} = (\tilde{F}_D^{(t)})^{-1}\mathbf{g}^{(t)}, \tag{21}$$

for time step t. Notice this is similar to the NGD update defined in Eq. (18), except that we approximate the FIM with a novel efficient method. This represents the AdaFisher, which will be presented in Chapter 4, augmented gradient and incorporates local loss *curvature information*. It focuses on the diagonal elements to reduce computational overhead while maintaining a reasonable FIM approximation. This simplification enhances the efficiency of the optimization process, which is crucial for training DNNs where computational resources are limited.

3.4 Concluding Remarks

In this chapter, we explored the empirical FIM within DNNs and investigated efficient approximations that facilitate second-order optimization. Our analysis began with a review of the negative log-likelihood loss in supervised learning and demonstrated how the FIM, closely related to the Hessian of the log-likelihood, captures essential curvature information for parameter updates. We then focused on K-FAC methods, which decompose the FIM into KFs for tractability in large-scale networks. By dissecting the diagonal concentration of these factors using Gershgorin circle theorem, we empirically observed a pronounced diagonal dominance across both convolutional and linear layers of a ResNet-18, even under noise perturbations. The spectral and Fourier-domain analyses further corroborated this diagonal-centric structure, indicating that injecting Gaussian noise into the off-diagonal entries yields only marginal shifts in the eigenvalues and minimal deterioration in signal-to-noise ratios. Such observations underscore the stability and robustness of the KFs, suggesting that the critical information for optimization resides predominantly along their diagonals. Building on these insights, we proposed a diagonal approximation of the KFs in Proposition 3.3.2, offering a simplified yet effective route to approximate the FIM in DL. This diagonalization strategy significantly reduces computational overhead while preserving key curvature cues, aligning with the practical demands of large-scale optimization.

Additionally, we extended K-FAC to normalization layers by establishing empirical factors for

scale and shift parameters, thereby ensuring comprehensive coverage of modern deep network architectures. Our results revealed that even these normalization-centric matrices exhibit strong diagonal behavior, reinforcing the broader applicability of our diagonal approximation across diverse layer types. The final step involved deriving a closed-form preconditioned gradient under the diagonal FIM approximation, leading to an *AdaFisher* update, which will be presented in Chapter 4, that incorporates local curvature without incurring the full cost of second-order methods. Overall, the work in this chapter highlights how diagonal-dominant structures in the FIM can be exploited to achieve computationally efficient second-order optimization. This diagonalization perspective not only aligns with empirical observations of robust eigenvalue spectra but also provides a principled foundation for designing fast, curvature-aware training algorithms. Subsequent chapters will build upon these insights, evaluating the performance and convergence properties of diagonal-based Fisher approximations in both theoretical and experimental settings.

Chapter 4

AdaFisher: An Adaptive Second order Optimization via Fisher Information

In this chapter, we introduce *AdaFisher*, a novel adaptive second-order optimization method that leverages the FIM to guide DNN training. Building on the adaptive framework established by Adam, AdaFisher employs a refined diagonal block-Kronecker approximation of the FIM, leading to more accurate curvature estimation. This enhancement not only accelerates convergence but also steers the optimizer toward flatter minima, thereby improving model generalization.

We begin by outlining how the FIM is integrated into the adaptive optimization framework and discuss the key differences between AdaFisher and conventional optimizers. Next, we present the distributed implementation of AdaFisher, demonstrating its scalability and efficiency in multi-GPU environments. Finally, we analyze the convergence properties of AdaFisher in both convex and non-convex settings, supported by theoretical insights and empirical results.

Overall, this chapter provides a comprehensive overview of the theoretical foundations and practical benefits of incorporating second-order information via the FIM into modern DL optimization.

Table 4.1: Summary of the first moment ($m^{(t)}$), second moment ($v^{(t)}$), regret bounds, and applicability used in various optimizers for updating model parameters $\theta^{(t+1)} = \theta^{(t)} - \alpha\, m^{(t)}/\sqrt{v^{(t)}}$. Here, β_1 and β_2 denote the hyperparameters for the first and second moments, respectively; $L^{(t)}$ and $R^{(t)}$ refer to the preconditioning matrices used in Shampoo V. Gupta et al. (2018); $g^{(t)} = \mathrm{vec}(G^{(t)})$; and T is the total number of steps. For AdaFactor, $r^{(t)}$ and $c^{(t)}$ denote the row and column accumulators that aggregate squared gradients to yield the factored second moment estimate. For Sophia, $\hat{h}^{(t)}$ is an approximation of the Hessian diagonal (with decay ρ and regularization ϵ). For SOAP, $Q^{(t)}$ denotes the eigenvector matrix of Shampoo's preconditioner, used to transform the gradients into its eigenbasis for Adam-like updates. Please refer to Chapter 2 for more details.

Optimizer	$m^{(t)}$	$v^{(t)}$	**Regret Bound**	**Applicability**	
				CNNs	Transf.
Adam	$\frac{(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}g_i}{1-\beta_1^t}$	$\left(\frac{(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}g_i g_i}{1-\beta_2^t}\right)^{1/2}$	$O(\log T\sqrt{T})$	✓	✓
AdaHessian	$\frac{(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}g_i}{1-\beta_1^t}$	$\left(\frac{(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}D_i^{(s)}D_i^{(s)}}{1-\beta_2^t}\right)^{1/2}$	$O(\log T\sqrt{T})$	✓	✓
K-FAC	$(\hat{F}^{(t)})^{-1}\mathbf{g}^{(t)}$	1	$O(\sqrt{T})$	✓	×
Shampoo	$(L^{(t)})^{\frac{-1}{4}}G^{(t)}(R^{(t)})^{\frac{-1}{4}}$	1	$O(\sqrt{T})$	✓	×
Sophia	$\frac{(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}g_i}{1-\beta_1^t}$	$\rho\,\hat{h}^{(t)}+\epsilon$	$O(\log T\sqrt{T})$	✓	✓
SOAP	$\frac{(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}Q^{(t)\top}g_i}{1-\beta_1^t}$	1	$O(\sqrt{T})$	✓	×
AdaFactor	$\frac{(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}g_i}{1-\beta_1^t}$	$\frac{r_i^{(t)}c_j^{(t)}}{\sum_k r_k^{(t)}}$	$O(\log T\sqrt{T})$	✓	✓
AdaFisher	$\frac{(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}g_i}{1-\beta_1^t}$	$\tilde{F}_D^{(t)}$	$O(\log T\sqrt{T})$	✓	✓

4.1 Integrating FIM into Adaptive Optimization Framework

Following the spirit of the adaptive optimization framework from Adam, which combines momentum and RMSProp principles (Sutskever, Martens, Dahl, & Hinton, 2013b), the parameters are updated via

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha\frac{\mathbf{m}^{(t)}}{\mathbf{v}^{(t)}}.$$

Here, α represents the learning rate, while $\mathbf{m}^{(t)}$ and $\mathbf{v}^{(t)}$ denote the first and second moment estimates, respectively, for time step t. Although Adam is widely used, its approximation of the second moment using simple diagonal elements of second-order statistics through squared gradients can mirror stability challenges (Kunstner, Hennig, & Balles, 2019). We overcome these challenges by utilizing a more refined diagonal block-Kronecker approximation of the FIM introduced in Section 3.3, a more precise approximation of the Hessian from Taylor series expansion viewpoint as

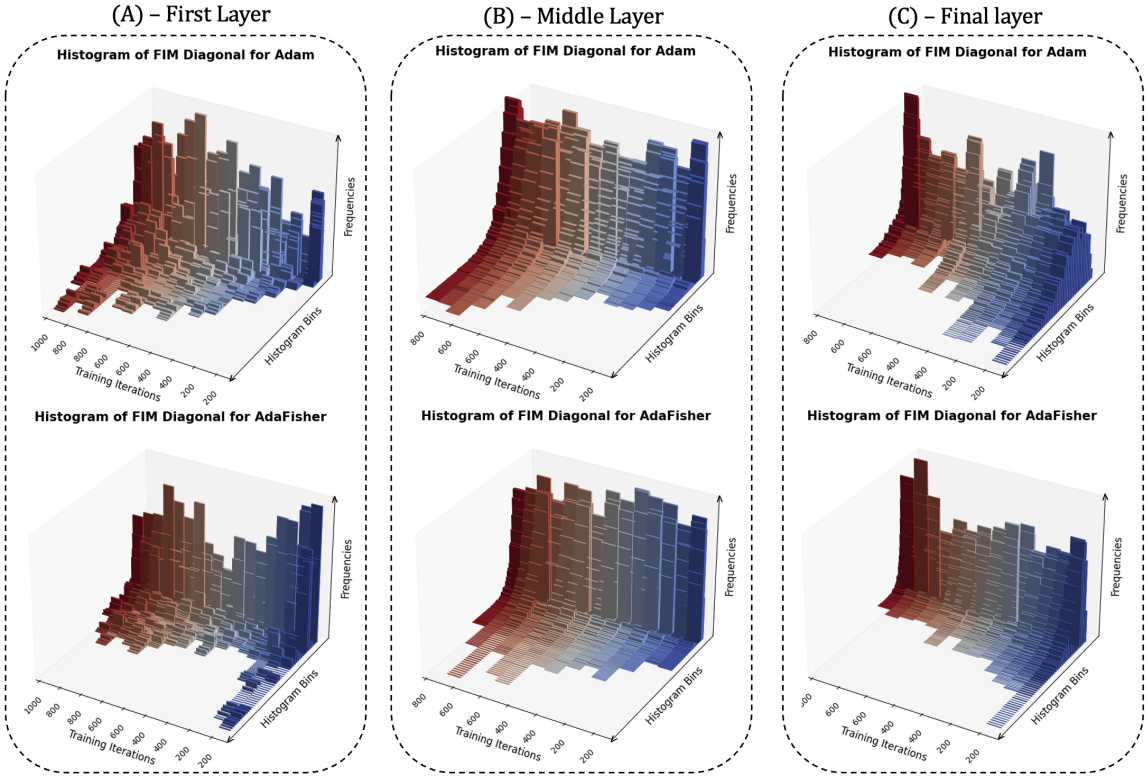

Figure 4.1: Comparison of FIM Diagonal Histograms during ResNet18 Training on CIFAR10 with Adam and AdaFisher over 1,000 training iterations. Panel (A) displays the FIM diagonal elements for the first convolutional layer; Panel (B) illustrates the FIM diagonal elements for the middle convolutional layer; Panel (C) shows the FIM diagonal elements for the last Linear layer.

discussed in Section 2.1.3. This structured alternative to Adam's diagonal approximation enables precise curvature estimation, mitigating stability issues and improving convergence in non-convex settings. AdaFisher distinguishes itself from Adam by providing a higher-fidelity approximation of the FIM, which enhances both optimization efficiency and model robustness throughout training. As shown in Figure 4.1, the FIM values computed by AdaFisher exhibit reduced variance and lower mean values during training—indications that the optimizer is converging toward flatter minima and inducing an implicit regularization effect. This behavior is especially pronounced during the final training stages, where AdaFisher outperforms its counterparts (see Chapter 5). The improvement stems from the FIM's dual role: in the early and mid-training phases, it approximates the Hessian (acting similarly to the Generalized Gauss-Newton Matrix (Eschenhagen, Immer, Turner, Schneider, & Hennig, 2024)), and near local minima, it aligns more precisely with the Hessian (Martens,

2020), thereby accelerating convergence. Our analysis of the diagonal block-Kronecker FIM distribution during ResNet18 training on CIFAR10 further corroborates these findings. Across various network layers—including the first (Panel A) and middle (Panel B) convolutional layers, as well as the final linear layer (Panel C)—AdaFisher consistently produces narrower FIM distributions with lower values compared to Adam, confirming its tendency to converge toward flatter local minima. This characteristic ultimately leads to more stable generalization when transitioning from training to testing distributions (Cha et al., 2021).

Additionally, AdaFisher eliminates the square root and the conventional EMA on the second moment. This is because the FIM naturally incorporates an EMA of its KFs (see Eq. (19)). Removing the square root is also consistent with the theoretical foundations of second-order methods, which rely on a second-order Taylor expansion approximation rather than a square-root transformation. A comparative summary of various moment estimates, $\mathbf{m}^{(t)}$ and $\mathbf{v}^{(t)}$, along with their regret bounds and applicability to different optimizers, is provided in Table 4.1. Notably, while SGD-based optimizers such as K-FAC or Shampoo exhibit a regret bound of $O(\sqrt{T})$, AdaFisher incurs a regret bound of $O(\log T \sqrt{T})$ as it belongs to the Adam's family, which is theoretically higher. However, this theoretical disadvantage is offset by AdaFisher's superior practical convergence. Specifically, AdaFisher leverages an approximate FIM to precondition its update steps, thereby reducing the variance of the gradients and adapting more accurately to the local curvature of the loss surface. In real-world scenarios—where the loss landscape is nonconvex and the worst-case assumptions underlying regret bounds are overly pessimistic—this refined curvature estimation facilitates more stable and efficient progress during training. Consequently, even though the worst-case bound for AdaFisher is higher, its enhanced exploitation of second-order information consistently results in faster and more robust convergence compared to its SGD-based counterparts.

Finally, inspired by AdamW—which improves Adam by directly integrating weight decay into the weight update to counteract suboptimal decay behavior and boost performance—we introduce AdaFisherW. This variant leverages the curvature information from AdaFisher within the AdamW framework to further enhance optimization. The implementation for both AdaFisher variants is delineated in the pseudo-code presented in Algorithm 1.

Algorithm 1 AdaFisher optimization algorithm. Good default settings for the tested machine learning problems are $\alpha = 0.001$ (learning rate), $\lambda = 0.001$ (Tikhonov damping parameter),$\gamma = 0.8$ (Exponentially decaying factor). [Default parameters are: $\beta = 0.9$ (Exponentially decaying factor of Adam), κ (weight decay) (Kingma and Ba (2015), Loshchilov and Hutter (2019))].

Require: Step size α; Exponential decay rate for KFs $\gamma \in [0, 1)$; Tikhonov damping parameter λ; Exponential decay rate for first moments β in $[0, 1)$; Initial parameters θ

Initialize 1st moment variable $m = 0$; FIM $\tilde{F}_{D_i} = \mathbf{I}$; time step $t = 0$

while stopping criterion not met **do**

Sample a minibatch of M examples from the training set $\{(x_n, y_n)\}_{n=1}^{M}$

Compute $\mathcal{H}_{D_{i-1}}, \mathcal{S}_{D_i}$ for $i \in \{1, \dots, L\}$ using Section 3.3 (notice that: $\mathcal{H}_{D_0} = x$)

Compute EMAs of $\mathcal{H}_{D_{i-1}}$ and $\mathcal{S}_{D_i}$ using Eq. (19)

Compute $\tilde{F}_{D_i}$ for $i \in \{1, \dots, L\}$ using Eq. (20)

$g^{(t)} \leftarrow \frac{1}{M} \sum_n \nabla_{\theta^{(t)}} \mathcal{L}(f(x_n; \theta^{(t)}), y_n)$ (Compute gradient)

$m^{(t+1)} \leftarrow \frac{\beta m^{(t)} + (1-\beta) h^{(t)}}{1-\beta^t}$ (Update and correct biased first moment)

Case AdaFisher: $\Delta\theta^{(t)} = -\alpha (\tilde{F}_D^{(t)})^{-1} m^{(t)}$

Case AdaFisherW: $\Delta\theta^{(t)} = -\alpha \left((\tilde{F}_D^{(t)})^{-1} m^{(t)} + \kappa \theta^{(t)} \right)$

$\theta^{(t+1)} \leftarrow \theta^{(t)} + \Delta\theta^{(t)}$ (Apply update)

$t \leftarrow t + 1$

end while

4.2 Distributed Implementation and Algorithm Overview

To complement the integration of the FIM into the adaptive optimization framework, we now detail the algorithmic implementation of AdaFisher and its compatibility with distributed multi-GPU environments. This section provides a concise overview of the core update rules, the mechanism for aggregating curvature information across GPUs, and a summary of the pseudo-code for AdaFisher. In distributed settings, aggregating these KFs across GPUs is crucial before updating model parameters. For a training setup with K GPUs and for any given layer i, the KFs are computed and aggregated as follows

$$\mathcal{H}_{Di-1}^{(\text{SUM})} = \frac{1}{K} \sum_{k=1}^{K} \mathcal{H}_{Di-1}^{(k)}, \quad \mathcal{S}_{D_i}^{(\text{SUM})} = \frac{1}{K} \sum_{k=1}^{K} \mathcal{S}_{D_i}^{(k)} \tag{22}$$

The theoretical justification for this aggregation lies in the linearity of expectation and the unbiasedness of the local KF estimates. Specifically, if each $\mathcal{H}_{D_{i-1}}^{(k)}$ and $\mathcal{S}_{D_i}^{(k)}$ are unbiased estimators

of their respective true factors $\mathcal{H}_{D_{i-1}}$ and $\mathcal{S}_{D_i}$ for $k = 1, \ldots, K$, then the averaged factors $\mathcal{H}_{D_{i-1}}^{(\text{SUM})}$ and $(\mathcal{S}_{D_i})^{(\text{SUM})}$ remain unbiased estimators of $\mathcal{H}_{D_{i-1}}$ and $\mathcal{S}_{D_i}$. Consequently, using Eq. (22), the aggregated EFIM for layer i can be calculated as

$$\tilde{F}_{D_i}^{\text{SUM}} = \mathcal{H}_{D_{i-1}}^{\prime(\text{SUM})} \otimes \mathcal{S}_{D_i}^{\prime(\text{SUM})} + \lambda \mathbf{I}$$

where λ is a regularization parameter that ensures numerical stability. This strategy allows each GPU to contribute effectively to the model update, thereby enhancing convergence and performance in large-scale distributed training. More details, results and ablation about the multi-GPU environments are detailed in Chapter 5.

4.3 Convergence Analysis

In this section, we provide a theoretical analysis of AdaFisher's convergence in both convex optimization and non-convex stochastic optimization. We first present a standard convergence behavior of Eq. (18) for a simple strongly convex and strictly smooth function $f(J)$.

Proposition 4.3.1 (Convergence in convex optimization)**.** *For the FIM defined in Eq. (20), the updating scheme* $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha(\tilde{F}_D^{(t)})^{-1}\nabla J(\boldsymbol{\theta}^{(t)})$ *converges. Moreover, if* ∇J *is Lipschitz, i.e.,* $||\nabla J(\boldsymbol{\theta}) - \nabla J(\boldsymbol{\theta}')||_2 \leq L||\boldsymbol{\theta} - \boldsymbol{\theta}'||$ *for any* $\boldsymbol{\theta}$ *and* $\boldsymbol{\theta}'$*, then for the* k*-step iteration with a fixed step size* $\alpha \leq 1/L$*, then*

$$J(\boldsymbol{\theta}^{(k)}) - J(\boldsymbol{\theta}^*) \leq \frac{||\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*||_2^2}{2\alpha k},$$

where $J(\boldsymbol{\theta}^*)$ *is the optimal value.*

For non-convex cases, we adopt the similar derivations of X. Chen, Liu, Sun, and Hong (2019b) since AdaFisher belongs to the family of generalized Adam-type methods.

Proposition 4.3.2 (Convergence in non-convex stochastic optimization)**.** *Under the assumptions:*
(i) f *is lower bounded and differentiable;* $||\nabla J(\boldsymbol{\theta}) - \nabla J(\boldsymbol{\theta}')||_2 \leq L||\boldsymbol{\theta} - \boldsymbol{\theta}'||_2$, $||\tilde{F}_D^{(t)}||_\infty < L$, $\forall t, \boldsymbol{\theta}, \boldsymbol{\theta}'$.
(ii) Both the true and stochastic gradient are bounded, i.e. $||\nabla J(\boldsymbol{\theta}^{(t)})||_2 \leq \lambda$ *and* $||g_t||_2 \leq \lambda$, $\forall t$

for some $\lambda > 0$.

(iii) Unbiased and independent noise in $\mathbf{g}^{(t)}$*, i.e.* $\mathbf{g}^{(t)} = \nabla J(\boldsymbol{\theta}^{(t)}) + \zeta^{(t)}$, $\mathbb{E}[\zeta^{(t)}] = 0$, *and* $\zeta^{(t)} \perp \zeta^{(t)}$, $\forall i \neq j$.

Assume $\eta^{(t)} = \frac{\eta}{\sqrt{t}}$, $\beta^{(t)} \leq \beta \leq 1$ *is non-increasing,* $\frac{\tilde{F}_D^{(t-1)}[j]}{\eta^{(t-1)}} \leq \frac{\tilde{F}_D^{(t)}[j]}{\eta^{(t)}}$, $\forall t \in [T], j \in [d]$, *we then have*

$$\min_{t\in[T]} \mathbb{E}[||\nabla J(\boldsymbol{\theta}^{(t)})||_2^2] \leq \frac{L}{\sqrt{T}}(C_1\eta^2\lambda^2(1+\log T) + C_2 d\eta + C_3 d\eta^2 + C_4)$$

where C_1, C_2, C_3 *are constants independent of* d *and* T, C_4 *is a constant independent of* T, *the expectation is taken w.r.t all the randomness corresponding to* $\{g^{(t)}\}$.

The proofs of propositions 4.3.1 and 4.3.2 are given in Appendix A. Proposition 4.3.2 implies the convergence rate for AdaFisher in the non-convex case is at $O(\log T/\sqrt{T})$, which is similar to Adam-type optimizers. While DNNs often include non-smooth components like ReLU and max pooling, which create non-differentiable points in the loss landscape, optimizers like AdaFisher handle these cases effectively, as shown by our results in Chapter 5. We further demonstrate AdaFisher's convergence on a simple model and dataset described in Appendix B. As illustrated in Figure 4.2, across various seeds, AdaFisher consistently outperforms the baseline optimizer by converging toward the optimal local minima in the loss landscape with respect to parameters W_1 and W_2. These results further validate the convergence analysis discussed above.

