Yale University

EliScholar – A Digital Platform for Scholarly Publishing at Yale

Yale Graduate School of Arts and Sciences Dissertations

Spring 2022

Machine Learning Methods to Estimate Whole-Brain Effective Connectome for ASD Identification

Juntang Zhuang
Yale University Graduate School of Arts and Sciences, zhuangjt12@gmail.com

Follow this and additional works at: https://elischolar.library.yale.edu/gsas_dissertations

Recommended Citation

Zhuang, Juntang, "Machine Learning Methods to Estimate Whole-Brain Effective Connectome for ASD Identification" (2022). *Yale Graduate School of Arts and Sciences Dissertations*. 691.
https://elischolar.library.yale.edu/gsas_dissertations/691

This Dissertation is brought to you for free and open access by EliScholar – A Digital Platform for Scholarly Publishing at Yale. It has been accepted for inclusion in Yale Graduate School of Arts and Sciences Dissertations by an authorized administrator of EliScholar – A Digital Platform for Scholarly Publishing at Yale. For more information, please contact elischolar@yale.edu.

Abstract

Machine Learning Methods to Estimate Whole-brain Effective Connectome for ASD Identification

Juntang Zhuang

2022

Functional Magnetic Resonance Imaging (fMRI) is widely used to study neural-developmental diseases such as Autism Spectrum Disorder (ASD). There are mainly two types of connectome to analyze fMRI: the Functional Connectome (FC) and the Effective Connectome (EC). FC is typically derived as the correlation between fMRI time-series from different brain regions, while EC is derived by fitting the measurement time-series to the Dynamical Causal Model (DCM) described by a system of Ordinary Differential Equations (ODEs). FC is typically easier to compute yet can not reveal the causal relations among brain regions; EC reveals the causal relations yet is much harder to compute and is more sensitive to observation noise. Therefore, this dissertation aims to propose a generic framework for estimation of EC, and identify ASD from fMRI based on EC.

First, we propose the **M**odel **D**riven **L**earning Framework (MDL) for parameter estimation in the continuous models. MDL iteratively performs three steps: 1) forward simulation according to prior knowledge of the model, 2) backward pass to derive the gradient of parameters, 3) update of parameters based on gradient information.

We derive various methods to solve each step in MDL. Specifically, for step 2), we identify the inaccuracy of existing gradient estimation methods for continuous time models (e.g. ODEs): the *adjoint* method has numerical errors in reverse-mode integration; the *naive* method suffers from a redundantly deep computation graph. We propose a series of new methods which guarantee the numerical accuracy with a low memory cost. For step 3), we propose the AdaBelief optimizer, which is a generic first-order adaptive op-

timizer that simultaneously achieves fast convergence, good generalization and training stability. Furthermore, we show that an asynchronous version of AdaBelief achieves provably weaker convergence condition and faster convergence rate. We show that our MDL significantly accelerates the fitting of DCM and estimation of EC.

To deal with the limited data and improve generalization of the classifier, we propose the **S**urrogate **G**ap Guided **S**harpness-**A**ware **M**inimization (GSAM). GSAM is based on the observation that poor generalization often comes with a sharp loss surface of the model, and improves generalization by jointly minimizing the training loss and the curvature of the loss surface.

Finally, we apply the proposed MDL to estimate whole-brain EC for fMRI, and performed group comparison to identify FC and EC edges that are related to ASD. Next, we apply the estimated EC for the identification of ASD. Specifically, we conducted experiments with both resting-state fMRI and task fMRI data, and compare the predictive power of FC and EC in both cases. Furthermore, we apply GSAM to further improve the generalization performance, which significantly improves the classification performance and reduces the dominant eigenvalue of the Hessian of the network. In summary, we apply the proposed framework for effective connectome analysis, and improve the identification of ASD from fMRI data.