4.4 Concluding Remarks

In this chapter, we introduced *AdaFisher*, a novel adaptive second-order optimization method that leverages refined approximations of the FIM to enhance deep network training. By integrating a diagonal block-Kronecker approximation into an adaptive framework inspired by Adam, AdaFisher offers a higher-fidelity estimation of curvature compared to conventional methods, thus enabling more accurate and stable updates. Our approach not only accelerates convergence by effectively capturing local curvature information but also steers the optimization trajectory toward flatter minima, which is beneficial for generalization. We detailed the theoretical foundations underlying

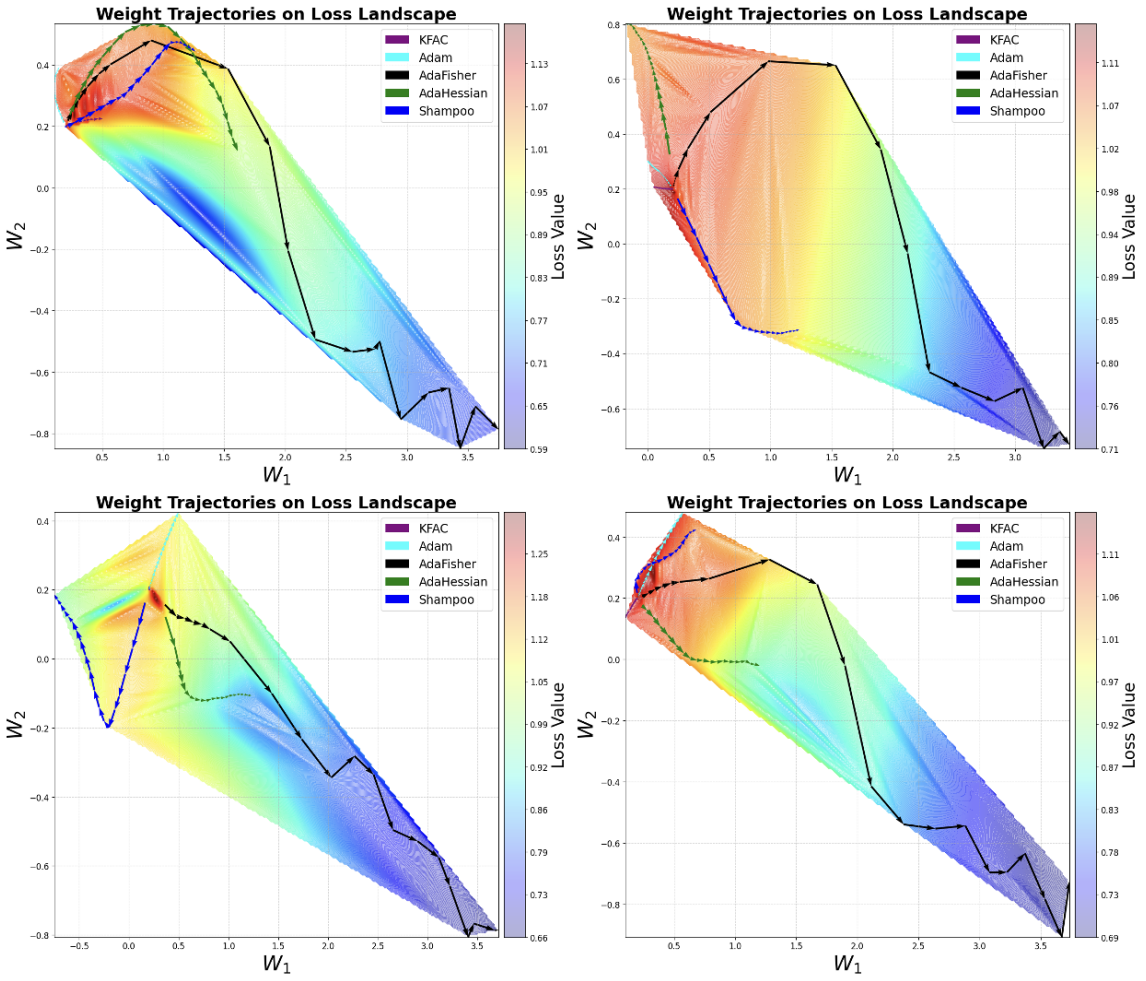

Figure 4.2: Weight trajectories within different loss landscapes (evaluated using four seeds) of a toy model for different optimizers.

AdaFisher, including its derivation from natural gradient descent principles and the approximation of the FIM using layer-wise Kronecker factorization. The distributed implementation further demonstrates that AdaFisher scales efficiently in multi-GPU environments, making it practical for large-scale deep learning applications. Our convergence analysis, covering both convex and non-convex settings, shows that AdaFisher attains convergence rates comparable to other SOTA Adam-type optimizers while benefiting from a more robust curvature estimation mechanism.

From a theoretical standpoint, AdaFisher distinguishes itself by achieving convergence rates comparable to Adam-type methods—specifically, an $O(\log T/\sqrt{T})$ rate in non-convex settings—while

incorporating more precise curvature information through a diagonal block-Kronecker approximation of the FIM. This refined curvature estimation not only improves the stability of updates by mitigating the effects of noisy gradients but also guides the optimization trajectory toward flatter minima, which are empirically linked to better generalization. In contrast, first-order methods like Adam rely solely on momentum and squared gradient accumulation, potentially compromising convergence in regions of high curvature. On the other hand, second-order methods such as K-FAC and Shampoo offer rapid convergence through full curvature information but often incur substantial computational overhead and reduced scalability. Hence, AdaFisher provides a compelling balance, leveraging enhanced second-order insights to maintain robust convergence while remaining computationally efficient for large-scale DL applications.

Empirical results, as summarized in Table 4.1 and illustrated in Figure 4.1, provide strong evidence that AdaFisher yields reduced variance in the Fisher estimates and drives the optimizer toward regions of the loss landscape that generalize well. Furthermore, the introduction of AdaFisherW, which integrates weight decay in the spirit of AdamW, underscores the flexibility and practical relevance of our method. Overall, the contributions presented in this chapter highlight a promising direction for the integration of second-order information into adaptive optimization. By bridging the gap between theoretical rigor and practical efficiency, AdaFisher paves the way for future research aimed at developing even more robust, scalable, and generalizable optimization methods for DNNs.

Chapter 5

Experiments

In this chapter, we present a comprehensive evaluation of AdaFisher across a range of tasks in both CV and NLP. Our experiments span image classification (using CIFAR-10, CIFAR-100 (Krizhevsky et al., 2009), Tiny ImageNet (Le & Yang, 2015), and ImageNet-1k (J. Deng et al., 2009)) and language modeling (using WikiText-2 (Merity, Xiong, Bradbury, & Socher, 2017) and Penn Treebank (PTB) (Marcus et al., 1993)). We compare AdaFisher with several baseline optimizers, including SGD, Adam/AdamW, K-FAC, AdaHessian, and Shampoo. In addition, we perform a transfer learning task using pretrained ImageNet-1k weights (Paszke et al., 2019). We finalize this section with a thorough ablation and stabilization studies of AdaFisher.

5.1 Hyper-Parameter Optimization

To ensure a fair comparison, we carefully tuned HP for all optimizers across the different architectures and datasets. The experiments were conducted on six benchmark datasets, with HP tuning performed separately for CNN-based models and Vision Transformers (ViTs). Below, we detail our approach. The baseline optimizers compared include SGD, Adam/AdamW, AdaHessian, K-FAC, and Shampoo. While we conducted a single experiment for ImageNet-1k and Tiny ImageNet datasets due to computation time requirements, the CIFAR results (with or without pretrained weights) represent averaged performance metrics across five independent runs. We conducted one experiment for each language modeling datasets.

Hardware. In total, we had a server with 6 NVIDIA RTX 6000 Ada Generation GPUS with 48 gigabytes of VRAM and 128 gigabytes of RAM available for all experiments. All experiments described in this report were conducted on a system equipped with a single NVIDIA RTX 6000 Ada Generation GPU and 64 gigabytes of RAM, except for training AdaFisher on ImageNet-1k with batch sizes of 512 and 1024, where four GPUs were utilized.

5.1.1 Image Classification Experiments

We provide further results and detailed descriptions of our image classification experiments in this section. We conducted five trials with random initializations for the CIFAR experiments and one trial each for Tiny ImageNet and ImageNet-1k, presenting the mean and standard deviation of the results for these trials.

Note on training time. Given that various optimizers demonstrate significantly different epoch durations, we have standardized our comparisons by restricting training to the total wall-clock time (WCT) consumed by 200 epochs using AdaFisher for both CIFAR and Tiny ImageNet experiments. Conversely, for ImageNet-1k, we report the results based on 90 WCT training epochs using Adam, as, surprisingly, AdaFisher and Adam exhibited the same duration in this experiment. The final selected number of epochs for each optimizer is detailed in Table 5.1. Note that we were unable to train AdaHessian on ImageNet-1k due to the significant computational resources required by this optimizer.

Table 5.1: Comparison of the final epoch counts for various optimizers across different datasets.

	CIFAR10/100 & Tiny ImageNet						ImageNet-1k			
Optimizers	SGD	Adam/AdamW	AdaHessian	K-FAC	Shampoo	AdaFisher/AdaFisherW	Adam	K-FAC	Shampoo	AdaFisher
Epochs	226	210	89	107	36	200	90	60	26	90

HP Tuning. Effective hyperparameter (HP) tuning is crucial for optimizing the performance of deep learning models. In this study, we systematically explored various hyperparameters for both CNNs and ViTs across multiple image classification tasks. The following sections detail the tuning strategies employed for each model architecture and dataset.

CNNs. For all image classification tasks involving CNNs, we utilized ResNet18 as the backbone architecture and evaluated its performance on the CIFAR-10 dataset with a fixed batch size of 256, trained for 50 epochs. The HP tuning process encompassed several components. First, for optimizer selection and learning rate tuning, each optimizer was fine-tuned using ResNet18 on CIFAR-10. We performed a grid search to identify the optimal learning rate from the set $\{0.0001, 0.0003, 0.0005, 0.0009, \ldots, 0.1, 0.3, 0.5, 0.9\}$. Second, a cosine annealing learning rate decay strategy was employed, aligning with the number of training epochs specified for each optimizer in Table 5.1. This approach follows the methodology proposed by Loshchilov and Hutter (2019) and was determined to be optimal for our experimental setup. Third, weight decay was applied uniformly at 5×10^{-4} across all optimizers for CIFAR-10 and Tiny ImageNet; an exception was made for MobileNetV3, where the weight decay was set to 1×10^{-5}, while for experiments on ImageNet-1k the weight decay was established at 1×10^{-4}. Fourth, for damping parameter tuning, different strategies were applied. For AdaFisher, K-FAC, and Shampoo, the damping parameter for K-FAC and AdaFisher was searched within the set $\{0.0001, 0.0003, 0.0005, 0.0009, 0.001, 0.003, 0.005, 0.009, 0.01, 0.03, 0.05, 0.09\}$, a range chosen based on prior research (Martens & Grosse, 2015b) and our own experiments that indicated optimal damping values around 1×10^{-3}. For Shampoo, the damping parameter was tuned within the set $\{1 \times 10^{-6}, 3 \times 10^{-6}, 5 \times 10^{-6}, 9 \times 10^{-6}, 1 \times 10^{-5}, 3 \times 10^{-5}, 5 \times 10^{-5}, 9 \times 10^{-5}, 1 \times 10^{-4}, 3 \times 10^{-4}, 5 \times 10^{-4}, 9 \times 10^{-4}\}$, as optimal values typically reside around 1×10^{-5}. In addition, for AdaHessian the Hessian power was tuned within the range $\{0.1, 0.2, \ldots, 0.9, 1.0\}$; for SGD the momentum was tuned within the range $\{0.1, 0.2, \ldots, 0.9, 1.0\}$; and for AdaFisher the decay factor γ was tuned within the set $\{0.1, 0.2, \ldots, 0.9, 0.99\}$, with the optimal value determined to be $\gamma = 0.8$. Finally, for implementation details, we utilized the ASDL library as implemented in PyTorch provided by Osawa, Ishikawa, Yokota, Li, and Hoefler (2023) for both the Shampoo and K-FAC optimizers.

ViTs. For ViT-based image classification tasks, we employed the Tiny Swin Transformer on the CIFAR-10 dataset with a batch size of 256. The HP tuning strategy for ViTs included several elements. Weight decay values were set as indicated in the respective original publications for each model: 1×10^{-2} for Tiny Swin, 5×10^{-2} for FocalNet, and 6×10^{-2} for CCT-2/3×2. For learning

rate tuning, a grid search was conducted over the set $\{0.3, 0.15, 0.1, 0.05, 0.03, 0.015, 0.01, 0.005, 0.003, 0.0015, 0.001\}$ for optimizers such as SGD, AdaFisher, AdaHessian, K-FAC, and Shampoo, since these optimizers typically operate with higher learning rates compared to Adam-based optimizers. For AdamW, the learning rates were adopted from the original publications: 1×10^{-4} for Tiny Swin and FocalNet, and 5.5×10^{-5} for CCT-2/3×2. The same grid search approach was applied for tuning the damping parameter for K-FAC, Shampoo, and AdaFisher, as well as for the Hessian power for AdaHessian, the momentum for SGD, and the decay factors for AdaFisher, as explained in the CNNs section.

This meticulous HP tuning process ensures that each optimizer is optimally configured for the respective model architectures and datasets, thereby facilitating a fair and comprehensive comparison of their performance across different image classification tasks. The final learning rates for all optimizers and models are detailed in Table 5.2.

Table 5.2: Final selected learning rates for each optimizer, tuned using ResNet18 (for CNN) and Tiny Swin (for ViT) on CIFAR10 using a batch size of 256. We selected based on final validation top-1 accuracy.

Architecture	SGD	Adam	AdamW	AdaHessian	K-FAC	Shampoo	AdaFisher	AdaFisherW
CNNs	0.1	0.001	-	0.15	0.3	0.3	0.001	-
ViTs	0.01	-	0.0001/0.000055	0.01	0.003	0.003	-	0.001

Dataset Details and Data Augmentation. The CIFAR-10/100 datasets contain 50k training and 10k test images. We use a batch size of 256. Data augmentation includes 32×32 random-resize cropping, random horizontal flipping, and Cutout (DeVries & Taylor, 2017) (see Takahashi, Matsubara, and Uehara (2020) for details). Tiny ImageNet Comprises 100k training and 10k test images, we perform 64×64 random-resize cropping and random horizontal flipping with a batch size of 256. Finally, ImageNet-1k With 1,281,167 training and 150k test images, we set a batch size of 256. Training employs random resized cropping to 224×244 and random horizontal flipping, while testing uses a resize to 256×256 followed by 224×224 center cropping.

Transfer Learning Experiments. For transfer learning, model weights are initialized using publicly available PyTorch checkpoints, except for the first convolutional layer of ResNet and the final dense layers, which are randomly initialized. Models are trained with a weight decay of 1×10^{-4} (or

1×10^{-5} for MobileNetV3). A grid search over learning rates in $\{0.3, 0.15, 0.1, 0.03, 0.015, 0.01, \ldots, 1 \times 10^{-5}\}$ was performed, with the final selections provided in Table 5.3. A batch size of 256 and cosine learning rate decay were used. Final epoch counts for each optimizer are listed in Table 5.4.

Table 5.3: Final selected learning rates for each optimizer in the transfer learning task, tuned using ResNet50 on CIFAR-10 (batch size 256).

SGD	Adam	AdaHessian	K-FAC	Shampoo	AdaFisher
0.01	0.0001	0.15	0.3	0.03	0.001

Table 5.4: Final epoch counts for various optimizers in the transfer learning task.

SGD	Adam/AdamW	AdaHessian	K-FAC	Shampoo	AdaFisher/AdaFisherW
58	55	22	27	18	50

5.1.2 Language Modeling Experiments

For language modeling, we utilize a streamlined GPT-1 architecture, which incorporates four self-attention layers, a reduction from the original twelve. This configuration retains core modeling capabilities while reducing complexity, encompassing a total of 28,351,488 learnable parameters. To expedite training, we employ pretrained embeddings from OpenAI's GPT, leveraging the benefits of parameter sharing for enhanced efficiency and faster convergence. The models were trained on WikiText-2 and PTB for 50 wall clock time epochs using AdaFisher. Final epoch counts for each optimizer are provided in Table 5.5. For AdamW, we follow the learning rate settings in ElNokrashy, AlKhamissi, and Diab (2022). For the other optimizers, a grid search over $\{0.3, 0.15, 0.1, 0.05, 0.03, 0.015, 0.01, \ldots, 1 \times 10^{-5}\}$ was conducted, and the selected values are summarized in Table 5.6. A batch size of 32 and weight decay of 0.1 were used. Notably, Shampoo failed to converge and K-FAC could not be trained on these tasks.

Table 5.5: Final epoch counts for various optimizers in the language modeling task.

AdamW	AdaHessian	Shampoo	AdaFisherW
55	18	12	50

Table 5.6: Final selected learning rates for each optimizer, tuned using GPT1 on WikiText-2 and PTB (batch size 32), based on final validation perplexity (PPL).

AdamW	AdaHessian	Shampoo	AdaFisherW
5×10^{-5}	0.015	0.003	1×10^{-4}

5.2 AdaFisher for Image Classification Tasks

We commence our analysis by assessing the convergence and generalization capabilities of various models on image classification tasks. Specifically, we deploy ResNet architectures (ResNetX where $X \in \{18, 50, 101\}$), DenseNet121 (G. Huang, Liu, Van Der Maaten, & Weinberger, 2017), MobileNetV3 (Howard et al., 2019), Tiny Swin (Z. Liu et al., 2021), FocalNet (Yang, Li, Dai, & Gao, 2022) and CCT-2/3×2 (Hassani et al., 2021) on CIFAR10 and CIFAR100, while utilizing standard ResNet50 for Tiny ImageNet and ImageNet-1k. The performance outcomes for CIFAR datasets are detailed in Table 5.7. Our empirical evaluation of AdaFisher optimizer across these models and datasets illustrates its efficiency in optimizing image classification, surpassing all SOTA optimizers. We employ the WCT method with a cutoff of 200 epochs for AdaFisher's training, except for ImageNet-1k, where we use a 90-epoch WCT for Adam, which surprisingly matched AdaFisher's training duration. Results confirm AdaFisher's superior classification accuracy on both CNNs and ViTs. The training losses and test errors for the CIFAR experiments, both with and without cutout, are visually represented in Figures C.1, C.2, C.3, and C.4 in Appendix C. Furthermore, Table 5.8 displays the results for the Tiny ImageNet dataset using ResNet50 and Big Swin networks, with visualizations provided in Figure 5.1. AdaFisher and AdaFisherW consistently outperform current SOTA optimizers. Notably, Figure 5.1 illustrates that although AdaFisher converges slower than K-FAC during ResNet50 training, it achieves superior generalization. This is evidenced by lower testing errors, suggesting that AdaFisher tends to converge to a flatter local minimum, enabling smoother transitions between training and testing datasets with minimal generalization loss. For further explanation, see Cha et al. (2021). Note that due to AdaHessian's high memory consumption, we were unable to train it on Big Swin.

Table 5.7: Performance metrics (mean, std) of different networks and optimizers on CIFAR10 and CIFAR100 using batch size 256 (a) without Cutout and (b) with Cutout. Reported using WCT of 200 AdaFisher training epochs as the cutoff.

(a) Without Cutout

Network	CIFAR10						CIFAR100					
	SGD	Adam	AdaHessian	K-FAC	Shampoo	AdaFisher	SGD	Adam	AdaHessian	K-FAC	Shampoo	AdaFisher
ResNet18	$94.89_{0.1}$	$93.64_{0.1}$	$94.05_{0.1}$	$94.04_{0.2}$	$94.52_{0.1}$	$\mathbf{95.02_{0.1}}$	$76.42_{0.1}$	$72.71_{0.2}$	$73.64_{0.2}$	$74.79_{0.2}$	$76.53_{0.1}$	$\mathbf{77.10_{0.2}}$
ResNet50	$95.07_{0.2}$	93.89_{02}	$94.26_{0.1}$	$94.25_{0.1}$	$94.92_{0.1}$	$\mathbf{95.42_{0.2}}$	$77.50_{0.2}$	$73.12_{0.7}$	$75.29_{0.3}$	$75.49_{0.2}$	$77.81_{0.2}$	$\mathbf{78.91_{0.9}}$
ResNet101	$94.77_{0.1}$	$93.14_{0.1}$	$94.73_{0.9}$	$94.23_{0.1}$	$94.22_{0.1}$	$\mathbf{95.51_{0.1}}$	$78.76_{0.2}$	$73.23_{0.4}$	$72.19_{0.2}$	$75.46_{0.3}$	$78.82_{0.1}$	$\mathbf{79.74_{0.3}}$
DenseNet121	$95.11_{0.1}$	$93.74_{0.2}$	$94.54_{0.1}$	$94.97_{0.1}$	$94.99_{0.1}$	$\mathbf{95.29_{0.1}}$	$78.61_{0.2}$	$75.38_{0.3}$	$72.54_{0.9}$	$77.09_{0.3}$	$78.70_{0.3}$	$\mathbf{79.03_{0.2}}$
MobileNetV3	$92.13_{0.2}$	$91.95_{0.1}$	$91.4_{3.1}$	$91.92_{0.1}$	$91.91_{0.2}$	$\mathbf{92.89_{0.1}}$	$73.81_{0.2}$	$65.64_{0.2}$	$60.78_{3.6}$	$69.87_{0.3}$	$68.01_{0.2}$	$\mathbf{73.15_{0.2}}$
Tiny Swin	$80.08_{0.2}$	$87.47_{0.2}$	$78.34_{0.2}$	$66.84_{0.3}$	$68.44_{0.2}$	$\mathbf{89.08_{0.1}}$	$57.43_{0.3}$	$62.20_{0.2}$	$54.12_{0.3}$	$36.12_{0.3}$	$33.75_{0.3}$	$\mathbf{66.47_{0.2}}$
FocalNet	$80.87_{0.2}$	$85.65_{0.1}$	$71.03_{0.3}$	$42.92_{0.2}$	$41.49_{0.2}$	$\mathbf{86.92_{0.1}}$	$45.66_{0.3}$	$52.88_{0.3}$	$38.05_{0.3}$	$11.23_{0.3}$	$11.06_{0.3}$	$\mathbf{52.9_{0.1}}$
CCT-2/3×2	$73.12_{0.2}$	$83.95_{0.1}$	–	$34.63_{1.1}$	$35.1_{0.8}$	$\mathbf{84.63_{0.3}}$	$52.12_{1.2}$	$60.14_{1.1}$	–	$8.06_{0.6}$	$9.76_{0.3}$	$\mathbf{60.63_{0.6}}$

*Note that Adam and AdaFisher were used for all CNN architectures, while AdamW and AdaFisherW were applied for all ViT experiments.