Machine Learning Methods to Estimate Whole-brain Effective Connectome for ASD Identification

A Dissertation
Presented to the Faculty of the Graduate School
of
Yale University
in Candidacy for the Degree of
Doctor of Philosophy

by
Juntang Zhuang

Dissertation Director: James S. Duncan

May, 2022

Copyright © 2022 by Juntang Zhuang

All rights reserved.

Acknowledgments

I would like to give my warmest thanks to my advisor, Professor James S. Duncan, for his patient guidance and strong support through all these years. It has been a great honor for me to work together with Jim on the exciting research projects, and I'm fortunate to learn from Jim not only the research techniques, but more importantly how to choose interesting research topics and work as an independent researcher. Conversations with Jim were truly encouraging and inspiring me to think out of the box, and I would not finish my PhD without the strongest support from Jim.

I would also like to thank Nicha C. Dvornek, who has been extremely kind and helpful. Nicha has helped with discussion and draft revision for almost all my papers and proposals since I joined this fantastic lab, and her help is vital during my career path as a PhD candidate.

I want to give my warmest thanks to Professor Sekhar Tatikonda, Professor Xenophon Papademetris, Professor Hemant Tagare, Professor Larry Staib, Professor Pamela Ventola, Professor John Onofrey and Xilin Shen. You have helped me so much and are always open to discussion with me and provide guidance on both research and life. I'm so grateful that I can learn from you.

I would like to thank Professor Tianyu Ma, who led me into the journey of research when I was an undergrad. I want to thank Professor Chi Liu and Richard E Carson, without your help I would not have the chance to start my PhD career at Yale. I also want to thank Chung Jan Chan, Jing Wu, Hui Liu, Silin Ren, Peng Fan, Yan Xia, Luyao Shi and Yihuan Lu, who gave so much warm help when I first entered the gate of research.

I want to thank my groups members – Francisca Melina Tibo, Zach Augenfeld, Fan Zhang, Allen Lu, Nripesh Parajuli, Xiaoxiao Li, Junlin Yang, Shawn Ahn, Zhao Liu, Yuan Zhou, Siyuan Dong, Daniel Pak, John Treilhard, Kevinminh Ta, Chenyu You, Jiyao Wang, Xiaoran Zhang and Tommy Tang. You have made my PhD life very colorful and

interesting.

I would like to thank Ting Liu, Boqing Gong, Liangzhe Yuan and Yin Cui for your kind help during my internship at Google. I also want to thank my friends at Yale, Cong Shen, Han Liu, Siyuan Gao, Xuechen Zhou, Xinxin Nie, Shaojie Ma, and the list goes on.

I want to give special thanks to Professor Chak Wong, who has encouraged me to pursue challenging goals and has influenced my way of thinking. I would like to thank Xiaolin Guo, who has been a supportive friend for me since undergrad and is always open to discussion and help.

I want to thank my parents for your firm support throughout my life, without you I would not have arrived at Yale starting from a small village. Finally, I would like to thank my wife Yifan Ding, your trust and love is the motivation for all the proactive changes I pursued, and I'm looking forward to all the beautiful moments for us in the future.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Autism Spectrum Disorder

Autism spectrum disorder (ASD) is a neurodevelopmental disorder that changes both structure and function of the brain, and affects both social behavior and mental health [107]. The core signs of ASD can be classified into two categories: persistent deficits in social communication and interaction, and persistent and repetitive patterns of behavior and interests [89]. The first sign include symptoms such as abnormal social approaches, deficits in non-verbal communications and difficulties in understanding relationships. The second sign include symptoms such as simple motor stereotypes and insistence on sameness. According to the data by the Center of Disease Control and Prevention (CDC), the prevalence of ASD in the United States has risen from 1 out of 150 in the year 2000, to 1 out of 44 in the year 2018. Despite the prevalence and severeness of ASD, there are no effective treatments for the core signs of ASD. Therefore, it's crucial to identify the cause and develop reliable identification methods for ASD [119, 40].

1.2 Introduction to fMRI

ASD is typically diagnosed with behavioral tests, and recently functional magnetic resonance imaging (fMRI) has been applied to analyze the cause of ASD [34]. fMRI measures brain activity by detecting changes associated with blood flow [62]. Specifically, the primary form of fMRI uses the blood-oxygen-level dependent (BOLD) contrast [17], which is related to neural activity and reflects the changes in regional cerebral blood flow, volume and oxygen level. fMRI has a wide range of translational applications [95]. fMRI has been applied together with MRI to guide neurosurgery via the functional-anatomical localization [2]. Intraoperative fMRI has been applied to monitor the brain simulation for Parkinson's disease [55]. Diffusion fMRI has been applied to identify drug action [38], the pre-symptomatic diagnosis of disease such as Alzheimer's disease [1], Huntinton's disease [116], schizophrenia [102] and ASD [30]. In this thesis, we mainly focus on the application of fMRI in early diagnosis of ASD.

fMRI can be broadly categorized into resting-state fMRI and task fMRI. Resting-state fMRI is acquired when the subject is scanned under no stimuli. resting-state fMRI has been applied in large-scale projects such as the Human Connectome Project (HCP) [157] and the ABIDE Project [34]. Most resting-state fMRI image are acquired with a spatial resolution of 1mm to 2mm at a time resolution of 1s to 2s between frames. Task-fMRI is acquired when the subject is performing certain tasks [6], such as the Monetary Incentive Delay Task [76], the Stop-Signal Task [80] and the Emotional N-back Task [11]. The purpose of task fMRI is to correlate different brain activity patterns with tasks, for example, active regions during a social-related task might be associated with neurodevelopmental diseases such as ASD.

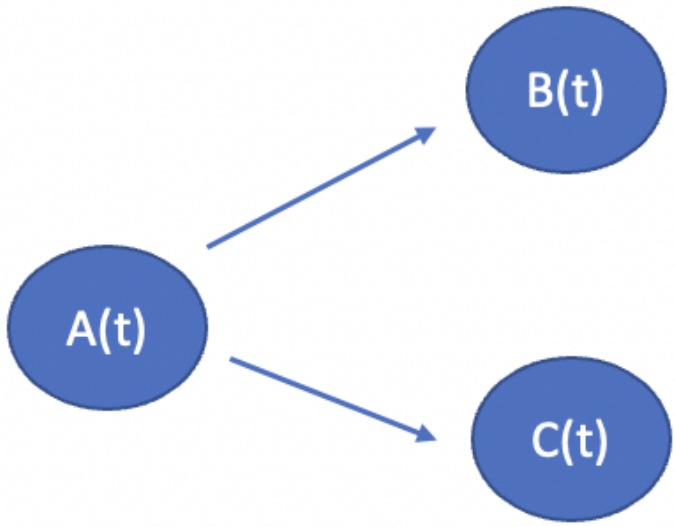


Figure 1.1: A toy model with directional edge among nodes.

1.3 Functional connectome analysis of fMRI

Researchers have studied the fMRI data from the perspectives of time-series analysis [40], activation pattern [33] and connectome among different brain regions [16]. Connectome analysis in fMRI aims to elucidate neural connections in the brain and can be generally categorized into two types: the functional connectome (FC) [156] and the effective connectome (EC) [43]. FC typically calculates the correlation between time-series of different regions-of-interest (ROIs) in the brain, which is typically robust and easy to compute; however, FC is descriptive and does not reveal the underlying dynamics. FC typically calculates the "static" connectome using the entire time-series. The recently proposed dynamic functional connectome (dFC) [122] calculates the connectome for the sliding-window partition of time-series. Studies show that dFC can capture the change of brain connectome varying with tasks during scan [50].

Despite the wide application of FC (and dFC), FC has an inherent drawback: the edge

estimated from FC is forced to be symmetric. As shown in Fig. 1.1, we consider a 3-node system, where $A(t)$ is the driving node that proactively evolves with time, $B(t)$ and $C(t)$ just follow $A(t)$. Due to the symmetry of FC, the edge from A to B is the same as the edge from B to A (e.g. $Correlation(A, B) = Correlation(B, A)$), hence FC can not tell whether A or B is the driving node. Furthermore, in this simple example, we have $Correlation(B, C) \neq 0$ even if node B and node C do not have direct interaction. In the next section, we introduce the Dynamic Causal Modeling (DCM), which models the numerical and directional interactions among nodes, hence can overcome the inherent limitations of FC.