(b) With Cutout

Network	CIFAR10						CIFAR100					
	SGD	Adam	AdaHessian	K-FAC	Shampoo	AdaFisher	SGD	Adam	AdaHessian	K-FAC	Shampoo	AdaFisher
ResNet18	$95.64_{0.1}$	$94.85_{0.1}$	$95.44_{0.1}$	$95.17_{0.2}$	$94.08_{0.2}$	$\mathbf{96.25_{0.2}}$	$76.56_{0.2}$	$75.74_{0.1}$	$71.79_{0.2}$	$76.03_{0.3}$	$76.78_{0.2}$	$\mathbf{77.28_{0.2}}$
ResNet50	$95.71_{0.1}$	$94.45_{0.2}$	$95.54_{0.1}$	$95.66_{0.1}$	$94.59_{0.1}$	$\mathbf{96.34_{0.2}}$	$78.01_{0.1}$	$74.65_{0.5}$	$75.81_{0.3}$	$77.40_{0.4}$	$78.07_{0.4}$	$\mathbf{79.77_{0.4}}$
ResNet101	$95.98_{0.2}$	$94.57_{0.1}$	$95.29_{0.6}$	$96.01_{0.1}$	$94.63_{0.1}$	$\mathbf{96.39_{0.1}}$	$78.89_{0.2}$	$75.56_{0.3}$	$73.38_{0.2}$	$77.01_{0.4}$	$78.83_{0.2}$	$\mathbf{80.65_{0.4}}$
DenseNet121	$96.09_{0.1}$	$94.86_{0.1}$	$96.11_{0.1}$	$96.12_{0.1}$	$95.66_{0.1}$	$\mathbf{96.72_{0.1}}$	$80.13_{0.4}$	$75.87_{0.4}$	$74.80_{0.9}$	$79.79_{0.2}$	$80.24_{0.3}$	$\mathbf{81.36_{0.3}}$
MobileNetV3	$94.43_{0.2}$	$93.32_{0.1}$	$92.86_{3.1}$	$94.34_{0.1}$	$93.81_{0.2}$	$\mathbf{95.28_{0.1}}$	$73.89_{0.3}$	$70.62_{0.3}$	$56.58_{4.5}$	$73.75_{0.3}$	$70.85_{0.3}$	$\mathbf{77.56_{0.1}}$
Tiny Swin	$82.34_{0.2}$	$87.37_{0.6}$	$84.15_{0.2}$	$64.79_{0.5}$	$63.91_{0.4}$	$\mathbf{88.74_{0.4}}$	$54.89_{0.4}$	$60.21_{0.4}$	$56.86_{0.5}$	$34.45_{0.4}$	$30.39_{1.2}$	$\mathbf{66.05_{0.5}}$
FocalNet	$82.03_{0.2}$	$86.23_{0.1}$	$64.18_{0.2}$	$38.94_{0.8}$	$37.96_{0.7}$	$\mathbf{87.90_{0.1}}$	47.76_{03}	$52.71_{0.5}$	$32.33_{0.3}$	$9.98_{0.6}$	$9.18_{0.1}$	$\mathbf{53.69_{0.3}}$
CCT-2/3×2	$78.76_{0.3}$	$83.89_{0.4}$	–	$33.08_{2.3}$	$35.16_{0.4}$	$\mathbf{84.94_{0.3}}$	$54.05_{0.4}$	$59.78_{0.5}$	–	$7.17_{0.2}$	$8.60_{0.1}$	$\mathbf{62.91_{0.5}}$

*Note that Adam and AdaFisher were used for all CNN architectures, while AdamW and AdaFisherW were applied for all ViT experiments.

Table 5.8: Performance of various networks and optimizers on Tiny ImageNet using batch size 256. Reported using wall clock time of 200 AdaFisher training epochs as the cutoff.

Network	Adam	AdaHessian	K-FAC	Shampoo	AdaFisher
ResNet50	53.06	50.21	50.05	53.53	**57.41**
Big Swin	48.11	–	8.89	4.11	**48.86**

5.2.1 ImageNet-1k Training

Training on ImageNet-1k typically requires multiple GPUs and large batch sizes. Our study showcases that AdaFisher achieves superior validation accuracy on a single GPU than its counterparts in scenarios marked by the light blue region. This performance outstrips traditional approaches like SGD, LAMB (You et al., 2020), and LARS (You, Gitman, & Ginsburg, 2017), which typically utilize batch sizes of 16K. While AdaFisher attains SOTA results on a single GPU, it further excels when scaled up in a distributed setting with larger batch sizes. The results, benchmarked using 256 batch size and a WCT of 90 Adam training epochs, are detailed in Table 5.9 and illustrated

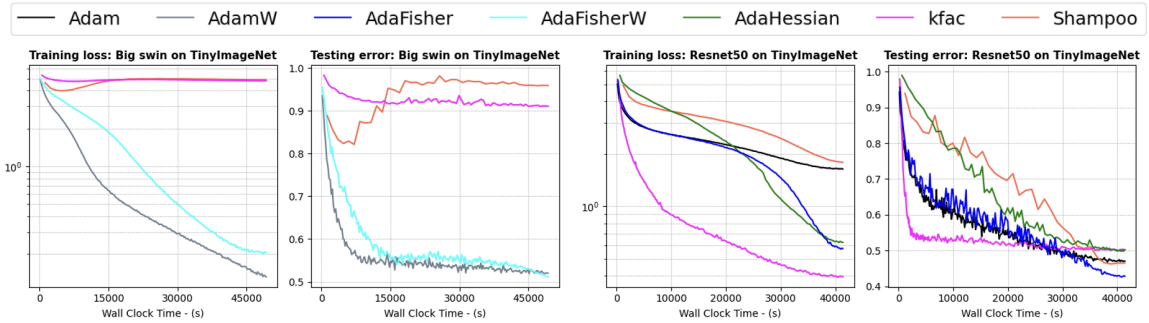

Figure 5.1: WCT training loss and testing error curves of several optimizers on Tiny ImageNet dataset, ResNet-50 and Big Swin with a batch size of 256. AdaFisher consistently achieves lower test error as compared to Adam, AdaHessian, K-FAC and Shampoo. The final accuracy results are reported in Table 5.8.

in Figure 5.2. Distributed AdaFisher curves are illustrated in Figure 5.3. The light blue highlights in the table represent our experiments with a batch size of 256 on a single GPU. The light green indicates results from a distributed version of AdaFisher employing larger batch sizes, whereas the orange reflects results from SOTA methods using a higher batch size of 16K, SGD with a batch size of 256 and AdamW with a batch size of 1024. It is important to note, however, that the training setups and augmentation techniques for the results highlighted in orange, taken from the literature, may differ from those in our study. These results are included to provide a broader context and intuition regarding AdaFisher's performance compared to other experiments. Overall, by balancing curvature-aware updates with parameter efficiency, AdaFisher maintains strong generalization even under constrained computational budgets, a critical advantage over methods like K-FAC that struggle with over-parameterized models. Moreover, Figure 5.3 illustrates the training and validation error of the distributed version of AdaFisher on ImageNet-1k across various batch sizes. AdaFisher not only outperforms its counterparts with smaller batch sizes (256), but it also continues to achieve superior generalization as batch sizes increase. Furthermore, these results reinforce the stability analysis concerning batch sizes presented in Section 6.2, extending it to a more challenging dataset.

Table 5.9: Validation of ImageNet-1k / ResNet50 by different optimizers reported on Top-1 and Top-5 accuracy.

Optimizers	Batch size	Top-1	Top-5
Adam	256	67.78	88.37
K-FAC	256	70.96	89.44
Shampoo	256	72.82	91.42
AdaFisher	256	**76.95**	**93.39**
AdaFisher	512	**77.01**	**93.45**
AdaFisher	1024	**77.09**	**93.56**
SGD Goyal et al. (2017)	256	76.40	-
AdamW X. Chen et al. (2024)	1024	76.34	-
LAMB You et al. (2020)	$16K$	76.66	93.22
SGD You et al. (2020)	$16K$	75.20	-
LARS Huo, Gu, and Huang (2021)	$16K$	75.1	-

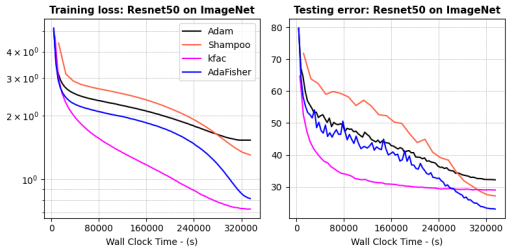

Figure 5.2: Training loss and validation error of ResNet-50 on ImageNet-1k. AdaFisher consistently achieves lower test error as compared to its counterparts.

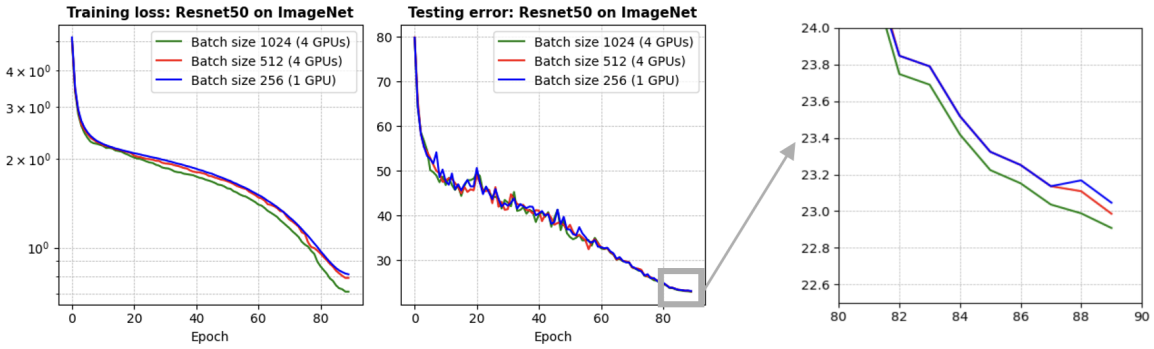

Figure 5.3: Performance of distributed AdaFisher using ResNet50 on ImageNet-1k with different batch sizes for 90 epochs. The final accuracy results are reported in Table 5.9.

5.2.2 Comparison with Other Relevant Methods

In this section, we compare AdaFisher with six baseline optimizers for image classification: SGD, Adam/AdamW, AdaHessian, KFAC, and Shampoo. These baselines were selected because they either represent the current state of the art or utilize second-order gradients, making them suitable comparisons for evaluating second-order optimizers. However, other optimizers, such as AdaFactor Shazeer and Stern (2018) and EVA L. Zhang et al. (2023), are also relevant in this context. AdaFactor is an enhanced Adam memory-efficient optimizer that approximates second-order moments using row and column factorizations, reducing memory consumption for large-scale models. EVA is a second-order optimizer designed to leverage the FIM with efficient matrix inversion

techniques. Therefore, we experimentally compare AdaFisher against the optimizer baselines, including Eva and AdaFactor. Regarding the HPs for EVA, we used the optimal values reported in its original paper and trained the model for 119 epochs using the WCT technique. For AdaFactor, we fine-tuned the learning rate as described in Section 5.1, identifying 0.001 as the optimal value, and trained the model for 216 epochs. Figure 5.4 illustrates the performance comparison on two distinct models: ResNet-18 with CIFAR-100 and MobileNetV3 with CIFAR10. The same data augmentation techniques were applied across all experiments, as detailed in Section 5.1. The best test accuracies achieved are summarized in Table 5.10. AdaFisher demonstrates superior performance compared to the new optimizer baselines, outperforming both EVA and AdaFactor.

Table 5.10: Performance comparison of AdaFisher and other optimizers using ResNet-18 (CIFAR100) and MobileNet-V3 (CIFAR10). Reported using WCT of 200 AdaFisher training epochs as the cutoff.

Network—Optimizer	SGD	Adam	AdaFactor	AdaHessian	K-FAC	Eva	Shampoo	AdaFisher
MobileNet-V3	94.43	93.32	93.21	92.86	94.34	94.41	93.81	**95.28**
ResNet-18	76.56	75.74	69.45	71.79	76.03	76.69	76.78	**77.28**

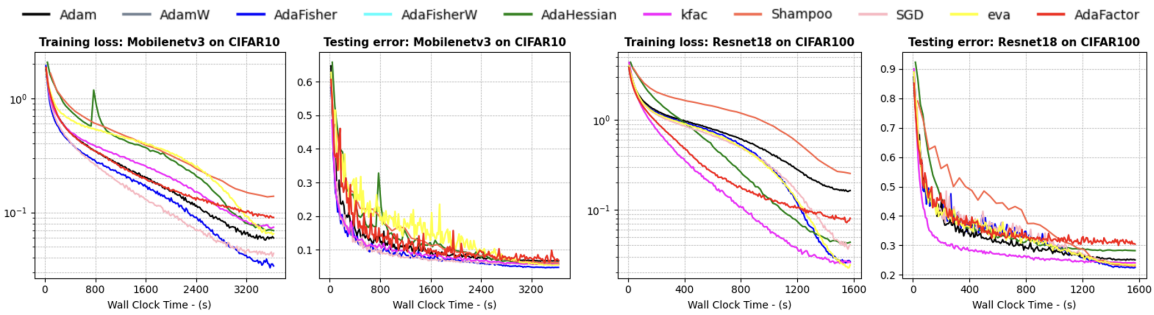

Figure 5.4: WCT training loss, test error, for ResNet-18 on CIFAR100 and MobileNet-V3 on CIFAR10. A batch size of 256 was used. The final accuracy and training time results are summarized in Table 5.10.

5.2.3 Comparison with Consistent Epoch Counts

We evaluated AdaFisher and its counterparts, including two prominent optimizers, Eva and Adafactor, over 200 epochs on ResNet-18, ResNet-50 and MobileNet-V3 using the CIFAR100 dataset. Figure 5.5 illustrates the training loss and test error trends over epochs, along with the

Table 5.11: Performance comparison of AdaFisher and other optimizers using (a) ResNet-18 and (b) MobileNet-V3 on CIFAR100 for 200 epochs.

(a) ResNet-18

Optimizer	SGD	Adam	AdaFactor	AdaHessian	K-FAC	Eva	Shampoo	AdaFisher
Test Acc	76.52	75.71	69.78	76.86	76.96	77.08	**77.35**	77.28
Training Time (min)	20.03	23.33	21.67	96.67	46.46	43.18	216.67	26.58

(b) MobileNet-V3

Optimizer	SGD	Adam	AdaFactor	AdaHessian	K-FAC	Eva	Shampoo	AdaFisher
Test Acc	73.42	70.53	71.08	62.36	75.16	75.48	70.65	**77.56**
Training Time (min)	50.03	56.63	54.22	206.28	116.86	96.78	487.21	60.12

(c) ResNet-50

Optimizer	SGD	Adam	AdaFactor	AdaHessian	K-FAC	Eva	Shampoo	AdaFisher
Test Acc	76.12	73.03	70.78	76.18	77.66	78.01	78.89	**78.91**
Training Time (min)	70.13	76.67	73.32	502.28	149.36	138.58	583.11	83.02

best test error achieved as a function of training time per epoch for all optimizers across both models. Table 5.11 summarizes the highest test accuracy and total training time for each method on both network architectures. Notably, while Shampoo achieved marginally better test accuracy than AdaFisher on ResNet-18, it required approximately eight times longer training time. Conversely, AdaFisher outperformed all baseline optimizers, including Shampoo, in the MobileNet-V3 and ResNet-50 experiments, achieving superior test accuracy while maintaining high efficiency comparable to first-order optimizers.

5.3 AdaFisher for Transfer Learning

Following a more sustainable practice of training DNNs, we employ pretrained models from ImageNet-1k in PyTorch on datasets like CIFAR10 and CIFAR100 to showcase AdaFisher's generalization capability for transfer learning. We applied these pretrained weights across various CNN architectures to train on these datasets. The results, presented in Table 5.12 and Figure C.5, highlight the significant advantages of using AdaFisher, consistently achieving top accuracy across both datasets.

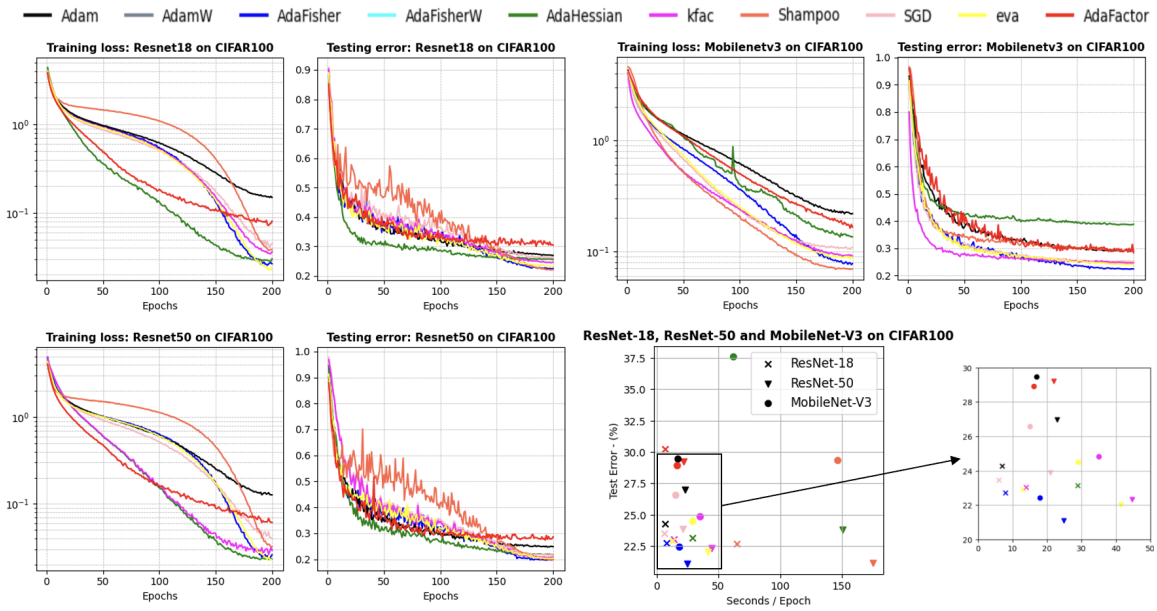

Figure 5.5: Performance comparison of AdaFisher and other well-finetuned optimizers at their best performances using ResNet-18 and MobileNet-V3 on CIFAR-100 for 200 epochs. A batch size of 256 was used. The final accuracy and training time results are summarized in Table 5.11.

Table 5.12: Performance comparison of different networks and optimizers on CIFAR10 and CIFAR100 using ImageNet-1k pretrained weights. Evaluation is based on wall clock time of 50 training epochs with AdaFisher.

Network	CIFAR10						CIFAR100					
	SGD	Adam	AdaHessian	K-FAC	Shampoo	AdaFisher	SGD	Adam	AdaHessian	K-FAC	Shampoo	AdaFisher
ResNet50	$96.50_{0.2}$	$96.45_{0.2}$	$96.35_{0.3}$	$96.45_{0.1}$	$96.03_{0.4}$	$\mathbf{97.13_{0.2}}$	$82.12_{0.1}$	$82.01_{0.4}$	$80.64_{0.9}$	$80.55_{0.4}$	$81.70_{0.2}$	$\mathbf{82.23_{0.2}}$
ResNet101	$97.07_{0.2}$	$96.70_{0.1}$	$96.65_{0.2}$	$96.84_{0.1}$	$96.63_{0.1}$	$\mathbf{97.22_{0.1}}$	$84.01_{0.1}$	$82.43_{0.2}$	$81.36_{0.8}$	$82.26_{0.3}$	$82.65_{0.2}$	$\mathbf{84.47_{0.2}}$
DenseNet121	$94.80_{0.1}$	$94.77_{0.1}$	$93.08_{0.1}$	$94.41_{0.2}$	$94.76_{0.1}$	$\mathbf{95.03_{0.1}}$	$75.98_{0.2}$	$75.65_{0.3}$	$71.06_{0.9}$	$76.10_{0.3}$	$76.08_{0.2}$	$\mathbf{76.92_{0.3}}$
MobileNetV3	$91.76_{0.3}$	$90.92_{0.3}$	$86.45_{2.5}$	$91.72_{0.2}$	$91.39_{0.3}$	$\mathbf{92.78_{0.2}}$	$71.86_{0.4}$	$66.11_{0.8}$	$59.69_{2.3}$	$69.85_{0.4}$	$68.87_{0.3}$	$\mathbf{72.38_{0.4}}$

5.4 AdaFisher for Natural Language Modeling

We employ the WikiText-2 dataset, which encompasses approximately 100 million tokens derived from over 2 million words extracted from a curated set of "Good" and "Featured" articles on Wikipedia. Additionally, we utilize the PTB dataset, renowned for its extensive collection of English words with part-of-speech tags, which has been widely used in NLP tasks for training and benchmarking language models. Our experiments utilize a scaled-down version of GPT-1 (Radford et al., 2019), featuring four self-attention

Table 5.13: Language Modeling performance (PPL) on Wikitext-2 and PTB test dataset (lower is better).

Optimizer	Test PPL	
	WikiText-2	PTB
AdamW	175.06	44.70
AdaHessian	407.69	59.43
Shampoo	1727.75	–
AdaFisherW	**152.72**	**41.15**

layers with masking capabilities with more than 28 million learnable parameters. More details about tuning HPs and models can be found in Section 5.1. The perplexity (PPL) on the test set, corresponding to the best-performing model during validation, is documented in Table 5.13 and Figure 6.3. Similar to approaches in image classification, we apply the WCT method with 50 epochs training time of AdaFisher as the cutoff period. Notice that Shampoo did not achieve convergence despite using optimal HPs, and the K-FAC was unable to train with ASDL library (Osawa et al., 2023).

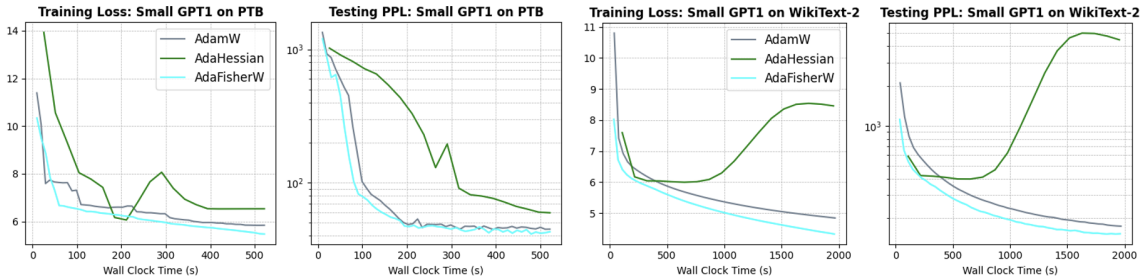

Figure 5.6: Training Loss and Test Perplexity of Small GPT-1 Model on WikiText-2 and PTB Datasets. Experiments were conducted using a batch size of 32 and optimal settings for all optimizers.

5.5 Concluding Remarks

In this chapter, we have presented a comprehensive experimental evaluation of AdaFisher across a wide range of tasks in both computer vision and natural language processing, covering image classification (CIFAR-10, CIFAR-100, Tiny ImageNet, and ImageNet-1k) and language modeling (WikiText-2 and PTB). Our results consistently highlight AdaFisher's effectiveness in comparison to several baseline optimizers. One important observation is that AdaFisher achieves high accuracy across diverse model architectures and datasets. When trained on CIFAR-10 and CIFAR-100 using convolutional neural networks (ResNet-18, ResNet-50, ResNet-101, DenseNet121, MobileNetV3) and vision transformers (Tiny Swin, FocalNet, CCT-2/3×2), AdaFisher and its weight-decay variant outperform both classical approaches like SGD and Adam, as well as second-order methods such as K-FAC, Shampoo, and AdaHessian. This pattern persists on Tiny ImageNet and ImageNet-1k, where AdaFisher demonstrates not only higher final accuracy but also robust convergence. Despite

being a curvature-aware method, AdaFisher exhibits training times that remain comparable to first-order optimizers, especially when contrasted with other second-order techniques.

On practical tasks where methods like K-FAC or Shampoo can become prohibitively expensive, AdaFisher balances curvature estimation with computational efficiency, allowing it to scale well on multi-GPU setups and handle large batch sizes while retaining strong generalization. Another notable feature of AdaFisher is its tendency to locate flatter minima, as evidenced by superior test accuracy and smaller generalization gaps.

The transfer learning experiments, which fine-tuned ImageNet-1k pretrained networks (ResNet-50, ResNet-101, DenseNet121, MobileNetV3) on smaller datasets (CIFAR-10, CIFAR-100), further highlight this characteristic by effectively leveraging partially pretrained weights and refined Fisher Information Matrix updates.

In the language modeling domain, our scaled-down GPT-1 experiments on WikiText-2 and PTB show that AdaFisherW achieves significantly reduced perplexity, surpassing second-order baselines that struggled to converge. Extending our comparisons, we also evaluated AdaFisher against recent optimizers such as EVA and AdaFactor, and AdaFisher maintains superior or on-par performance, confirming that approximate curvature updates can be integrated without severely impacting memory or computational overhead. Altogether, our experiments confirm that AdaFisher outperforms established first- and second-order optimizers while remaining practical.

By incorporating a refined form of the Fisher Information Matrix into an adaptive framework, AdaFisher stands out as a strong choice for training large-scale neural networks in both vision and language tasks. Potential future directions include deeper exploration of curvature approximation techniques in relation to flat minima, expanded distributed implementations for extremely large models, and integration with advanced data augmentation or regularization strategies. These results highlight AdaFisher's capacity to leverage curvature information while preserving computational viability, establishing it as a strong contender for next-generation second-order optimization in deep learning.

Chapter 6

Dissecting Performance: Ablative and Stability Analysis of AdaFisher

In this chapter, we systematically examine the inner workings and robustness of the AdaFisher optimizer. Here, we investigate the impact of several key components that distinguish AdaFisher from its predecessors. Our ablation studies evaluate the effect of different learning rate schedulers, convergence efficiency, the role of removing the square root transformation in the update rule, the use of an EMA for KFs, and the integration of FIM computation in normalization layers. By scrutinizing these elements individually, we aim to elucidate how each contributes to AdaFisher's superior convergence behavior and generalization performance, all while maintaining efficiency across diverse training environments.

6.1 Ablation Studies

This section explores the ablation study of AdaFisher to investigate the impact of various learning rate schedulers and convergence efficiency, as discussed in Section 6.1.1. Additionally, we conduct an in-depth examination of the key components of AdaFisher. This includes analyzing the effects of the EMA, the use of square root transformations, our novel approximation of the FIM, and the critical role of computing the FIM for normalization layers, all of which are detailed in Section 6.1.2.

6.1.1 Evaluating Stability Across Learning Rate Schedulers, and Assessing Convergence Efficiency

Learning rate schedulers. This analysis evaluates the impact of different learning rate schedulers–Cosine Annealing, StepLR, and no scheduler—on the performance of AdaFisher, as depicted in Figure 6.1. AdaFisher exhibits remarkable robustness across these scheduling strategies. Notably, its performance remains stable and efficient, whether it is paired with the gradual adjustments of Cosine Annealing, the abrupt changes of StepLR, or even in the absence of any scheduler. This underscores AdaFisher's adaptability and effectiveness in diverse training environments.

Convergence Efficiency. As training progresses, AdaFisher optimizer demonstrates a significant enhancement in performance compared to its counterparts, especially evident towards the end of the training period (see Appendix C). This rapid convergence is attributed to AdaFisher's approach by incorporating the FIM. Early and mid-training, the FIM serves as an approximation to the Hessian matrix, equivalent to the Generalized Gauss Newton Matrix (Eschenhagen et al., 2024). However, as the model approaches a local minimum, the FIM increasingly aligns precisely with the Hessian (Martens, 2020). This precise alignment accelerates convergence, markedly improving the optimizer's efficiency in the final phases of training. Additionally, AdaFisher's tendency to converge to flat local minima leads to more stable generalization when transitioning from training to testing distributions (Cha et al., 2021), contrasting sharply with other optimizers. To support these points, we analyze the training distribution of our diagonal block-Kronecker FIM during the training of ResNet18 on CIFAR10. Specifically, we examine the FIM distribution for the first (Panel A), middle (Panel B) convolutional layers and the last linear layer (Panel C), as shown in Figure 4.1 In Section 4.2. It is evident that for each layer, the FIM distribution with AdaFisher narrows to smaller values with fewer variations compared to that with Adam. This pattern demonstrates AdaFisher's convergence toward flatter local minima, as the Fisher Information, an approximation of the Hessian, contains crucial curvature information.

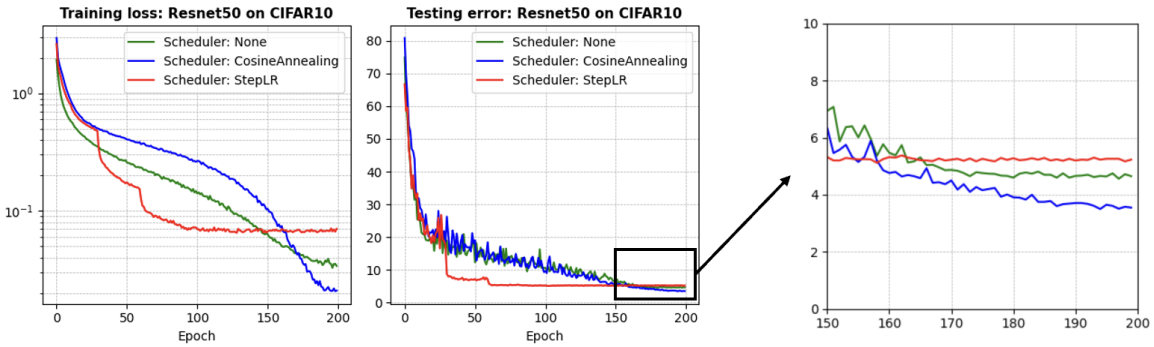

Figure 6.1: Performance comparison of AdaFisher using the ResNet50 on the CIFAR10 with a batch size of 256 with different learning rate schedulers.

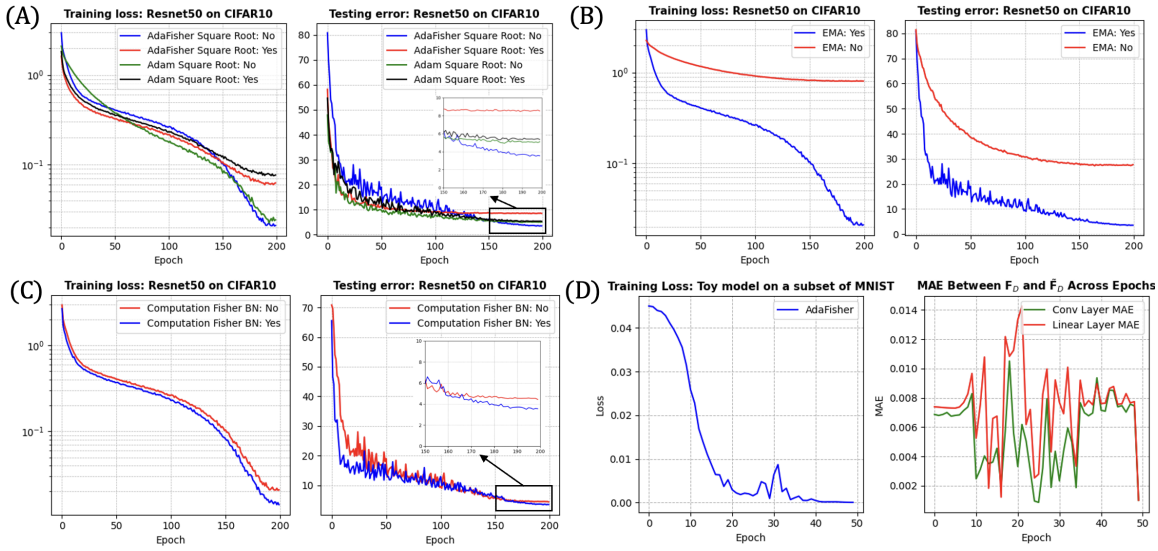

Figure 6.2: AdaFisher Component Analysis. (A) Comparison of MAE between the true FIM F_D and our approximation $\tilde{F}_D$ across convolutional and dense layers. (B) Performance comparison of AdaFisher with and without the EMA of KFs. (C) Assessment of AdaFisher's performance with and without the computation of EFIM for Batch Normalization (BN) layers.

6.1.2 Component Analysis: Evaluating the Significance of AdaFisher's Elements

AdaFisher incorporates several key components, including a novel approximation of the FIM, the EMA of the KFs, the omission of the square root in the update rule, and a new EFIM formula for normalization layers. In this part, we elucidate each component and its significance within the AdaFisher optimizer.

Square Root Utilization. Recent studies, such as (Lin et al., 2024), have reevaluated the necessity of the square root operation in the Adam family's update rules. These studies suggest that eliminating the square root does not affect convergence and may even narrow the generalization gap compared to SGD in CNN models. Our analysis, shown in panel (A) of Figure 6.2, investigates this aspect by comparing the performance of AdaFisher and Adam, both with and without the square root operation. The findings reveal that removing the square root not only boosts the performance and stability of both optimizers but also significantly enhances computational efficiency. Specifically, AdaFisher without the square root not only outperforms the version with the square root but also surpasses Adam without the square root. However, Adam without the square root typically requires an additional **scaling factor** proportional to the batch size, denoted as $f \propto$ batch size, to function correctly. Without this factor, Adam, without the square root, fails to learn effectively, making direct comparisons with AdaFisher invalid.

EMA of KFs. As elucidated in Section 3.3, employing an EMA over the KFs facilitates a more sophisticated curvature estimation. This technique leverages data across multiple mini-batches, enabling continuous updates to the Fisher information rather than relying solely on the data from a single batch. Panel (B) of Figure 6.2 underscores, using ResNet-50 on CIFAR10 over 200 epochs, the benefits of using EMA on KFs, a strategy particularly advantageous in methods that utilize diagonal or block-diagonal approximations of the curvature matrix.

Importance of Fisher Computation for Normalization Layers. The integration of the EFIM in normalization layers, as detailed in Proposition 3.3.1, significantly enhances the generalization process. Panel (C) of Figure 6.2 illustrates the impact of incorporating Fisher computation in these layers during the training of AdaFisher with ResNet-50 on CIFAR10 over 200 epochs. In contrast, the identity matrix is employed when Fisher's computation is omitted. The superior performance of AdaFisher when incorporating Fisher computation can be attributed to the critical role normalization layers play in adjusting the input distribution for each mini-batch. This adjustment substantially enhances the NN's learning stability (Jiang, Gu, Zhu, & Pan, 2024). By quantifying the information

each output $\mathbf{y}$ carries about the parameters $\boldsymbol{\theta}$ under the model distribution $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$, the computation of the FIM in these layers provides valuable insights into parameter sensitivity and gradient variability. This insight is crucial for optimizing training dynamics and enhancing model convergence—areas that are often inadequately addressed by existing optimizers.

New Approximation of the FIM. In Proposition 3.3.2, we introduce a new methodology for approximating the FIM that diverges from the K-FAC optimizer. Unlike K-FAC, which utilizes the full Kronecker product, our approach focuses solely on the diagonal elements of the FIM, where, as demonstrated in Section 3.2, the energy of the KFs is predominantly concentrated. This method enables a more efficient computation of the FIM without sacrificing critical information. To validate our approach, we compare the true FIM diagonal with our approximation in convolutional and dense layers using a toy model composed of 2 convolutional layers and two linear layers on a subset of the MNIST dataset (L. Deng, 2012) over 50 epochs. Specifically, we calculate the true Fisher using the NNgeometry Python package (George, 2021), which facilitates the computation of the FIM, Gauss-Newton Matrix, or NTK applied to neural networks. We estimate $p(\mathbf{y}|\mathbf{x})$ through Monte-Carlo sampling. During each epoch, we collected both the empirical and true Fisher information and calculated the Mean Absolute Error (MAE) between these two measures. Panel (D) of Figure 6.2 showcases the close approximation of AdaFisher's empirical diagonal to the true Fisher, thus validating the efficacy of our approximation method.

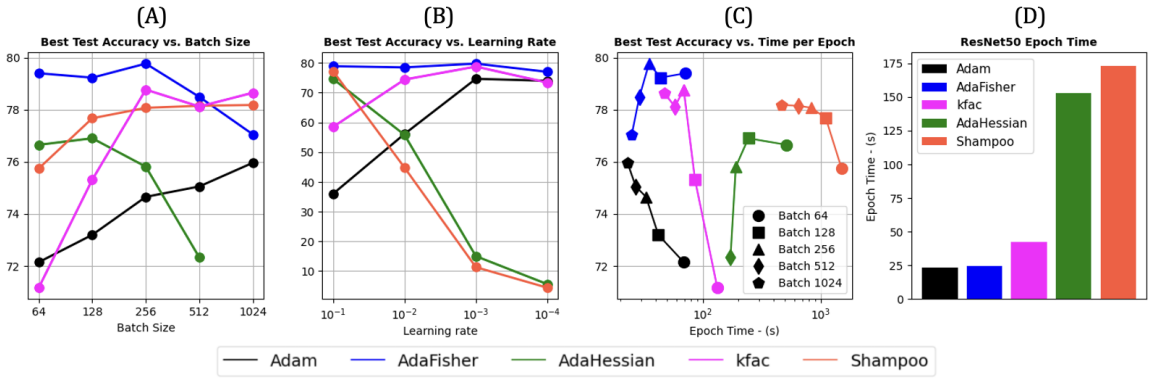

Figure 6.3: Performance comparison of AdaFisher and other optimizers using the ResNet50 network on the CIFAR100 dataset. (A) Test accuracy by batch size. (B) Accuracy vs. learning rates. (C) Accuracy related to epoch time across batch sizes. (D) Epoch time for different optimizers with a batch size of 256.

6.2 Stability Analysis

6.2.1 HP Sensitivity Analysis

In this section, we assess AdaFisher's stability under varying learning rates and batch sizes using ResNet50 on CIFAR100 and compare its performance to other optimizers. Improved stability indicates a reduced need for HP tuning while maintaining high performance. To ensure a fair comparison, all methods were evaluated using a consistent experimental setup, with parameters tailored to each optimizer's strengths. However, we exclude AdaHessian results for a batch size of 1024 due to its significant computation cost.

Batch Size Analysis. We examine the impact of batch size on AdaFisher's performance, as shown in Panels (A) and (C) of Figure 6.3. AdaFisher maintains high test accuracy across various batch sizes, excelling particularly at smaller sizes despite some sensitivity to larger ones. Panel (C) highlights AdaFisher's efficiency, achieving high accuracy with shorter epoch times compared to Adam, detailed further in Panel (D), where AdaFisher shows competitive epoch durations against other optimizers. These results, discussed in Section 6.2.2, underscore AdaFisher's effective performance across batch size variations without adjusting other HPs.

Learning Rate Stability. This analysis evaluates the impact of learning rate variations on AdaFisher's performance, as depicted in Panel (B) of Figure 6.3. AdaFisher demonstrates superior stability, particularly at lower learning rates, maintaining consistent performance across a broad spectrum. This stability alleviates the need for meticulous learning rate adjustments, thereby streamlining model training in various computational environments. Additionally, AdaFisher's stability across various learning rates can be attributed to its effective approximation of the curvature matrix.

6.2.2 Comparison of Training Speed and Memory Utilization

As discussed in Section 6.2, AdaFisher emerges as a balanced trade-off between time complexity and performance. Similarly, its memory footprint is comparable to that of Adam, showcasing efficient VRAM utilization. We extend our stability analysis to the CIFAR10 dataset to provide a dataset-independent evaluation of performance metrics, as depicted in panel (A) of Figure 6.4.

Additionally, we analyze the memory usage for different batch sizes using the ResNet-50 model on the CIFAR-10/100, presented in panel (B) of Figure 6.4. The analysis reveals that AdaFisher while maintaining high accuracy levels, uses memory comparably to Adam, especially evident in larger batch sizes. This suggests that AdaFisher can achieve competitive performance without excessive VRAM consumption, making it an optimal choice for scenarios with memory constraints.

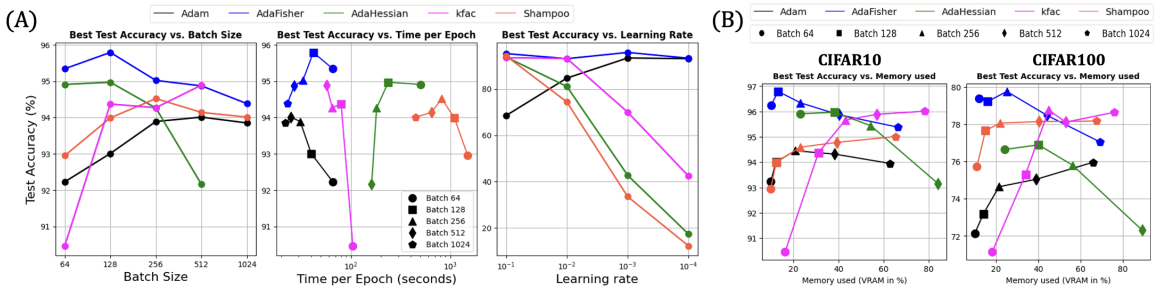

Figure 6.4: (A) Performance comparison of AdaFisher and other optimizers across various batch sizes, epoch times and learning rates (with a batch size of 256), evaluated using the ResNet50 on the CIFAR-10. (B) Performance comparison of AdaFisher and other optimizers regarding the memory used, assessed using ResNet50 and CIFAR10/100 across different batch sizes. This figure highlights how AdaFisher competes closely with Adam in terms of memory efficiency and performance.

Epoch Times. Continuing our analysis of the time complexity for each optimizer, we present in Figure 6.5 the epoch times for various network architectures and datasets. Specifically, we compare the epoch times of Adam, AdaFisher, K-FAC, AdaHessian, and Shampoo optimizers on CIFAR10 and CIFAR100 datasets. As depicted in Figure 6.5 panel (A), AdaFisher demonstrates a comparable training time to Adam across multiple network architectures on the CIFAR10 dataset. This indicates that AdaFisher achieves efficient optimization without incurring significant additional computational costs. Similarly, in Figure 6.5 panel (B), we observe that the epoch times for AdaFisher remain close to those of Adam on the CIFAR100 dataset. While K-FAC and AdaHessian exhibit increased training times, Shampoo shows the highest epoch times across all tested networks. This further highlights the efficiency of AdaFisher as an optimizer, combining the advantages of advanced optimization techniques with practical training times.