1.4 Effective connectome and dynamic causal modeling

The effective connectome (EC) has a underlying model, which describes the directional interactions among nodes with a system of ordinary differential equations. Because the underlying dynamical system is directional, EC overcomes the disadvantages caused by the symmetry limitations with FC. Therefore, EC is a model-driven method that aims to reveal the underlying dynamics of the system.

EC is typically estimated from the Dynamic Causal Modeling (DCM) [43], which models the directed influence between ROIs, and is widely used in analysis of EEG [73] and fMRI [142]. DCM is typically a Bayesian model comparison procedure which compares the observation with a generative model, and the underlying model is often assumed to be a system of ordinary differential equations (ODEs). Although the DCM can model both linear and non-linear models in theory, in practice DCM is often restricted to low-dimensional linear ODEs due to computation complexity.

Parameter estimation in DCM is often performed by the Expectation-Maximization (EM) Algorithm [103]. However, conventional DCM with EM typically requires $O(n^2)$

memory where n is the number of parameters to estimate, and the exact algorithm needs to be re-derived for different models, which limits the application of DCM to high-dimensional non-linear dynamical systems. To solve these problems, we aim to propose a generic framework for parameter estimation in high-dimensional non-linear dynamical systems.

1.5 Summary of contributions

We propose the Model-Driven Learning Framework (MDL) in Chapter 2, which is a generic framework for parameter estimation in high-dimensional non-linear dynamical systems. MDL iteratively performs three steps iteratively: 1) forward simulation according to the model, 2) gradient estimation of parameters for continuous-time dynamical systems, 3) update of parameters based on gradient. Step 1) is easy and can be performed by any existing numerical ODE solvers (e.g. the Runge-Kutta family) [18]. We develop a series of methods to for step 2) and 3), as summarized below.

Step 2). Gradient descent for explicit models and discrete-layer neural networks is easy; however, for implicit models whose forward-pass is defined in an integration form (e.g. solution to ODE), derivation of the gradient requires much effort. In step 2) we consider both the theory and numerical implementation to derive the gradient in an ODE. In Chapter 3, we study the numerical issues with current numerical methods for gradient estimation in ODE, and identify that the mismatch between forward-time and reverse-time trajectories (caused by inevitable errors in numerical integration) causes the error in gradient estimation. We propose a series of methods to achieve accuracy at a constant memory cost, and validate the proposed methods in both benchmark deep learning tasks and DCM for fMRI data.

For step 3), we propose AdaBelief in Chapter 4, a first-order gradient optimizer that achieves fast convergence, good generalization and training stability. We demonstrate that

AdaBelief consistently improves performance for deep learning models, and accelerates the fitting of DCM at a twice faster speed compared to existing methods such as Adam. We further demonstrate that an asynchronous version of AdaBelief achieves a provably weaker convergence condition and a faster convergence rate.

In Chapter 5, we propose a generic training scheme to improve the generalization performance. Specifically, we define an equivalent measure of sharpness of the loss landscape, and jointly minimize the training loss and the sharpness of loss landscape. Since a flat loss surface is typically associated with a better generalization, out training scheme searches for a solution with both a low training loss and a good generalization. We validate our proposed method in both deep learning task, as well as ASD classification tasks with both resting-state fMRI and task fMRI.

In Chapter 6, we apply the proposed MDL to estimate whole-brain EC for fMRI, and performed group comparison to identify FC and EC edges that are related to ASD. Next, we apply the estimated EC for the identification of ASD. Specifically, we conducted experiments with both resting-state fMRI and task fMRI data, and compare the predictive power of FC and EC in both cases. Furthermore, we apply the method in Chapter 5 to further improve the generalization performance, which significantly improves the classification performance.