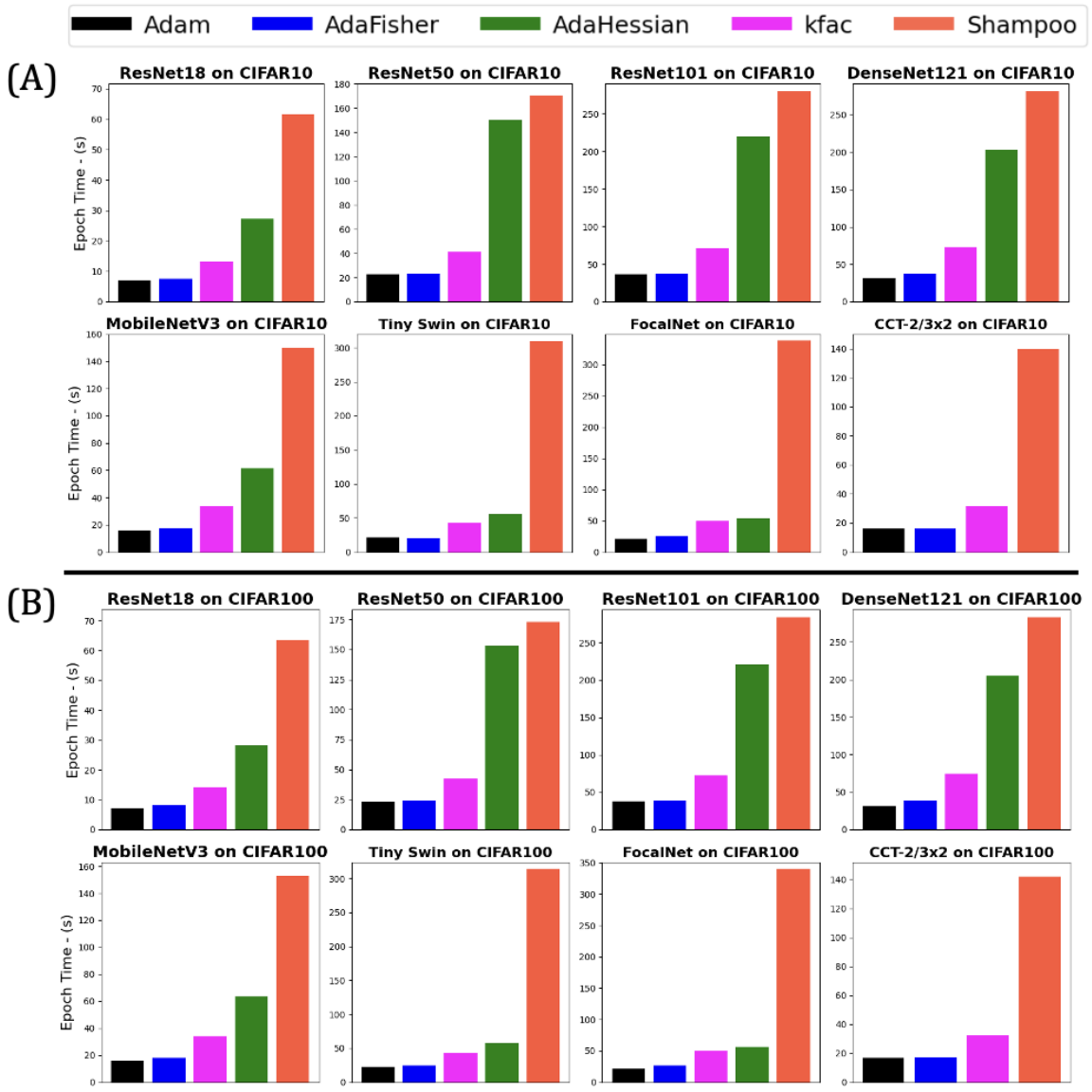

Figure 6.5: Epoch times for various networks on CIFAR10 (A) and CIFAR100 (B) using Adam, AdaFisher, K-FAC, AdaHessian and Shampoo.

6.3 Concluding Remarks

In summary, our comprehensive ablative and stability analyses demonstrate that AdaFisher achieves robust and efficient performance through a judicious integration of adaptive and curvature-aware techniques. The experiments reveal that AdaFisher is resilient across various learning rate

schedulers, and its refined diagonal block-Kronecker approximation of the FIM significantly enhances convergence efficiency, particularly in the later training stages. Notably, eliminating the square root in the update rule and incorporating an EMA over the KFs yield substantial improvements in both performance and computational efficiency. Additionally, the inclusion of FIM computation in normalization layers further stabilizes training and fosters better generalization by steering the optimizer towards flatter minima. Overall, these findings not only validate the design choices underlying AdaFisher but also underscore its potential as a scalable and practical optimizer for modern deep learning applications. The insights provided here offer a promising roadmap for future research in adaptive second-order optimization.

Chapter 7

Concluding Remarks

7.1 Overview of Chapters

This thesis has explored the challenge of making second-order optimization practical for deep learning, focusing on the FIM as a source of curvature information. Chapter 1 introduced the motivation and problem statement, highlighting the gap between the superior convergence properties of second-order methods and their impracticality in modern DNN training. Chapter 2 provided a literature review of optimization techniques, from classical first-order methods to advanced second-order approaches, identifying the central trade-off: first-order methods are efficient but capture limited curvature information, whereas second-order methods use richer curvature information for faster convergence but are often infeasible at scale. Chapter 3 developed a new, efficient approximation of the FIM suitable for large-scale DNNs. By analyzing the FIM's eigen-spectrum and sparsity patterns, we revealed that much of the curvature information can be retained through structured approximations. The chapter introduced a block-diagonal (Kronecker-factored) approximation to the FIM, naturally arising from layer-wise factorization, and demonstrated that a diagonal-centric structure in the FIM emerges during training, justifying using primarily the diagonal of each factor as an efficient proxy for curvature. Chapter 4 introduced AdaFisher, a novel adaptive second-order optimizer leveraging our FIM approximation. AdaFisher employs a refined diagonal block-Kronecker approximation of the FIM as its preconditioner, effectively integrating curvature information into each update step without incurring the full cost of second-order methods. The chapter detailed

AdaFisher's theoretical underpinnings, algorithmic structure, and distributed implementation. We provided theoretical analysis showing that AdaFisher's adaptive nature inherits stability similar to first-order methods while its FIM-based scaling guides the model toward minima that generalize better. Chapter 5 presented extensive experiments validating AdaFisher across computer vision tasks (image classification with CNNs and Vision Transformers) and natural language tasks (language modeling with transformers). Results demonstrated that AdaFisher consistently converged faster and reached lower testing error than baselines including Adam, AdaHessian, K-FAC, and Shampoo. AdaFisher exhibited stability and robustness to hyper-parameter tuning, performing consistently well over a range of settings while using standard learning rate schedules. Chapter 5 provided an in-depth ablation and stability analysis of AdaFisher's design. We confirmed that each key design choice is justified: the adaptive square-root scaling, exponential moving average for Kronecker factors, and FIM-based preconditioning in all layers all contributed to better outcomes. AdaFisher maintained strong performance under different batch sizes and learning rate schedules. Crucially, models trained with AdaFisher had significantly lower effective curvature than those trained with conventional optimizers, indicating that AdaFisher steers optimization toward broader, flatter minima known to correlate with improved generalization.

7.2 Solving Deep Learning Optimization Problem with AdaFisher

The core problem addressed by this thesis is how can we leverage the second order information in an efficient way to both achieve better generalization while maintaining bounded computation overhead. AdaFisher tackles this by using the Fisher Information Matrix as a guide for optimization. Theoretically, AdaFisher can be viewed as an adaptive natural gradient method: it preconditions the gradient with an estimate of the inverse Fisher matrix, effectively normalizing gradient directions by local curvature. This means AdaFisher **takes smaller steps where loss curvature is steep and larger steps where the surface is flat**. This curvature-aware scaling is similar to second-order optimizers but with far less computational overhead. AdaFisher integrates the adaptive moment estimation machinery of Adaptive framework, marrying the efficiency of first-order updates with the precision of second-order curvature information.

From a theoretical standpoint, AdaFisher's use of the FIM grants several advantages over purely first-order methods. The Fisher matrix is positive semi-definite and approximates the Hessian in many cases, helping AdaFisher avoid issues of negative curvature and ill-conditioning that plague Hessian-based methods. AdaFisher's update rule performs gradient descent in a whitened space where coordinate scaling is determined by curvature, yielding more balanced progress across all dimensions of parameter space. By steering updates towards regions favored by the FIM's structure, AdaFisher introduces implicit regularization towards flatter regions of the loss landscape, explaining the improved generalization observed empirically. Experimentally, AdaFisher outperforms other optimizers across various benchmarks. It not only speeds up convergence but also finds solutions with lower test error. On ImageNet and CIFAR benchmarks, AdaFisher consistently achieved lower error rates than Adam and outperformed specialized second-order methods. Unlike other second-order methods which often require large batch sizes and careful tuning, AdaFisher remains robust with standard batch sizes and learning rate schedules.

Compared to Adam, AdaFisher provides more informed preconditioning based on expected curvature rather than gradient magnitudes. Compared to AdaHessian, AdaFisher's use of the Fisher matrix is better suited for classification problems where the Hessian may not be positive-definite. Versus K-FAC and Shampoo, AdaFisher is considerably more lightweight, storing only per-parameter statistics similar to Adam in memory cost, enabling scaling to larger models with far less overhead. AdaFisher also demonstrated efficient distributed training, making it practical for real-world large-scale applications.

7.3 Future Directions

While AdaFisher makes significant progress towards practical second-order optimization, several avenues for improvement and further research remain: Richer Fisher Approximation (**Band-Diagonal FIM**): One immediate enhancement would be to improve the approximation of the Fisher Information Matrix. Currently, AdaFisher relies on the diagonal of each Kronecker factor to construct its preconditioner. A natural extension would be to use a band-diagonal approximation instead, capturing a band of off-diagonal entries around the main diagonal to account for limited

correlations between parameters. This approach would bridge the gap between the sparse diagonal approximation and the full dense matrix: a small bandwidth could significantly increase accuracy of curvature representation with only a modest increase in computation. Future work can analyze the trade-off between bandwidth and performance gains, and develop algorithms to efficiently update and invert these banded approximations. Informative Neurons for Parameter-Efficient Fine-Tuning: AdaFisher's insights could be applied to parameter-efficient transfer learning. The FIM can identify the most "informative" neurons or weights – those to which the loss is most sensitive – and fine-tuning could be restricted to those parameters for new tasks. This would create a systematic approach to parameter-efficient fine-tuning rather than guessing which layers to train. Research could involve computing per-neuron Fisher diagonals after training on a base task, then fine-tuning only neurons with top-tier Fisher values for target tasks. This could be combined with lightweight adaptation modules like LoRA or adapter methods, making large model deployment more efficient.

Efficiency through Low-Level Optimization (CUDA Kernels): To further improve runtime performance, future work could focus on low-level optimization. Developing custom CUDA kernels for AdaFisher's core operations could greatly speed up the optimizer on GPU hardware. By optimizing memory access patterns and parallelizing FIM computations, we could reduce per-iteration overhead to be almost on par with Adam. Integration with deep learning compiler frameworks and exploring mixed-precision implementations might further reduce memory and computation costs without affecting efficacy. These engineering improvements could make AdaFisher practically as fast as the best first-order optimizers. Application to LLMs: An important next step is testing and refining AdaFisher on truly large-scale models, such as Transformer-based LLMs with hundreds of billions of parameters. This will likely require combining memory-efficient approximations and low-level optimizations, since storing even diagonal Fisher information for billions of parameters is non-trivial. It may also require algorithmic adaptations, such as segmenting the model into modules optimized with local curvature information. The payoff could be substantial: AdaFisher might help stabilize the fine-tuning of LLMs on downstream tasks by virtue of its curvature-guided updates. Evaluating AdaFisher on RLHF or long-horizon training could reveal whether curvature information helps navigate complex non-convex training dynamics. Broader Research Opportunities: Additional research directions include theoretical analysis of generalization for curvature-aware

optimizers, combining AdaFisher with techniques like sharpness-aware minimization, developing quantized or memory-compressed variants for enormous models, applying AdaFisher's methodology to continual learning to mitigate catastrophic forgetting, and exploring its use with advanced regularization techniques or in federated learning scenarios.

In summary, the development of AdaFisher opens many future research avenues in both algorithmic innovation and practical applications. By improving the approximation quality, efficiency, and applicability of second-order methods, we move closer to a paradigm where curvature-informed optimization becomes a standard component of deep learning training, leading to faster convergence, better generalization, and more efficient use of computational resources.

Appendix A

Proofs

Proposition A.0.1 (FIM for normalization layer)**.** *Let $(\boldsymbol{\nu}_i, \boldsymbol{\beta}_i) \in \mathbb{R}^{C_i}$ be the scale and shift parameters of a normalization layer i. The empirical KFs for the FIM approximation are*

$$\mathcal{H}_{i-1}\Big|_{\boldsymbol{\nu}_i} = \frac{1}{|\mathcal{T}_i|}\sum_{x\in\mathcal{T}_i}\mathbf{h}_{i-1,x}\mathbf{h}_{i-1,x}^\top,\ \mathcal{H}_{i-1}\Big|_{\boldsymbol{\beta}_i} = \mathbf{1}\mathbf{1}^\top, \quad \mathcal{S}_i = \frac{1}{|\mathcal{T}_i|}\sum_{x\in\mathcal{T}_i}\mathbf{s}_{i,x}\mathbf{s}_{i,x}^\top$$

where $\mathbf{h}_{i-1}, \mathbf{s}_i \in \mathbb{R}^{C_i\times|\mathcal{T}_i|}$ represent the pre-normalized activations and gradients, respectively. Here, $\mathcal{T}_i$ is the set of dimensions over which normalization statistics are computed, and C_i is the channels/features size.

Proof. Let $(\boldsymbol{\nu}_i, \boldsymbol{\beta}_i) \in \mathbb{R}^{C_i}$ be the scale and shift parameters of a normalization layer i, with transformation

$$\mathbf{h}_i = \mathbf{a}_i = \boldsymbol{\nu}_i \odot \mathbf{h}_{i-1} + \boldsymbol{\beta}_i$$

where $\mathbf{h}_{i-1} \in \mathbb{R}^{C_i\times|\mathcal{T}_i|}$ contains normalized activations and $\odot$ denotes element-wise multiplication. Let $\nabla_{\boldsymbol{\nu}_i} J(\boldsymbol{\theta}) = \sum_x \mathbf{h}_{i-1,x} \odot \mathbf{s}_{i,x}$ and $\nabla_{\boldsymbol{\beta}_i} J(\boldsymbol{\theta}) = \sum_x \mathbf{s}_{i,x}$ where $\mathbf{s}_i = \nabla_{\mathbf{h}_i} J(\boldsymbol{\theta})$.

For $\boldsymbol{\nu}_i$ parameters:

$$\begin{aligned}
\mathbb{E}[\nabla_{\boldsymbol{\nu}_i}\mathcal{L}\nabla_{\boldsymbol{\nu}_i}\mathcal{L}^\top] &= \mathbb{E}\left[\left(\sum_x \mathbf{h}_{i-1,x}\odot\mathbf{s}_{i,x}\right)\left(\sum_{x'}\mathbf{h}_{i-1,x'}\odot\mathbf{s}_{i,x'}\right)^\top\right] \\
&\approx \mathbb{E}\left[\sum_x(\mathbf{h}_{i-1,x}\mathbf{h}_{i-1,x}^\top)\otimes(\mathbf{s}_{i,x}\mathbf{s}_{i,x}^\top)\right] \quad \text{(K-FAC independence assumption)} \\
&= \left(\frac{1}{|\mathcal{T}_i|}\sum_x \mathbf{h}_{i-1,x}\mathbf{h}_{i-1,x}^\top\right)\otimes\left(\frac{1}{|\mathcal{T}_i|}\sum_x \mathbf{s}_{i,x}\mathbf{s}_{i,x}^\top\right) \\
&= \mathcal{H}_{i-1}\Big|_{\boldsymbol{\nu}_i}\otimes\mathcal{S}_i
\end{aligned}$$

For $\boldsymbol{\beta}_i$ parameters:

$$\begin{aligned}
\mathbb{E}[\nabla_{\boldsymbol{\beta}_i}\mathcal{L}\nabla_{\boldsymbol{\beta}_i}\mathcal{L}^\top] &= \mathbb{E}\left[\left(\sum_x \mathbf{s}_{i,x}\right)\left(\sum_{x'}\mathbf{s}_{i,x'}\right)^\top\right] \\
&= \mathbf{1}\mathbf{1}^\top\otimes\left(\frac{1}{|\mathcal{T}_i|}\sum_x \mathbf{s}_{i,x}\mathbf{s}_{i,x}^\top\right) \quad \text{(Bias term factorization)} \\
&= \mathcal{H}_{i-1}\Big|_{\boldsymbol{\beta}_i}\otimes\mathcal{S}_i
\end{aligned}$$

Cross-terms between $\boldsymbol{\nu}_i$ and $\boldsymbol{\beta}_i$ are excluded under the diagonal block assumption. □

Proposition A.0.2 (Efficient FIM)**.** *Let $\mathcal{H}_{i-1}$ and $\mathcal{S}_i$ represent the KFs for a given layer index i within a neural network, where these factors exhibit semi-diagonal characteristics indicating energy concentration predominantly along the diagonal, as elaborated in Section 3.2. Define g_i as the gradient obtained through backpropagation at layer i. Assume that $\mathcal{H}_{i-1}$ and $\mathcal{S}_i$ can be closely approximated by diagonal matrices, denoted by $\mathcal{H}_{D_{i-1}}$ and $\mathcal{S}_{D_i}$ respectively at layer i, such that $\mathcal{H}_{D_{i-1}} = Diag(\mathcal{H}_{i-1})$, $\mathcal{S}_{D_i} = Diag(\mathcal{S}_i)$ where $Diag(\mathcal{M})$ denote the diagonal approximation of a matrix $\mathcal{M}$, which retains only the main diagonal. Therefore, we define the Empirical FIM as*

$$\tilde{F}_{D_i} \triangleq \mathcal{H}'_{D_{i-1}}\otimes\mathcal{S}'_{D_i} + \lambda\mathbf{I}, \tag{23}$$

where $\mathcal{M}'$ denotes the Min-Max normalization technique Patro and Sahu (2015) for $\mathcal{M} = \mathcal{H}_{D_{i-1}}$ or $\mathcal{S}_{D_i}$. The regularization parameter λ set to 0.001 serves as damping factors, in alignment with

the principles of Tikhonov regularization, to enhance computational stability and improve the conditioning of the matrix. The foundational aspects of the K-FAC optimization approach are detailed in Martens and Grosse (2015b). Then, the closed-form solution for the preconditioned gradient $\bar{\mathbf{g}}^{(t)}$*, derived from the diagonal approximation of the FIM, is given by:* $\bar{\mathbf{g}}^{(t)} = (\tilde{F}_D^{(t)})^{-1}\mathbf{g}^{(t)}$.

Proof. The justification of our approach comprises two principal components: the rationale for adopting a diagonal approximation of the KFs and the methodology for normalization and regularization of these factors.

Part 1: Diagonalization of KFs

The assumption of independent neuronal activity within layers is foundational to our approach. This assumption posits that the covariance matrices $\mathcal{H}$ and $\mathcal{S}$, encapsulating the second-order statistics of activations and sensitivities, respectively, are diagonal. This diagonal nature arises because independence among random variables implies a covariance of zero for any pair of distinct variables, thereby nullifying all off-diagonal elements of these covariance matrices.

Consider matrices A and B, each being diagonal with elements a_{ii} and b_{jj}, respectively. The Kronecker product $A \otimes B$, by definition, generates elements $a_{ii}b_{jj}$ at the corresponding (i, j) positions. For diagonal A and B, this product maintains non-zero values exclusively at diagonal positions where $i = j$, resulting in:

$$A \otimes B = \text{diag}(a_{11}b_{11}, \ldots, a_{nn}b_{mm}),$$

yielding a purely diagonal matrix. Moreover, we have empirically demonstrated that the energy of the KFs is concentrated along the diagonal, as detailed in Section 3.2. These arguments support our initial premise.

Part 2: Normalization and Regularization

Let $\mathcal{M} \in \{\mathcal{H}_{D_i}, \mathcal{S}_{D_i}\}$ be a diagonal matrix with entries $m_k > 0$. The min-max normalized matrix $\mathcal{M}'$ satisfies

$$\mathcal{M}' = D^{-1}(\mathcal{M} - m_{\min}I)D^{-1}, \quad D = \text{diag}(\sqrt{m_{\max} - m_{\min}})$$

where $m_{\min} = \min_k m_k$, $m_{\max} = \max_k m_k$. This affine transformation ensures that $0 \preccurlyeq \mathcal{M}' \preccurlyeq I$ where $\preccurlyeq$ denotes Loewner ordering. Combined with Tikhonov regularization, the modified FIM, $\tilde{F}_{D_i} = \mathcal{H}'_{D_{i-1}} \otimes \mathcal{S}'_{D_i} + \lambda \mathbf{I}$ admits eigenvalue bounds

$$\lambda \leq \lambda_k(\tilde{F}_{D_i}) \leq 1 + \lambda \quad \forall k$$

which guarantees numerical stability for inversion. This approach ensures that all elements are scaled uniformly, preserving their relative magnitudes and distances. Compared to other normalization methods, such as z-score normalization (Patro & Sahu, 2015), Min-Max normalization offers several advantages such as the normalization Stability, where for $\mathcal{M}' = (\mathcal{M} - m_{\min} I)/(m_{\max} - m_{\min})$ we have $\sigma(\mathcal{M}') \subseteq [0, 1]$ where $\sigma(\cdot)$ denotes matrix spectrum. Moreover, the Kronecker product satisfies $\sigma(\mathcal{H}'_{D_{i-1}} \otimes \mathcal{S}'_{D_i}) \subseteq [0, 1]$, thus $\lambda_{\min}(\tilde{F}_{D_i}) \geq \lambda > 0$, guaranteeing invertibility. And the relative error satisfies $\frac{\|\tilde{F}_{D_i}^{-1} - F_{D_i}^{-1}\|}{\|F_{D_i}^{-1}\|} \leq \mathcal{O}(\epsilon + \lambda)$ where ϵ measures diagonal approximation error. Therefore, the preconditioned gradient can be written has $\bar{\mathbf{g}}^{(t)} = (\mathcal{H}'_{D_{i-1}} \otimes \mathcal{S}'_{D_i} + \lambda \mathbf{I})^{-1} \mathbf{g}^{(t)} = (\tilde{F}_{D_i}^{(t)})^{-1} \mathbf{g}^{(t)}$. □

Proposition A.0.3 (Convergence in convex optimization)**.** *For the FIM defined in Eq. (23), the updating scheme $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha(\tilde{F}_D^{(t)})^{-1} \nabla J(\boldsymbol{\theta}^{(t)})$ converges. Moreover, if ∇J is Lipschitz, i.e., $||\nabla J(\boldsymbol{\theta}) - \nabla J(\boldsymbol{\theta}')||_2 \leq L||\boldsymbol{\theta} - \boldsymbol{\theta}'||$ for any $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$, then for the k-step iteration with a fixed step size $\alpha \leq 1/L$, then*

$$J(\boldsymbol{\theta}^{(k)}) - J(\boldsymbol{\theta}^*) \leq \frac{||\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*||_2^2}{2\alpha k},$$

where $J(\boldsymbol{\theta}^*)$ is the optimal value.

Proof. We follow the same proof as in Yao et al. (2021). Assume that $J(\boldsymbol{\theta})$ is a strongly convex and strictly smooth function in $\mathbb{R}^d$, such that there exist positive constants α and β so that $\alpha I \leq \nabla^2 J(\boldsymbol{\theta}) \leq \beta I$ for all w. We can show that the update formulation $\triangle \boldsymbol{\theta}^{(t)} = (\tilde{F}^{(t)})^{-1} \mathbf{g}^{(t)}$ converges by showing that with the proper learning rate:

$$\triangle \boldsymbol{\theta}^{(t)} \coloneqq J(\boldsymbol{\theta}^{(t+1)}) - J(\boldsymbol{\theta}^{(t)}) \leq -\frac{\alpha}{2\beta^2} ||g^{(t)}||^2$$

Note that when $k = 0$ or 1, the convergence rate is the same as gradient descent or Newton method, respectively. Our proof is similar to Boyd and Vandenberghe (2004) for the Newton method. We denote $\lambda(\boldsymbol{\theta}^{(t)}) = (g^{(t)})^\top (\tilde{F}^{(t)})^{-1}\mathbf{g}^{(t)})^{1/2}$. Since $J(\boldsymbol{\theta})$ is strongly convex, we have

$$\begin{aligned} J(\boldsymbol{\theta}^{(t)} - \eta\triangle\boldsymbol{\theta}^{(t)}) &\leq J(\boldsymbol{\theta}^{(t)}) - \eta(\mathbf{g}^{(t)})^\top\triangle\boldsymbol{\theta}^{(t)} + \frac{\eta^2\beta||\triangle\boldsymbol{\theta}^{(t)}||^2}{2} \\ &\leq J(\boldsymbol{\theta}^{(t)}) - \eta\lambda(\boldsymbol{\theta}^{(t)})^2 + \frac{\beta}{2\alpha}\eta^2\lambda(\boldsymbol{\theta}^{(t)})^2. \end{aligned}$$

The second inequality comes from the fact that

$$\lambda(\boldsymbol{\theta}^{(t)})^2 = \triangle(\boldsymbol{\theta}^{(t)})^\top\tilde{F}^{(t)}\triangle\boldsymbol{\theta}^{(t)} \geq \alpha||\triangle\boldsymbol{\theta}^{(t)}||^2.$$

Therefore, the step size $\hat{\eta} = \alpha/\beta$ will make f decrease as follows,

$$J(\boldsymbol{\theta}^{(t)} - \hat{\eta}\triangle\boldsymbol{\theta}^{(t)}) - J(\boldsymbol{\theta}^{(t)}) \leq -\frac{1}{2}\hat{\eta}\lambda(\boldsymbol{\theta}^{(t)})^2.$$

Since $\alpha I \preceq \tilde{F}^{(t)} \preceq \beta I$, we have

$$\lambda(\boldsymbol{\theta}^{(t)})^2 = (\mathbf{g}^{(t)})^\top(\tilde{F}^{(t)})^{-1}\mathbf{g}^{(t)} \geq \frac{1}{\beta}||\mathbf{g}^{(t)}||^2.$$

Therefore,

$$J(\boldsymbol{\theta}^{(t)} - \hat{\eta}\triangle\boldsymbol{\theta}^{(t)}) - J(\boldsymbol{\theta}^{(t)}) \leq -\frac{1}{2\beta}\hat{\eta}||\mathbf{g}^{(t)}||^2 = -\frac{\alpha}{2\beta^2}||\mathbf{g}^{(t)}||^2 \tag{24}$$

Since $F_D^{(t)}$ is positive definite, hence Eq. (24) holds true. For the bound on convergence rate, we refer to Ryu and Boyd (2016) for the details of the complete proof. □

Proposition A.0.4 (Convergence in non-convex stochastic optimization)**.** *Under the assumptions:*

(i) f is lower bounded and differentiable; $||\nabla J(\boldsymbol{\theta}) - \nabla J(\boldsymbol{\theta}')||_2 \leq L||\boldsymbol{\theta} - \boldsymbol{\theta}'||_2$, $||\tilde{F}_D^{(t)}||_\infty < L$, $\forall t, \boldsymbol{\theta}, \boldsymbol{\theta}'$.

(ii) Both the true and stochastic gradient are bounded, i.e. $||\nabla J(\boldsymbol{\theta}^{(t)})||_2 \leq \lambda$ and $||g_t||_2 \leq \lambda$, $\forall t$ for some $\lambda > 0$.

(iii) Unbiased and independent noise in $\mathbf{g}^{(t)}$, i.e. $\mathbf{g}^{(t)} = \nabla J(\boldsymbol{\theta}^{(t)}) + \zeta^{(t)}$, $\mathbb{E}[\zeta^{(t)}] = 0$, and $\zeta^{(t)} \perp$

$\zeta^{(t)}, \forall i \neq j$.

Assume $\eta^{(t)} = \frac{\eta}{\sqrt{t}}$, $\beta^{(t)} \leq \beta \leq 1$ *is non-increasing,* $\frac{\tilde{F}_D^{(t-1)}[j]}{\eta^{(t-1)}} \leq \frac{\tilde{F}_D^{(t)}[j]}{\eta^{(t)}}$, $\forall t \in [T], j \in [d]$, *we then have*

$$\min_{t\in[T]} \mathbb{E}[||\nabla J(\boldsymbol{\theta}^{(t)})||_2^2] \leq \frac{L}{\sqrt{T}}(C_1\eta^2\lambda^2(1+\log T) + C_2 d\eta + C_3 d\eta^2 + C_4) \tag{25}$$

where C_1, C_2, C_3 *are constants independent of* d *and* T, C_4 *is a constant independent of* T, *the expectation is taken w.r.t all the randomness corresponding to* $\{g^{(t)}\}$.

Proof. Follow X. Chen, Liu, Sun, and Hong (2019b), as AdaFisher is an Adam-type method with the condition $||\eta^{(t)} m^{(t)}/\tilde{F}_D^{(t)}||_2 \leq G$ for some G (which can be obtained by $\eta^{(t)} < \eta$, $||\mathbf{g}^{(t)}||_2 \leq \lambda$ and $||\tilde{F}_D^{(t)}||_2 \geq 1$), we have

$$\begin{aligned}\mathbb{E}\left[\sum_{t=1}^{T} \eta^{(t)} \langle \nabla J(\boldsymbol{\theta}^{(t)}), \nabla J(\boldsymbol{\theta}^{(t)})/\tilde{F}_D^{(t)} \rangle\right] \leq & \mathbb{E}\left[C_1 \sum_{t=1}^{T} \left\| \frac{\eta^{(t)}\mathbf{g}^{(t)}}{\tilde{F}_D^{(t)}} \right\|_2^2 + C_2 \sum_{t=1}^{T} \left\| \frac{\eta^{(t)}}{\tilde{F}_D^{(t)}} - \frac{\eta^{(t-1)}}{\tilde{F}_D^{(t-1)}} \right\|_1 \right. \\ & \left. + C_3 \sum_{t=1}^{T} \left\| \frac{\eta^{(t)}}{\tilde{F}_D^{(t)}} - \frac{\eta^{(t)}}{\tilde{F}_D^{(t)}} \right\|_2^2 \right] + C_4. \end{aligned} \tag{26}$$

We first bound non-constant terms in RHS of Eq. (26). For the term with C_1, since $||\tilde{F}_D^{(t)}||_2 \geq 1$, we have

$$\begin{aligned}\mathbb{E}\left[\sum_{t=1}^{T} \left\| \frac{\eta^{(t)}\mathbf{g}^{(t)}}{\tilde{F}_D^{(t)}} \right\|_2^2\right] &\leq \mathbb{E}\left[\sum_{t=1}^{T} ||\eta^{(t)}\mathbf{g}^{(t)}||_2^2\right] \\ &= \mathbb{E}\left[\sum_{t=1}^{T} \left\| \frac{\eta}{\sqrt{t}}\mathbf{g}^{(t)} \right\|_2^2\right] \\ &\leq \eta^2\lambda^2 \sum_{t=1}^{T} \frac{1}{t} \leq \eta^2\lambda^2(1+\log T).\end{aligned}$$

For the term with C_2, we have

$$\begin{aligned}\mathbb{E}\left[\sum_{t=1}^{T}\left\|\frac{\eta^{(t)}}{\tilde{F}_D^{(t)}}-\frac{\eta^{(t-1)}}{\tilde{F}_D^{(t-1)}}\right\|_1\right] &= \mathbb{E}\left[\sum_{j=1}^{d}\sum_{t=2}^{T}\left(\frac{\eta^{(t-1)}}{\tilde{F}_D^{(t-1)}[j]}-\frac{\eta^{(t)}}{\tilde{F}_D^{(t)}[j]}\right)\right] \\ &= \mathbb{E}\left[\sum_{j=1}^{d}\frac{\eta^{(1)}}{\tilde{F}_D^{(1)}[j]}-\frac{\eta^{(T)}}{\tilde{F}_D^{(T)}[j]}\right] \\ &\leq \mathbb{E}\left[\sum_{j=1}^{d}\frac{\eta^{(1)}}{\tilde{F}_D^{(1)}[j]}\right] \leq d\eta\end{aligned}$$

where the first equality is due to $\frac{\tilde{F}_D^{(t-1)}[j]}{\eta^{(t-1)}} \leq \frac{\tilde{F}_D^{(t)}[j]}{\eta^{(t)}}, \forall t \in [T], j \in [d]$.

For the term with C_3, we have

$$\begin{aligned}\mathbb{E}\left[\sum_{t=1}^{T}\left\|\frac{\eta^{(t)}}{\tilde{F}_D^{(t)}}-\frac{\eta^{(t-1)}}{\tilde{F}_D^{(t-1)}}\right\|_2^2\right] &= \mathbb{E}\left[\sum_{t=1}^{T}\sum_{j=1}^{d}\left(\frac{\eta^{(t)}}{\tilde{F}_D^{(t)}[j]}-\frac{\eta^{(t-1)}}{\tilde{F}_D^{(t)}[j]}\right)^2\right] \\ &= \mathbb{E}\left[\sum_{t=1}^{T}\sum_{j=1}^{d}\left|\frac{\eta^{(t)}}{\tilde{F}_D^{(t)}[j]}-\frac{\eta^{(t-1)}}{\tilde{F}_{D^{(t-1)}}[j]}\right|\cdot\left|\frac{\eta^{(t)}}{\tilde{F}_D^{(t)}[j]}-\frac{\eta^{(t-1)}}{\tilde{F}_D^{(t-1)}[j]}\right|\right] \\ &\leq \mathbb{E}\left[\sum_{t=1}^{T}\sum_{j=1}^{d}\left|\frac{\eta^{(t)}}{\tilde{F}_D^{(t)}[j]}-\frac{\eta^{(t-1)}}{\tilde{F}_D^{(t-1)}[j]}\right|\cdot\left|\frac{\eta}{\sqrt{t}\tilde{F}_D^{(t)}[j]}-\frac{\eta}{\sqrt{t-1}\tilde{F}_D^{(t-1)}[j]}\right|\right] \\ &\leq \mathbb{E}\left[\eta\sum_{t=1}^{T}\sum_{j=1}^{d}\left|\frac{\eta_t}{\tilde{F}_D^{(t)}[j]}-\frac{\eta^{(t-1)}}{\tilde{F}_D^{(t-1)}[j]}\right|\right] \\ &= \eta\mathbb{E}\left[\sum_{t=1}^{T}\left\|\frac{\eta^{(t)}}{\tilde{F}_D^{(t)}}-\frac{\eta^{(t-1)}}{\tilde{F}_D^{(t-1)}}\right\|_1\right] \\ &\leq d\eta^2\end{aligned}$$

Hence

$$\begin{aligned}&\mathbb{E}\left[C_1\sum_{t=1}^{T}\left\|\frac{\eta^{(t)}g^{(t)}}{\tilde{F}_D^{(t)}}\right\|_2^2+C_2\sum_{t=1}^{T}\left\|\frac{\eta^{(t)}}{\tilde{F}_D^{(t)}}-\frac{\eta^{(t-1)}}{\tilde{F}_D^{(t-1)}}\right\|_1+C_3\sum_{t=1}^{T}\left\|\frac{\eta^{(t)}}{\tilde{F}_D^{(t)}}-\frac{\eta^{(t-1)}}{\tilde{F}_{D^{(t-1)}}}\right\|_2^2\right]+C_4 \\ &\qquad\leq C_1\eta^2\lambda^2(1+\log T)+C_2d\eta+C_3d\eta^2+C_4 \qquad (27)\end{aligned}$$

Now we lower bound the LHS of Eq. (25). With the assumption $||\tilde{F}_D^{(t)}||_\infty \leq L$, we have

$$(\eta^{(t)}/\tilde{F}_D^{(t)})_j \geq \frac{\eta}{L\sqrt{t}}.$$

Thus

$$\mathbb{E}\left[\sum_{t=1}^{T} \eta^{(t)} \langle \nabla J(\boldsymbol{\theta}^{(t)}), \nabla J(\boldsymbol{\theta}^{(t)})/\tilde{F}_D^{(t)} \rangle\right] \geq \mathbb{E}\left[\sum_{t=1}^{T} \frac{\eta}{L\sqrt{t}} ||\nabla J(\boldsymbol{\theta}^{(t)})||_2^2\right] \geq \frac{\sqrt{T}}{L} \min_{t\in[T]} \mathbb{E}[||\nabla J(\boldsymbol{\theta}^{(t)})||_2^2] \tag{28}$$

Combining Eq. (27) and (28) gives the desired result. □

Appendix B

Visualization

The convergence rate of an optimizer is crucial, serving as an indicator of its robustness against saddle points and its ability to generalize effectively. In this section, we introduce a novel methodology for visualizing the convergence behavior of optimizers through a statistical model, as depicted in Figure 1.1. Initially, our process employs Principal Component Analysis (PCA) for dimensionality reduction, reducing the dataset dimensions from $\mathcal{D} \in \mathbb{R}^{m \times n}$ to $\hat{\mathcal{D}} \in \mathbb{R}^{m \times 2}$, following the protocol established in F.R.S. (1901). We then apply this reduced dataset to a toy model composed of an L-layer multi-layer perceptron (MLP). Notably, we focus on the first weight matrix W_1^e of this MLP, which resides in $\mathbb{R}^2$, where e denotes the epoch number. For consistency and to ensure comparability, all layers' weights are initialized identically across different optimizers. Following

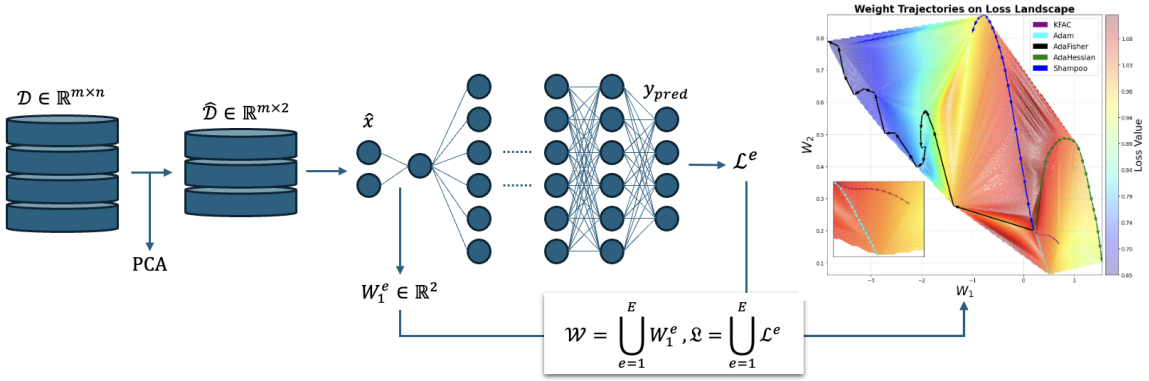

Figure B.1: Pipeline for visualization of optimization paths for various algorithms on a loss surface, comparing their convergence efficiency.

the training phase with various optimizers where we denote a set of optimizer results $\mathcal{O}$, we analyze

both the collection of first-layer weights, $\mathcal{W}$, and the evolution of the loss function, $\mathfrak{L}$ defined as:

$$\mathcal{W} = \begin{bmatrix} (W_1^1)^\top \\ (W_1^2)^\top \\ \vdots \\ (W_1^E)^\top \end{bmatrix}, \quad \mathfrak{L} = [\mathcal{L}_1^1, \mathcal{L}_1^2, \ldots, \mathcal{L}_1^E]^\top$$

where $(W_1^e)^\top$ represents the weight vector at the eth epoch, and $\mathcal{L}_1^e$ represents the loss at the eth epoch, extracted from the optimization results $\mathcal{O}$. We construct a grid $(\mathbf{X}, \mathbf{Y})$ spanning the range of weight parameters, discretized into 200 linearly spaced points along each axis:

$$\mathbf{X}, \mathbf{Y} = \text{meshgrid}\left(\min(\mathcal{W}_{:,1}), \max(\mathcal{W}_{:,1}), \min(\mathcal{W}_{:,2}), \max(\mathcal{W}_{:,2}), 200\right)$$

Finally, we interpolate the loss values $\mathcal{L}$ over the grid using cubic interpolation to obtain a smooth loss surface $\mathbf{Z}$:

$$\mathbf{Z} = \text{griddata}(\mathcal{W}, \mathcal{L}, (\mathbf{X}, \mathbf{Y}), \text{method} = \texttt{cubic})$$

These elements are integral to the visualization process, which elucidates the optimizer's trajectory through the parameter space across training epochs. It is important to note that while we focus on the first layer's weight matrix for clarity, the methodology can be adapted to visualize the weights of any layer within the network. Figure B.1 summarizes the pipeline.

In the experiment depicted in Figure 1.1, we selected the IRIS dataset (*iris*, 2018), owing to its widespread recognition and compatibility with PCA application. Our model employs a 2-layer MLP architecture. We specifically attend to the weight matrix of the first layer, denoted by $W_1 \in \mathbb{R}^2$. This particular focus is informed by the empirical observation that the parameters of the first layer tend to exhibit a faster convergence rate compared to those of subsequent layers in the network. Such a phenomenon can be attributed to the more direct influence of the input features on the first layer's weights, which often results in a more pronounced and expedited learning dynamic. Given the classification nature of the task, we employed the Cross-Entropy loss function (Z. Zhang & Sabuncu,

2018). The network was trained over 20 epochs using a suite of optimizers: Adam, AdaHessian, K-FAC, Shampoo, and AdaFisher. We standardized the learning rate across all optimizers at 1×10^{-3} to ensure comparability of results. Examination of Figure 1.1 reveals that AdaFisher's convergence is markedly superior to that of its counterparts, achieving rapid convergence to the local minimum of the loss landscape concerning the first weight parameter within a few iterations. Conversely, the alternative optimizers demonstrate convergence to less optimal local minima. Note that while the results may vary due to the stochastic nature of parameter initialization, AdaFisher typically converges to a better local minimum compared to its counterparts.