Chapter 2

Overview of Dynamic Causal Modeling and Model-Driven Learning Framework

2.1 Dynamic Causal Modeling

The Effective Connectome is typically estimated from the dynamical causal modeling (DCM) [43]. Suppose there are p nodes (ROIs) and denote the observed fMRI time-series signal as $s(t)$, which is a p-dimensional vector at each time t. Denote the hidden neuronal state as $z(t)$; then $z(t)$ and $s(t)$ are p-dimensional vectors for each time point t. Denote the hemodynamic response function (HRF) [87] as $h(t)$, and denote the external stimulation as $u(t)$, which is an n-dimensional vector for each t. The model is:

$$f\Big([z(t)\ \ D(t)]\Big) = \begin{bmatrix} dz(t)/dt \\ dD(t)/dt \end{bmatrix} = \begin{bmatrix} D(t)z(t) + Cu(t) \\ Bu(t) \end{bmatrix}, \ \ D(0) = A \tag{2.1}$$

$$s(t) = \Big(z(t) + \epsilon(t)\Big) * h(t), \ \ \widetilde{z(t)} = z(t) + \epsilon(t) = Deconv\Big(s(t), h(t)\Big) \tag{2.2}$$

where $\epsilon(t)$ is the noise at time t, which is assumed to follow an independent Gaussian distribution, and $*$ represents convolution operation. $D(t)$ is a $p \times p$ matrix for each t, representing the effective connectome between nodes. A is a matrix of shape $p \times p$,

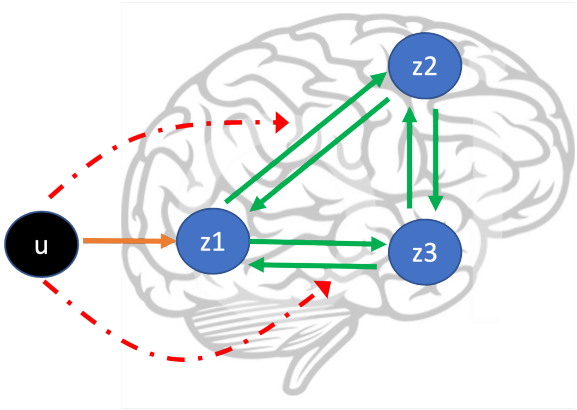


Figure 2.1: Scheme of the Dynamic Causal Modeling (DCM). For simplicity we only consider a 3-node system. For whole-brain fMRI DCM, the number of nodes equals the number of parcellated regions (e.g. 100 to 200).

representing the interaction between ROIs. B is a tensor of shape $p \times p \times n$, representing the effect of stimulation on the effective connectome. C is a matrix of shape $p \times n$, representing the effect of stimulation on neuronal state. An example of $n = 1, p = 3$ is shown in Fig. 2.1. The task is to estimate parameters A, B, C (and infer $D(t)$) from noisy observation $s(t)$. Eq. 6.1 assumes the connection among nodes vary with time, which is a common assumption for fMRI data acquired under different tasks during the scan. For resting-state fMRI, we can assume $D(t)$ is a constant, so the ODE on $dD(t)/dt$ can be removed from the model.

2.2 Model-Driven Learning Framework

We propose the Model-Driven Learning Framework (MDL) for parameter estimation in generic non-linear dynamical systems. As shown in Algo. 1, MDL iteratively performs three steps:

1) the forward-pass. For discrete layer models or explicit models the forward-pass is easy; for dynamical systems defined by ODEs, the forward-pass in the integration using any existing ODE solvers such as the RUnge-Kutta methods [18].

2) the backward pass. For explicit models or discrete-layer neural networks, the backward-pass can be performed by layer-wise back-propagation [134]; for dynamical systems, the analytical result of gradient is defined by the adjoint state equation [121], but in practice we need to take numerical implementations very carefully as discussed in Chapter 3.