Appendix C

Results

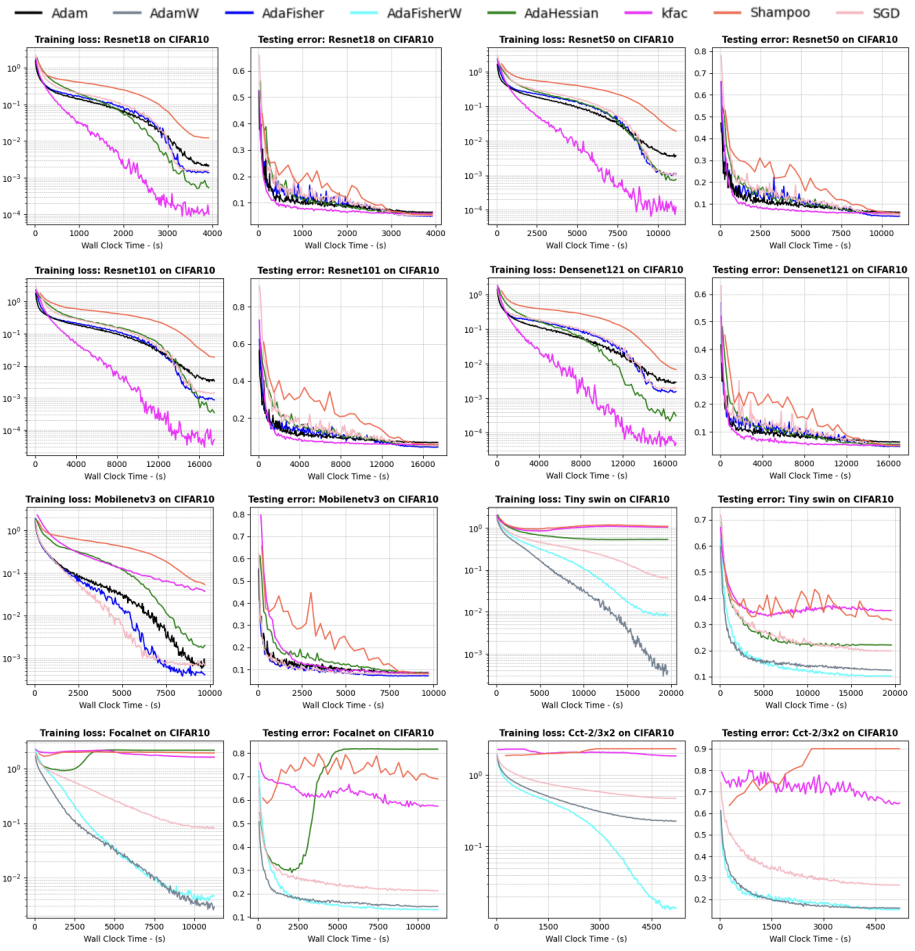

Figure C.1: WCT training loss, test error, for CNNs and ViTs on CIFAR10 experiments, without Cutout. A batch size of 256 was used, and all networks were tuned using ResNet18 applied on CIFAR10. The final accuracy results are reported in Table 5.7 (a).

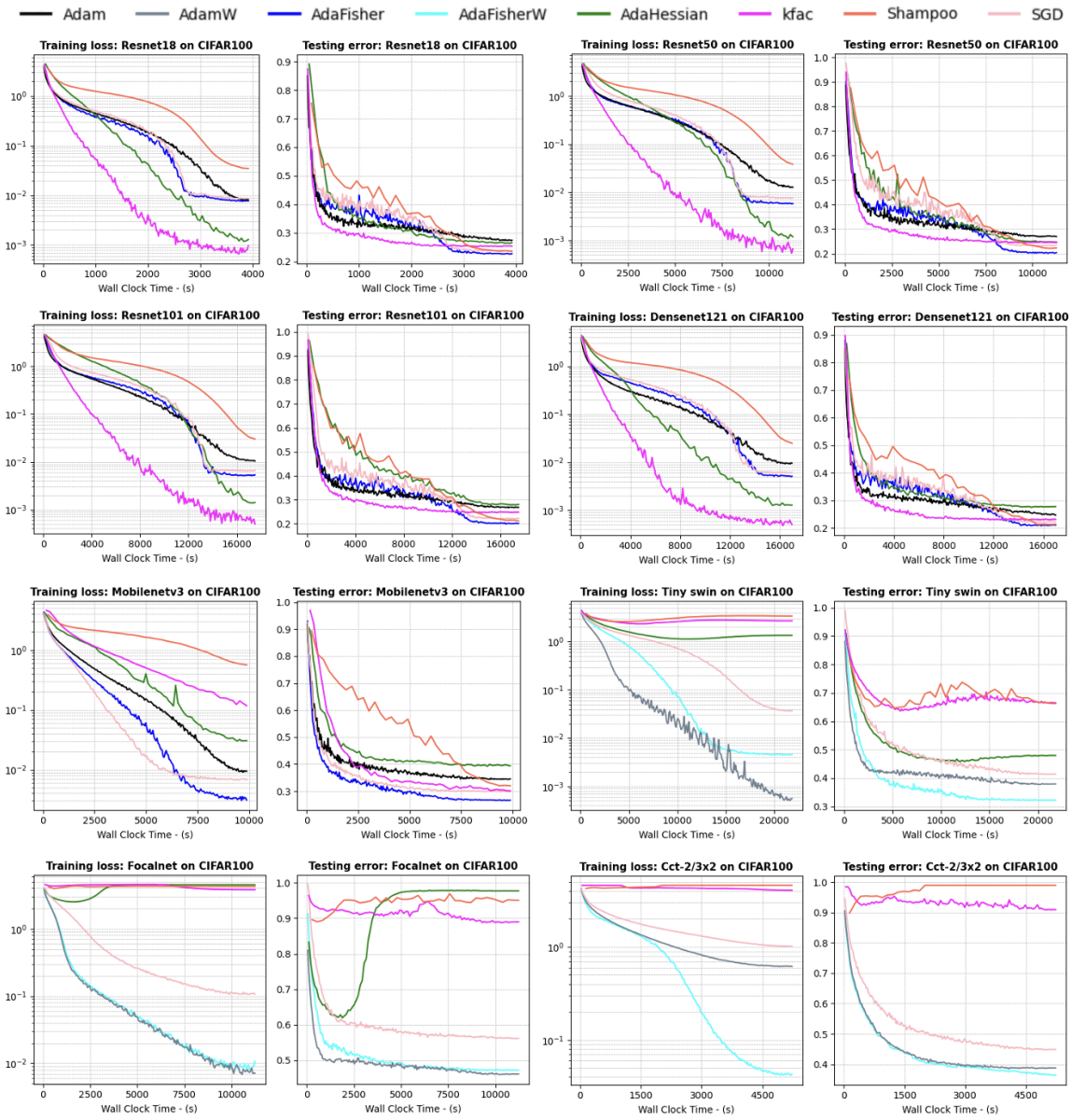

Figure C.2: WCT training loss, test error, for CNNs and ViTs on CIFAR100 experiments, without Cutout. A batch size of 256 was used, and all networks were tuned using ResNet18 applied on CIFAR10. The final accuracy results are reported in Table 5.7 (a).

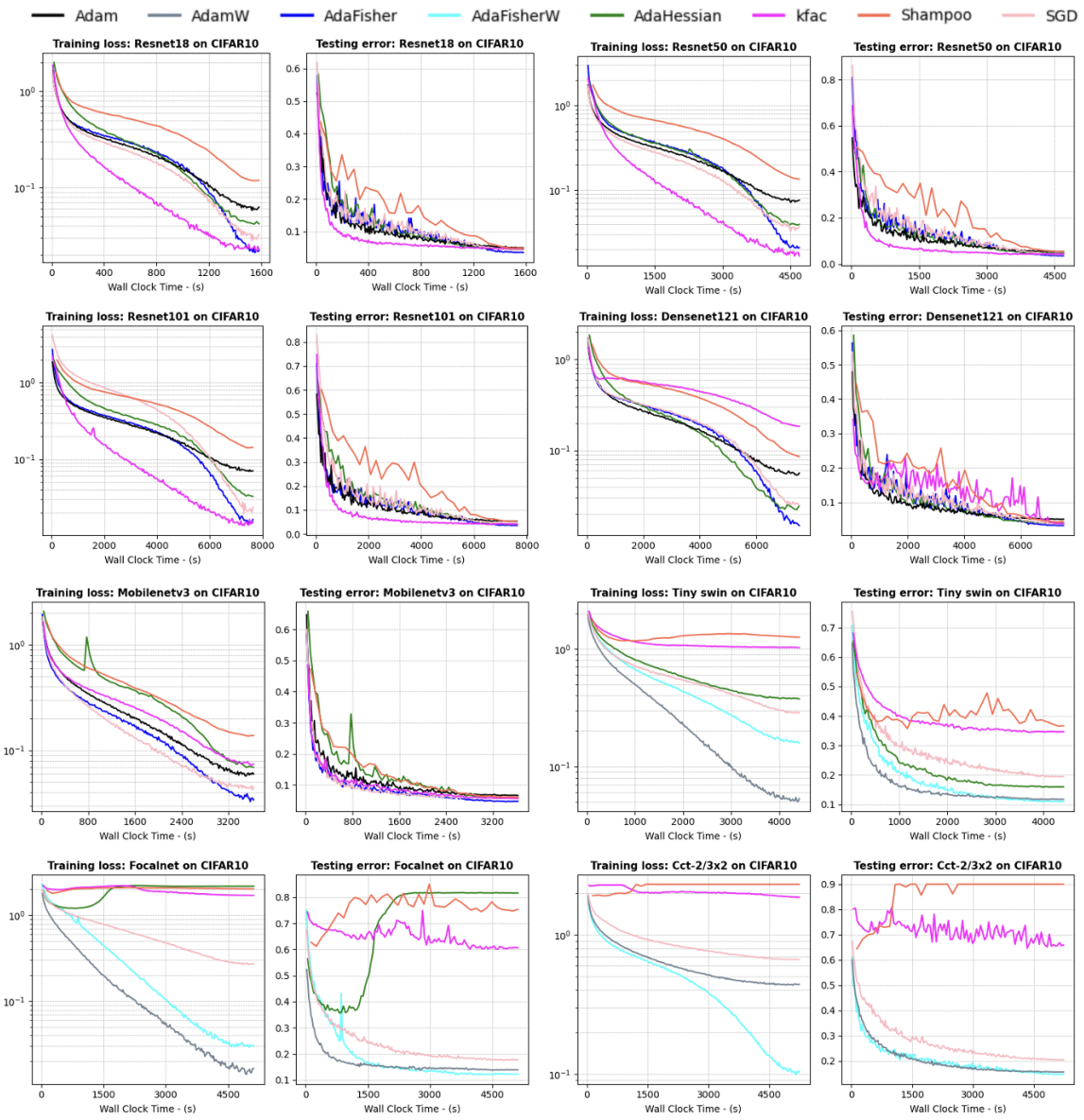

Figure C.3: WCT training loss, test error, for CNNs and ViTs on CIFAR10 experiments, with Cutout. A batch size of 256 was used, and all networks were tuned using ResNet18 applied on CIFAR10. The final accuracy results are reported in Table 5.7 (b).

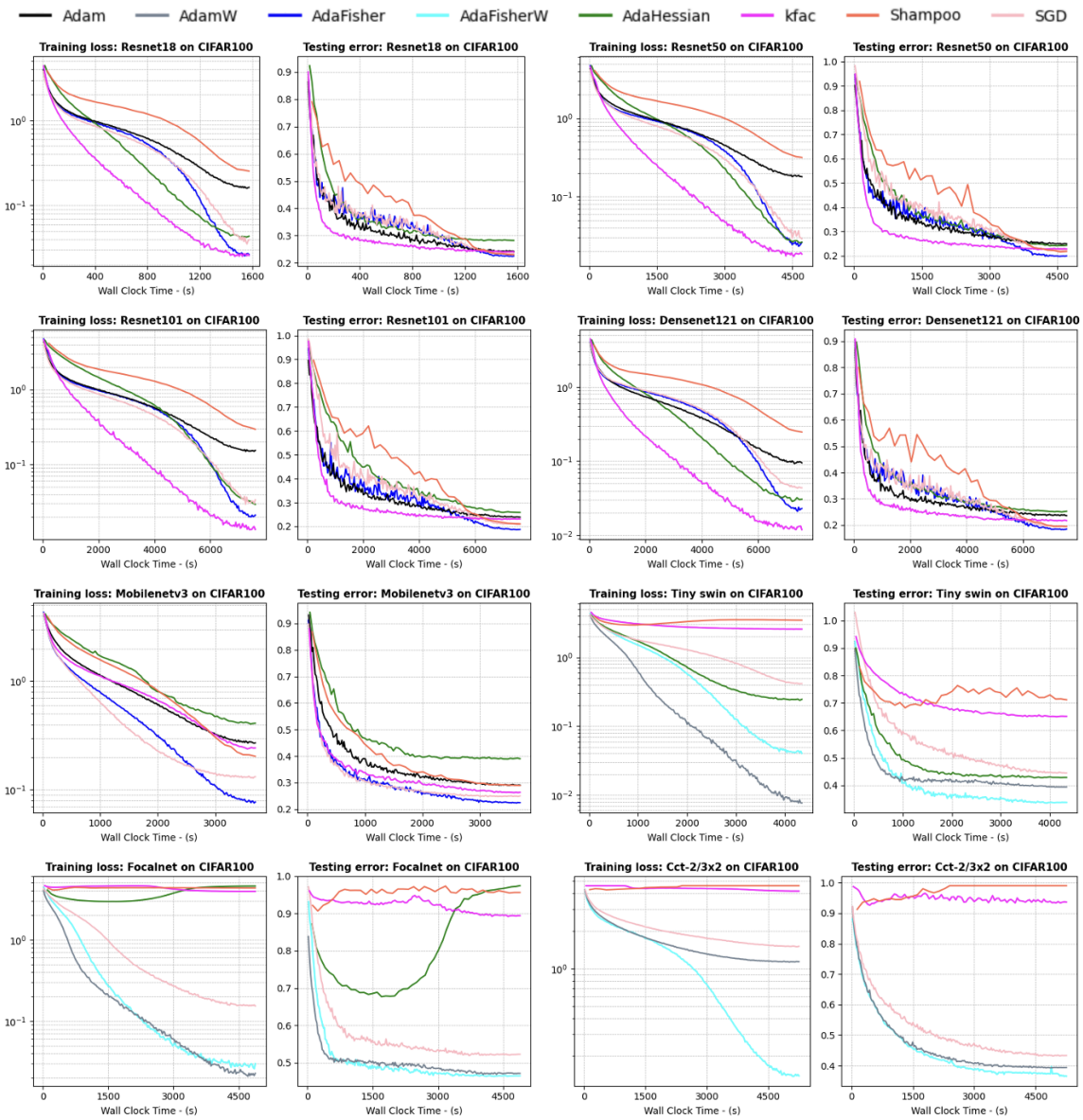

Figure C.4: WCT training loss, test error, for CNNs and ViTs on CIFAR100 experiments, with Cutout. A batch size of 256 was used, and all networks were tuned using ResNet18 applied on CIFAR10. The final accuracy results are reported in Table 5.7 (b).

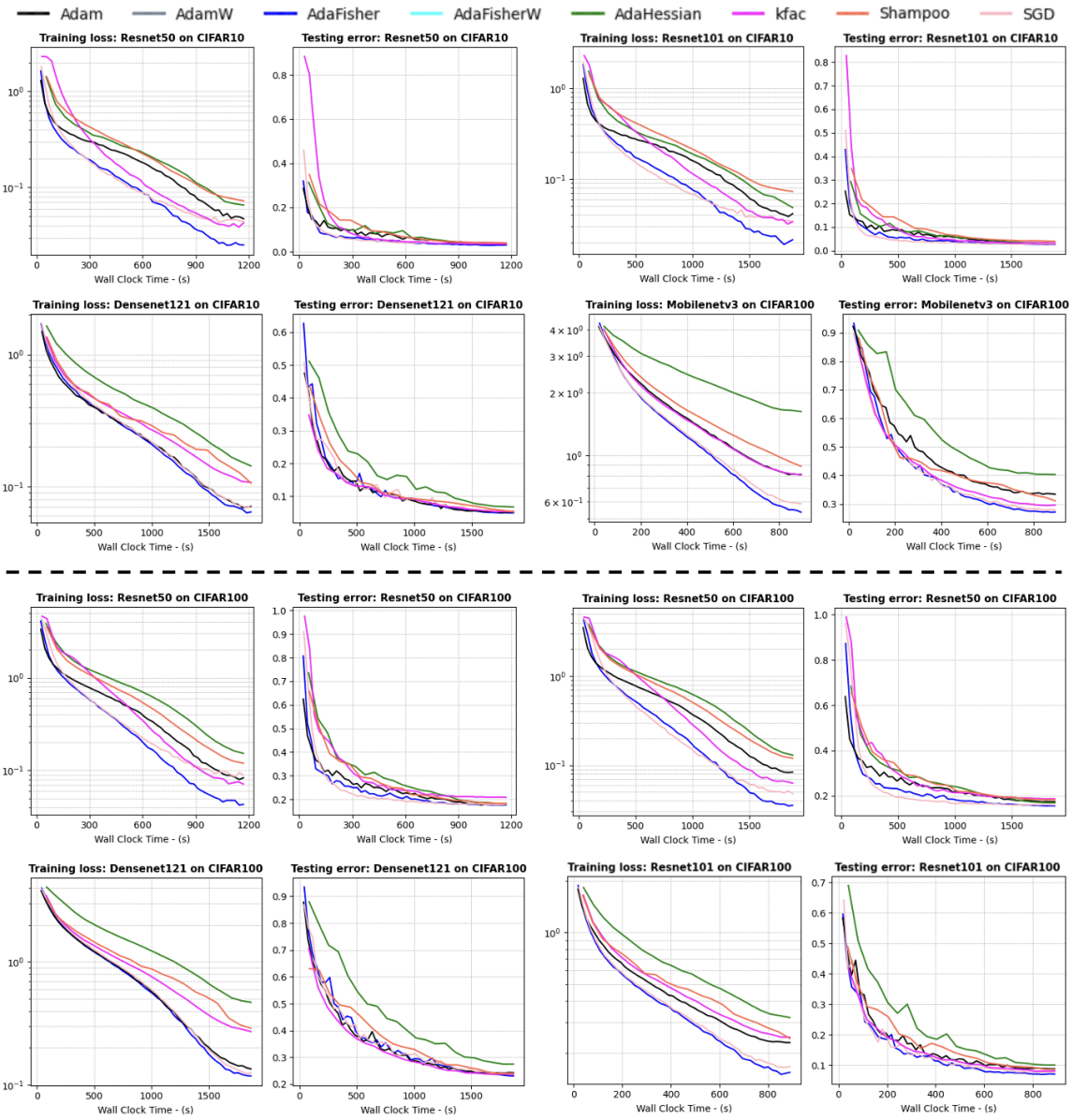

Figure C.5: WCT training loss and test error for CNNs on CIFAR-10/100 experiments with pre-trained weights on ImageNet-1k. A batch size of 256 was used, and all networks were tuned using ResNet50 applied on CIFAR10. The final accuracy results are reported in Table 5.12.

References

Al-Dujaili, A., & O'Reilly, U.-M. (2020). Sign bits are all you need for black-box attacks. In *International conference on learning representations.*

Allen-Zhu, Z., Li, Y., & Liang, Y. (2019). Learning and generalization in overparameterized neural networks, going beyond two layers. *Advances in neural information processing systems*, *32*.

Amari, S., & Nagaoka, H. (2000). Methods of information geometry.. Retrieved from https://api.semanticscholar.org/CorpusID:116976027

Amid, E., Anil, R., & Warmuth, M. (2022). Locoprop: Enhancing backprop via local loss optimization. In *International conference on artificial intelligence and statistics* (pp. 9626–9642).

Bartlett, P. L., Foster, D. J., & Telgarsky, M. J. (2017). Spectrally-normalized margin bounds for neural networks. *Advances in neural information processing systems*, *30*.

Benzing, F. (2022). Gradient descent on neurons and its link to approximate second-order optimization. In *International conference on machine learning* (pp. 1817–1853).

Bollapragada, R., & Wild, S. M. (2023). Adaptive sampling quasi-newton methods for zeroth-order stochastic optimization. *Mathematical Programming Computation*, *15*(2), 327–364.

Bolte, J., Boustany, R., Pauwels, E., & Purica, A. (2024). A second-order-like optimizer with adaptive gradient scaling for deep learning. *arXiv preprint arXiv:2410.05871*.

Botev, A., Ritter, H., & Barber, D. (2017). Practical gauss-newton optimisation for deep learning. In *International conference on machine learning* (pp. 557–565).

Boyd, S. (2004). Convex optimization. *Cambridge UP*.

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press.

Braeken, J., & Van Assen, M. A. (2017). An empirical kaiser criterion. *Psychological methods*,

22(3), 450.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... others (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877–1901.

Cha, J., Chun, S., Lee, K., Cho, H.-C., Park, S., Lee, Y., & Park, S. (2021). Swad: Domain generalization by seeking flat minima. *Advances in Neural Information Processing Systems*, *34*, 22405–22418.

Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., ... Zecchina, R. (2019). Entropy-sgd: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment*, *2019*(12), 124018.

Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., ... others (2018). {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In *13th usenix symposium on operating systems design and implementation (osdi 18)* (pp. 578–594).

Chen, X., Liang, C., Huang, D., Real, E., Wang, K., Pham, H., ... others (2024). Symbolic discovery of optimization algorithms. *Advances in neural information processing systems*, *36*.

Chen, X., Liu, S., Sun, R., & Hong, M. (2019a). On the convergence of a class of adam-type algorithms for non-convex optimization. In *International conference on learning representations.*

Chen, X., Liu, S., Sun, R., & Hong, M. (2019b). On the convergence of a class of adam-type algorithms for non-convex optimization. In *International conference on learning representations.* Retrieved from https://openreview.net/forum?id=H1x-x309tm

Chen, X., Liu, S., Xu, K., Li, X., Lin, X., Hong, M., & Cox, D. (2019). Zo-adamm: Zeroth-order adaptive momentum method for black-box optimization. *Advances in neural information processing systems*, *32*.

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., & LeCun, Y. (2015). The loss surfaces of multilayer networks. In *Artificial intelligence and statistics* (pp. 192–204).

Dauphin, Y., De Vries, H., & Bengio, Y. (2015). Equilibrated adaptive learning rates for non-convex optimization. *Advances in neural information processing systems*, *28*.

Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., & Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *Advances in neural information processing systems*, *27*.

Dedieu, J.-P. (2015). Newton-raphson method. In B. Engquist (Ed.), *Encyclopedia of applied and computational mathematics* (pp. 1023–1028). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from https://doi.org/10.1007/978-3-540-70529-1_374 doi: 10.1007/978-3-540-70529-1_374

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 ieee conference on computer vision and pattern recognition* (p. 248-255). doi: 10.1109/CVPR.2009.5206848

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, *29*(6), 141-142. doi: 10.1109/MSP.2012.2211477

Devlin, J. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

DeVries, T., & Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*.