3) Update of parameters based on gradient. Gradient based optimizers has been an ancient yet active topic [132], for deep learning specifically, the widely used Stochastic Gradient Descent (SGD) [15] typically generalizes well but converges slower compared to the adaptive optimizers such as Adam [74]. In Chapter 5, we propose a new optimizer that simultaneously achieves fast convergence and generalization, and show that our proposed optimizer significantly accelerates the fitting of MDL.

Algorithm 1: Model-Driven Learning Framework

While not converge

1. Forward-pass

Under current parameter θ, simulate the system (e.g. integrate ODE).

2. Backward-pass

Calculate loss, derive the gradient w.r.t. parameters.

3. Update

Update parameters by gradient.

Chapter 3

Numerical methods for gradient estimation in continuous-time models

3.1 Introduction

Recent research builds the connection between continuous models and neural networks. The theory of dynamical systems has been applied to analyze the properties of neural networks or guide the design of networks [164, 137, 91]. In these works, a residual block [53] is typically viewed as a one-step Euler discretization of an ODE; instead of directly analyzing the discretized neural network, it might be easier to analyze the ODE.

Another direction is the neural ordinary differential equation (Neural ODE) [21], which takes a continuous depth instead of discretized depth. The dynamics of a Neural ODE is typically approximated by numerical integration with adaptive ODE solvers. Neural ODEs have been applied in irregularly sampled time-series [131], free-form continuous generative models [48, 41], mean-field games [138], stochastic differential equations [82] and physically informed modeling [140, 176].

Though the Neural ODE has been widely applied in practice, how to train it is not extensively studied. The *naive method* directly backpropagates through an ODE solver, but

tracking a continuous trajectory requires a huge memory. Chen et al. [21] proposed to use the *adjoint method* to determine the gradient in continuous cases, which achieves constant memory cost *w.r.t* integration time; however, we show that the adjoint method suffers from numerical errors due to the inaccuracy in reverse-time trajectory. We propose the *adaptive checkpoint adjoint (ACA)* method to achieve accuracy in gradient estimation at a much smaller memory cost compared to the naive method, yet the memory consumption of ACA still grows linearly with integration time. Due to the non-constant memory cost, neither ACA nor naive method are suitable for large scale datasets (e.g. ImageNet) or high-dimensional Neural ODEs (e.g. FFJORD [48]).

In this project, we further propose the Memory-efficient Asynchronous Leapfrog Integrator (MALI) to achieve advantages of both the adjoint method and ACA: constant memory cost *w.r.t* integration time and accuracy in reverse-time trajectory. MALI is based on the asynchronous leapfrog (ALF) integrator [105]. With the ALF integrator, each numerical step forward in time is reversible. Therefore, with MALI, we delete the trajectory and only keep the end-time states, hence achieve constant memory cost *w.r.t* integration time; using the reversibility, we can accurately reconstruct the trajectory from the end-time value, hence achieve accuracy in gradient. Our contributions are:

1. We theoretically analyze the numerical error with the adjoint and naive methods, and propose ACA to accurately estimate gradients of NODEs.

2. We propose a new method (MALI) to solve Neural ODEs, which achieves constant memory cost *w.r.t* number of solver steps in integration and accuracy in gradient estimation. We provide theoretical analysis.

3. We validate our method with extensive experiments: (a) for image classification tasks, ACA and MALI enable a Neural ODE to achieve better accuracy than a well-tuned ResNet with the same number of parameters; to our knowledge, MALI is the first

method to enable training of Neural ODEs on a large-scale dataset such as ImageNet, while existing methods fail due to either heavy memory burden or inaccuracy. (b) In time-series modeling, ACA and MALI achieve comparable or better results than other methods. (c) For generative modeling, a FFJORD model trained with MALI achieves new state-of-the-art results on MNIST and Cifar10. (d) For DCM modeling in fMRI analysis, our method achieves significantly lower fitting loss.

3.2 Preliminaries

3.2.1 Numerical Integration Methods

An ordinary differential equation (ODE) typically takes the form

$$\frac{\mathrm{d}z(t)}{\mathrm{d}t} = f_\theta(t, z(t)) \quad s.t. \quad z(t_0) = x,\ t \in [t_0, T], \quad Loss = L(z(T), y) \tag{3.1}$$

where $z(t)$ is the hidden state evolving with time, T is the end time, t_0 is the start time (typically 0), x is the initial state. The derivative of $z(t)$ w.r.t t is defined by a function f, and f is defined as a sequence of layers parameterized by θ. The loss function is $L(z(T), y)$, where y is the target variable. Eq. 3.1 is called the initial value problem (IVP) because only $z(t_0)$ is specified.

Algorithm 2: Numerical Integration

Input initial state x, start time t_0, end time T, error tolerance $etol$, initial stepsize h.

Initialize $z(0) = x, t = t_0$

While $t < T$

  $error\_est = \infty$

  While $error\_est > etol$

    $h \leftarrow h \times DecayFactor$

    $\hat{z},\ error\_est = \psi_h(t, z)$

  If $error\_est < etol$

    $h \leftarrow h \times IncreaseFactor$

  $t \leftarrow t + h,\ \ z \leftarrow \hat{z}$

Notations We summarize the notations following Zhuang et al. [180].

- $z_i(t_i)/\overline{z}(\tau_i)$: hidden state in forward/reverse time trajectory at time t_i/τ_i.
- $\Phi_{t_i}^{t}(z_i)$: the *oracle* solution of the ODE at time t, starting from (t_i, z_i). Black dashed curve in Fig. 3.1 and Fig. 3.2. Φ is called the *flow map*.
- $\psi_{h_i}(t_i, z_i)$: the *numerical* solution at time $t_i + h_i$, starting from (t_i, z_i). Blue solid line in Fig. 3.1.
- $L_{h_i}(t_i, z_i)$: local truncation error between numerical approximation and oracle solution, where

$$L_{h_i}(t_i, z_i) = \psi_{h_i}(t_i, z_i) - \Phi_{t_i}^{t_i+h_i}(z_i) \tag{3.2}$$

- R_i: the local error $L_{h_i}(t_i, z_i)$ propagated to end time.

$$R_i = \Phi_{t_{i+1}}^{T}(z_{i+1}) - \Phi_{t_i}^{T}(z_i) \tag{3.3}$$

- N_f, N_z: N_f is the number of layers in f in Eq. 3.1, N_z is the dimension of z.
- N_t/N_r: number of discretized points (outer iterations in Algo. 2) in forward / reverse

integration.

- m: average number of inner iterations in Algo. 2 to find an acceptable stepsize.

Numerical Integration The algorithm for general adaptive-stepsize numerical ODE solvers is summarized in Algo. 2 [161]. The solver repeatedly advances in time by a step, which is the outer loop in Algo. 2 (blue curve in Fig. 3.1). For each step, the solver decreases the stepsize until the estimate of error is lower than the tolerance, which is the inner loop in Algo. 2 (green curve in Fig. 3.1). For fixed-stepsize solvers, the inner loop is replaced with a single evaluation of $\psi_h(t, z)$ using predefined stepsize h. Different methods typically use different ψ, for example different orders of the Runge-Kutta method [135].

3.2.2 Analytical form of gradient in continuous case

We first briefly introduce the analytical form of the gradient in the continuous case, then we compare different numerical implementations in the literature to estimate the gradient. The analytical form of the gradient in the continuous case is

$$\frac{\mathrm{d}L}{\mathrm{d}\theta} = -\int_T^0 a(t)^\top \frac{\partial f(z(t), t, \theta)}{\partial \theta} dt \tag{3.4}$$

$$\frac{\mathrm{d}a(t)}{dt} + \Big(\frac{\partial f(z(t), t, \theta)}{\partial z(t)}\Big)^\top a(t) = 0 \ \ \forall t \in (0, T), \quad a(T) = \frac{\partial L}{\partial z(T)} \tag{3.5}$$

where $a(t)$ is the "adjoint state". Detailed proof is given in [121]. In the next section we compare different numerical implementations of this analytical form.