Dinh, L., Pascanu, R., Bengio, S., & Bengio, Y. (2017). Sharp minima can generalize for deep nets. In *International conference on machine learning* (pp. 1019–1028).

Dozat, T. (2016). Incorporating nesterov momentum into adam. In *International conference on learning representations.*

Dubey, S. R., Basha, S. S., Singh, S. K., & Chaudhuri, B. B. (2022). Adainject: Injection-based adaptive gradient descent optimizers for convolutional neural networks. *IEEE Transactions on Artificial Intelligence*, *4*(6), 1540–1548.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*(61), 2121–2159. Retrieved from http://jmlr.org/papers/v12/duchi11a.html

ElNokrashy, M., AlKhamissi, B., & Diab, M. (2022). Depth-wise attention (dwatt): A layer fusion method for data-efficient classification. *arXiv preprint arXiv:2209.15168*.

Erhan, D., Courville, A., Bengio, Y., & Vincent, P. (2010). Why does unsupervised pre-training help deep learning? In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 201–208).

Eschenhagen, R., Immer, A., Turner, R., Schneider, F., & Hennig, P. (2024). Kronecker-factored approximate curvature for modern neural network architectures. *Advances in Neural Information Processing Systems*, *36*.

Foret, P., Kleiner, A., Mobahi, H., & Neyshabur, B. (2021). Sharpness-aware minimization for efficiently improving generalization. In *International conference on learning representations.* Retrieved from https://openreview.net/forum?id=6Tm1mposlrM

Fort, S., Dziugaite, G. K., Paul, M., Kharaghani, S., Roy, D. M., & Ganguli, S. (2020). Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. *Advances in Neural Information Processing Systems*, *33*, 5850–5861.

Fort, S., & Scherlis, A. (2019). The goldilocks zone: Towards better understanding of neural network loss landscapes. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 33, pp. 3574–3581).

Frankle, J., & Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International conference on learning representations.* Retrieved from https://openreview.net/forum?id=rJl-b3RcF7

Frantar, E., Kurtic, E., & Alistarh, D. (2021). M-FAC: Efficient matrix-free approximations of second-order information. In A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in neural information processing systems.* Retrieved from https://openreview.net/forum?id=EEq6YUrDyfO

F.R.S., K. P. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *2*(11), 559-572. doi: 10.1080/14786440109462720

Garipov, T., Izmailov, P., Podoprikhin, D., Vetrov, D. P., & Wilson, A. G. (2018). Loss surfaces, mode connectivity, and fast ensembling of dnns. *Advances in neural information processing systems*, *31*.

George, T. (2021, February). *NNGeometry: Easy and Fast Fisher Information Matrices and Neural Tangent Kernels in PyTorch.* Zenodo. Retrieved from https://doi.org/10.5281/zenodo.4532597 doi: 10.5281/zenodo.4532597

George, T., Laurent, C., Bouthillier, X., Ballas, N., & Vincent, P. (2018). Fast approximate natural gradient descent in a kronecker factored eigenbasis. *Advances in Neural Information Processing Systems*, *31*.

Ghadimi, S., & Lan, G. (2013). Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM journal on optimization*, *23*(4), 2341–2368.

Goldfarb, D., Ren, Y., & Bahamou, A. (2020). Practical quasi-newton methods for training deep neural networks. *Advances in Neural Information Processing Systems*, *33*, 2386–2396.

Golovin, D., Karro, J., Kochanski, G., Lee, C., Song, X., & Zhang, Q. (2020). Gradientless descent: High-dimensional zeroth-order optimization. In *International conference on learning representations.* Retrieved from https://openreview.net/forum?id=Skep6TVYDB

GOMES, D. M., Zhang, Y., Belilovsky, E., Wolf, G., & Hosseini, M. S. (2025). Adafisher: Adaptive second order optimization via fisher information. In *The thirteenth international conference on learning representations.* Retrieved from https://openreview.net/forum?id=puTxuiK2qO

Goodfellow, I. (2016). *Deep learning.* MIT press.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., ... He, K. (2017). Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.

Grosse, R., & Martens, J. (2016). A kronecker-factored approximate fisher matrix for convolution layers. In *International conference on machine learning.*

Gupta, S., Agrawal, A., Gopalakrishnan, K., & Narayanan, P. (2015). Deep learning with limited numerical precision. In *International conference on machine learning* (pp. 1737–1746).

Gupta, V., Koren, T., & Singer, Y. (2018). Shampoo: Preconditioned stochastic tensor optimization. In *International conference on machine learning* (pp. 1842–1850).

Hassani, A., Walton, S., Shah, N., Abuduweili, A., Li, J., & Shi, H. (2021). Escaping the big data paradigm with compact transformers. *arXiv preprint arXiv:2104.05704*.

He, H., Huang, G., & Yuan, Y. (2019). Asymmetric valleys: Beyond sharp and flat local minima.

Advances in neural information processing systems, *32*.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 770–778).

Heo, B., Chun, S., Oh, S. J., Han, D., Yun, S., Kim, G., ... Ha, J.-W. (2021). Adamp: Slowing down the slowdown for momentum optimizers on scale-invariant weights. In *International conference on learning representations*. Retrieved from https://openreview.net/forum?id=Iz3zU3M316D

Hinton, G., Srivastava, N., & Swersky, K. (2012). Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, *14*(8), 2.

Horn, R. A., & Johnson, C. R. (2012). *Matrix analysis* (2nd ed.). Cambridge University Press.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., ... others (2019). Searching for mobilenetv3. In *Proceedings of the ieee/cvf international conference on computer vision* (pp. 1314–1324).

Hu, Z., & Huang, H. (2023). Optimization and bayes: a trade-off for overparameterized neural networks. *Advances in Neural Information Processing Systems*, *36*, 13492–13517.

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 4700–4708).

Huang, L., et al. (2023). Normalization techniques in training dnns: Methodology, analysis and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. doi: 10.1109/TPAMI.2023.3250241

Huo, Z., Gu, B., & Huang, H. (2021). Large batch optimization for deep learning using new complete layer-wise adaptive rate scaling. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 35, pp. 7883–7890).

Ioffe, S. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

iris. (2018). IEEE Dataport. Retrieved from https://dx.doi.org/10.21227/rz7n-kj20 doi: 10.21227/rz7n-kj20

Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., & Wilson, A. G. (2018). Averaging weights leads to wider optima and better generalization. In *34th conference on uncertainty in artificial intelligence 2018, uai 2018* (pp. 876–885).

Jacot, A., Gabriel, F., & Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, *31*.

Jastrzebski, S., Szymczak, M., Fort, S., Arpit, D., Tabor, J., Cho*, K., & Geras*, K. (2020). The break-even point on optimization trajectories of deep neural networks. In *International conference on learning representations.* Retrieved from https://openreview.net/forum?id=r1g87C4KwB

Jiang, Z., Gu, J., Zhu, H., & Pan, D. (2024). Pre-rmsnorm and pre-crmsnorm transformers: equivalent and efficient pre-ln transformers. *Advances in Neural Information Processing Systems*, *36*.

Jin, C., Ge, R., Netrapalli, P., Kakade, S. M., & Jordan, M. I. (2017). How to escape saddle points efficiently. In *International conference on machine learning* (pp. 1724–1732).

Kalra, D. S., & Barkeshli, M. (2023). Phase diagram of early training dynamics in deep neural networks: effect of the learning rate, depth, and width. In *Thirty-seventh conference on neural information processing systems.* Retrieved from https://openreview.net/forum?id=Al9yglQGKj

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2017). On large-batch training for deep learning: Generalization gap and sharp minima. In *International conference on learning representations.* Retrieved from https://openreview.net/pdf?id=H1oyRlYgg

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In *International conference on learning representations.* Retrieved from http://arxiv.org/abs/1412.6980

Koller, D. (2009). *Probabilistic graphical models: Principles and techniques.* The MIT Press.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.

Kunstner, F., Hennig, P., & Balles, L. (2019). Limitations of the empirical fisher approximation for natural gradient descent. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2019/file/46a558d97954d0692411c861cf78ef79-Paper.pdf

Larson, J., Menickelly, M., & Wild, S. M. (2019). Derivative-free optimization methods. *Acta Numerica*, *28*, 287–404.

Le, Y., & Yang, X. (2015). Tiny imagenet visual recognition challenge. *CS 231N*, *7*(7), 3.

Lecun, Y. (2001, 08). A theoretical framework for back-propagation.

LeCun, Y., Bottou, L., Orr, G., & Müller, K. (2012). Efficient backprop. In *Neural networks* (pp. 9–48). Springer Verlag. (Copyright: Copyright 2021 Elsevier B.V., All rights reserved.) doi: 10.1007/978-3-642-35289-8_3

Lei Ba, J., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *ArXiv e-prints*, arXiv–1607.

Lian, X., Zhang, H., Hsieh, C.-J., Huang, Y., & Liu, J. (2016). A comprehensive linear speedup analysis for asynchronous stochastic parallel optimization from zeroth-order to first-order. *Advances in Neural Information Processing Systems*, *29*.

Lin, W., Dangel, F., Eschenhagen, R., Bae, J., Turner, R. E., & Makhzani, A. (2024). Can we remove the square-root in adaptive gradient methods? a second-order perspective. In *Forty-first international conference on machine learning*. Retrieved from https://openreview.net/forum?id=vuMD71R20q

Liu, D. C., & Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical programming*, *45*(1), 503–528.

Liu, H., Li, Z., Hall, D. L. W., Liang, P., & Ma, T. (2024). Sophia: A scalable stochastic second-order optimizer for language model pre-training. In *The twelfth international conference on learning representations*. Retrieved from https://openreview.net/forum?id=3xHDeA8Noi

Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., & Han, J. (2020). On the variance of the adaptive learning rate and beyond. In *International conference on learning representations*.

Retrieved from https://openreview.net/forum?id=rkgz2aEKDr

Liu, S., Kailkhura, B., Chen, P.-Y., Ting, P., Chang, S., & Amini, L. (2018). Zeroth-order stochastic variance reduction for nonconvex optimization. *Advances in Neural Information Processing Systems*, *31*.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., ... Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the ieee/cvf international conference on computer vision* (pp. 10012–10022).

Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization. In *International conference on learning representations*. Retrieved from https://openreview.net/forum?id=Bkg6RiCqY7

Lucas, J. (2022). *Optimization and loss landscape geometry of deep learning* (Unpublished doctoral dissertation). University of Toronto (Canada).

Luo, L., Xiong, Y., & Liu, Y. (2019). Adaptive gradient methods with dynamic bound of learning rate. In *International conference on learning representations*. Retrieved from https://openreview.net/forum?id=Bkg3g2R9FX

Ma, X. (2020). Apollo: An adaptive parameter-wise diagonal quasi-newton method for nonconvex stochastic optimization. *arXiv preprint arXiv:2009.13586*.

Malladi, S., Gao, T., Nichani, E., Damian, A., Lee, J. D., Chen, D., & Arora, S. (2023). Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems*, *36*, 53038–53075.

Marcus, M. P., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, *19*(2), 313–330. Retrieved from https://aclanthology.org/J93-2004

Martens, J. (2020). New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, *21*(146), 1–76.

Martens, J., & Grosse, R. (2015a). Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning* (pp. 2408–2417).

Martens, J., & Grosse, R. (2015b, 07–09 Jul). Optimizing neural networks with kronecker-factored approximate curvature. In F. Bach & D. Blei (Eds.), *Proceedings of the 32nd international*

conference on machine learning (Vol. 37, pp. 2408–2417). Lille, France: PMLR. Retrieved from https://proceedings.mlr.press/v37/martens15.html

Martens, J., et al. (2010). Deep learning via hessian-free optimization. In *Icml* (Vol. 27, pp. 735–742).

Merity, S., Xiong, C., Bradbury, J., & Socher, R. (2017). Pointer sentinel mixture models. In *International conference on learning representations*. Retrieved from https://openreview.net/forum?id=Byj72udxe

Misra, J., & Saha, I. (2010). Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing*, *74*(1-3), 239–255.

Mozaffari, M., Li, S., Zhang, Z., & Dehnavi, M. M. (2023). MKOR: Momentum-enabled kronecker-factor-based optimizer using rank-1 updates. In *Thirty-seventh conference on neural information processing systems*. Retrieved from https://openreview.net/forum?id=jcnvDO96N5

Nesterov, Y. (1983). A method for solving the convex programming problem with convergence rate o (1/k2). In *Dokl akad nauk sssr* (Vol. 269, p. 543).

Nocedal, J., & Wright, S. J. (1999). *Numerical optimization*. Springer.

Osawa, K., Ishikawa, S., Yokota, R., Li, S., & Hoefler, T. (2023). *Asdl: A unified interface for gradient preconditioning in pytorch.*

Park, H., Amari, S.-I., & Fukumizu, K. (2000). Adaptive natural gradient learning algorithms for various stochastic models. *Neural Networks*, *13*(7), 755-764. Retrieved from https://www.sciencedirect.com/science/article/pii/S0893608000000514 doi: https://doi.org/10.1016/S0893-6080(00)00051-4

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... others (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, *32*.

Patro, S., & Sahu, K. K. (2015). Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, *4*(5), 1–17.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners.

Rando, M., Molinari, C., Rosasco, L., & Villa, S. (2024). An optimal structured zeroth-order algorithm for non-smooth optimization. *Advances in Neural Information Processing Systems*, *36*.

Reddi, S. J., Kale, S., & Kumar, S. (2018). On the convergence of adam and beyond. In *International conference on learning representations*. Retrieved from https://openreview.net/forum?id=ryQu7f-RZ

Rotem, N., Fix, J., Abdulrasool, S., Catron, G., Deng, S., Dzhabarov, R., ... others (2018). Glow: Graph lowering compiler techniques for neural networks. *arXiv preprint arXiv:1805.00907*.

Ryu, E., & Boyd, S. (2016, 01). A primer on monotone operator methods survey. *Applied and computational mathematics*, *15*, 3-43.

Schaul, T., Zhang, S., & LeCun, Y. (2013). No more pesky learning rates. In *International conference on machine learning* (pp. 343–351).

Shazeer, N., & Stern, M. (2018). Adafactor: Adaptive learning rates with sublinear memory cost. In *International conference on machine learning* (pp. 4596–4604).

Shu, Y., Dai, Z., Sng, W., Verma, A., Jaillet, P., & Low, B. K. H. (2023). Zeroth-order optimization with trajectory-informed derivative estimation. In *The eleventh international conference on learning representations*. Retrieved from https://openreview.net/forum?id=n1bLgxHW6jW

Sivan, H., Gabel, M., & Schuster, A. (2024). FOSI: Hybrid first and second order optimization. In *The twelfth international conference on learning representations*. Retrieved from https://openreview.net/forum?id=NvbeD9Ttkx

Snell, J., Swersky, K., & Zemel, R. (2017). Prototypical networks for few-shot learning. *Advances in neural information processing systems*, *30*.

Soudry, D., Hoffer, E., Nacson, M. S., Gunasekar, S., & Srebro, N. (2018). The implicit bias of gradient descent on separable data. *Journal of Machine Learning Research*, *19*(70), 1–57.

Sun, H., Shen, L., Zhong, Q., Ding, L., Chen, S., Sun, J., ... Tao, D. (2024). Adasam: Boosting sharpness-aware minimization with adaptive learning rate and momentum for training deep

neural networks. *Neural Networks*, *169*, 506–519.

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013a). On the importance of initialization and momentum in deep learning. In *International conference on machine learning* (pp. 1139–1147).

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013b, 17–19 Jun). On the importance of initialization and momentum in deep learning. In S. Dasgupta & D. McAllester (Eds.), *Proceedings of the 30th international conference on machine learning* (Vol. 28, pp. 1139–1147). Atlanta, Georgia, USA: PMLR. Retrieved from https://proceedings.mlr.press/v28/sutskever13.html

Takahashi, R., Matsubara, T., & Uehara, K. (2020, September). Data augmentation using random image cropping and patching for deep cnns. *IEEE Transactions on Circuits and Systems for Video Technology*, *30*(9), 2917–2931. Retrieved from http://dx.doi.org/10.1109/TCSVT.2019.2935128 doi: 10.1109/tcsvt.2019.2935128

Tang, Z., Jiang, F., Gong, M., Li, H., Wu, Y., Yu, F., ... Wang, M. (2021). Skfac: Training neural networks with faster kronecker-factored approximate curvature. In *2021 ieee/cvf conference on computer vision and pattern recognition (cvpr)* (p. 13474-13482). doi: 10.1109/CVPR46437.2021.01327

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In I. Guyon et al. (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Vyas, N., Morwani, D., Zhao, R., Shapira, I., Brandfonbrener, D., Janson, L., & Kakade, S. M. (2025). SOAP: Improving and stabilizing shampoo using adam. In *The thirteenth international conference on learning representations.* Retrieved from https://openreview.net/forum?id=IDxZhXrpNf

Vysogorets, A. M., Dawid, A., & Kempe, J. (2024). Deconstructing the goldilocks zone of neural network initialization. In *Forty-first international conference on machine learning.* Retrieved from https://openreview.net/forum?id=DJXt63RLO1

Wang, S., Zhou, P., Li, J., & Huang, H. (2024). 4-bit shampoo for memory-efficient network training. In *The thirty-eighth annual conference on neural information processing systems.* Retrieved from https://openreview.net/forum?id=ASqdVeifn7

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., & Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. *Advances in neural information processing systems*, *30*.

Xie, Z., Sato, I., & Sugiyama, M. (2021). A diffusion theory for deep learning dynamics: Stochastic gradient descent exponentially favors flat minima. In *International conference on learning representations.* Retrieved from https://openreview.net/forum?id=wXgk_iCiYGo

Xie, Z., Wang, X., Zhang, H., Sato, I., & Sugiyama, M. (2022). Adaptive inertia: Disentangling the effects of adaptive learning rate and momentum. In *International conference on machine learning* (pp. 24430–24459).

Xie, Z., zhiqiang xu, Zhang, J., Sato, I., & Sugiyama, M. (2023). On the overlooked pitfalls of weight decay and how to mitigate them: A gradient-norm perspective. In *Thirty-seventh conference on neural information processing systems.* Retrieved from https://openreview.net/forum?id=vnGcubtzR1

Yang, J., Li, C., Dai, X., & Gao, J. (2022). Focal modulation networks. *Advances in Neural Information Processing Systems*, *35*, 4203–4217.

Yao, Z., Gholami, A., Shen, S., Mustafa, M., Keutzer, K., & Mahoney, M. (2021). Adahessian: An adaptive second order optimizer for machine learning. In *proceedings of the aaai conference on artificial intelligence* (Vol. 35, pp. 10665–10673).

yeh Chiang, P., Ni, R., Miller, D. Y., Bansal, A., Geiping, J., Goldblum, M., & Goldstein, T. (2023). Loss landscapes are all you need: Neural network generalization can be explained without the implicit bias of gradient descent. In *The eleventh international conference on learning representations.* Retrieved from https://openreview.net/forum?id=QC10RmRbZy9

You, Y., Gitman, I., & Ginsburg, B. (2017). *Large batch training of convolutional networks.*

You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., ... Hsieh, C.-J. (2020). Large batch optimization for deep learning: Training bert in 76 minutes. In *International conference*

on learning representations. Retrieved from https://openreview.net/forum?id=Syx4wnEtvH

Yu, Y., Xia, R., Ma, Q., Lengyel, M., & Hennequin, G. (2024). Second-order forward-mode optimization of recurrent neural networks for neuroscience. In *The thirty-eighth annual conference on neural information processing systems.* Retrieved from https://openreview.net/forum?id=Pox8jNQOo5

Zaheer, M., Reddi, S., Sachan, D., Kale, S., & Kumar, S. (2018). Adaptive methods for nonconvex optimization. *Advances in neural information processing systems*, *31*.

Zhang, L., Shi, S., & Li, B. (2023). Eva: Practical second-order optimization with kronecker-vectorized approximation. In *The eleventh international conference on learning representations.* Retrieved from https://openreview.net/forum?id=_Mic8V96Voy

Zhang, Y., Yao, Y., Jia, J., Yi, J., Hong, M., Chang, S., & Liu, S. (2022). How to robustify black-box ML models? a zeroth-order optimization perspective. In *International conference on learning representations.* Retrieved from https://openreview.net/forum?id=W9G_ImpHlQd

Zhang, Z., & Sabuncu, M. (2018). Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems*, *31*.

Zhao, J., Zhang, Z., Chen, B., Wang, Z., Anandkumar, A., & Tian, Y. (2024). Galore: Memory-efficient LLM training by gradient low-rank projection. In *Forty-first international conference on machine learning.* Retrieved from https://openreview.net/forum?id=hYHsrKDiX7

Zhao, Y., Dang, S., Ye, H., Dai, G., Qian, Y., & Tsang, I. (2025). Second-order fine-tuning without pain for LLMs: A hessian informed zeroth-order optimizer. In *The thirteenth international conference on learning representations.* Retrieved from https://openreview.net/forum?id=bEqI61iBue

Zhou, P., Xie, X., & YAN, S. (2022). Win: Weight-decay-integrated nesterov acceleration for adaptive gradient algorithms. In *Has it trained yet? neurips 2022 workshop.* Retrieved from https://openreview.net/forum?id=dNK2bw4y0R

Zhuang, J., Gong, B., Yuan, L., Cui, Y., Adam, H., Dvornek, N. C., … Liu, T. (2022). Surrogate gap minimization improves sharpness-aware training. In *International conference on learning representations*. Retrieved from https://openreview.net/forum?id=edONMAnhLu-

Zhuang, J., Tang, T., Ding, Y., Tatikonda, S. C., Dvornek, N., Papademetris, X., & Duncan, J. (2020). Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *Advances in neural information processing systems*, *33*, 18795–18806.