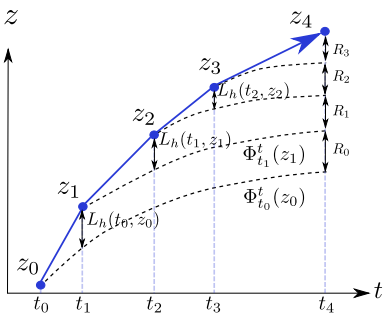


Figure 3.1: Illustration of numerical solver in forward-pass. For adaptive solvers, for each step forward-in-time, the stepsize is recursively adjusted until the estimated error is below predefined tolerance; the search process is represented by green curve, and the accepted step (ignore the search process) is represented by blue curve.

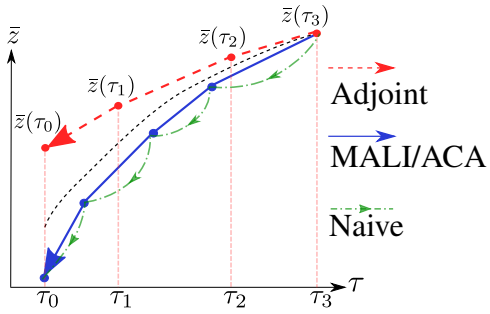


Figure 3.2: In backward-pass, the adjoint method reconstructs trajectory as a separate IVP. Naive, ACA and MALI track the forward-time trajectory, hence are accurate. ACA and MALI backpropagate through the accepted step, while naive method backpropagates through the search process hence has deeper computation graphs.

3.3 Numerical implementations in the literature

We compare different numerical implementations of the analytical form in this section. The forward-pass and backward-pass of different methods are demonstrated in Fig. 3.1 and Fig. 3.2 respectively. Forward-pass is similar for different methods. The comparison of backward-pass among different methods are summarized in Table. 3.1. We explain methods in the literature below.

Naive method The naive method saves all of the computation graph (including search for optimal stepsize, green curve in Fig. 3.2) in memory, and backpropagates through it. Hence the memory cost is $N_z N_f \times N_t \times m$ and depth of computation graph are $N_f \times N_t \times m$, and the computation is doubled considering both forward and backward passes. Besides the large memory and computation, the deep computation graph might cause vanishing or exploding gradient [115].

Adjoint method Note that we use "*adjoint state equation*" to refer to the *analytical form* in Eq. 6.11 and 6.10, while we use "*adjoint method*" to refer to the *numerical implementation* by Chen et al. [21]. As in Fig. 3.1 and 3.2, the adjoint method forgets

forward-time trajectory (blue curve) to achieve memory cost $N_z N_f$ which is constant to integration time; it takes the end-time state (derived from forward-time integration) as the initial state, and solves a separate Initial Value Problem (IVP, red curve) in reverse-time.

3.3.1 Adjoint method suffers from numerical errors

Theorem 3.3.1 (Picard-Lindelöf Theorem)**.** *[86] Consider the initial value problem (IVP):*

$$\frac{dz}{dt} = f(t, z), \quad z(t = 0) = z_0$$

Suppose in a region $R = [t_0 - a, t_0 + a] \times [z_0 - b, z_0 + b]$*,* f *is bounded (*$||f|| \leq M$*), uniformly continuous in* z *with Lipschitz constant* L*, and continuous in* t*; then there exists a unique solution for the IVP, valid on the region where* $a < min\{\frac{b}{M}, \frac{1}{L}\}$.

The Picard-Lindelöf Theorem states a sufficient condition for existence and uniqueness for an IVP. Okamura [113] stated a necessary and sufficient condition. Without going deeper, we emphasize that Theorem 3.3.1 has a validity region, outside this region the theorem may not hold.

It is trivial to check NODE satisfies the above conditions; see the proof in Appendix B. For simplicity, we assume Theorem 3.3.1 always holds on $t \in [0, T]$. (If $[0, T]$ is outside the region of validity, the adjoint method cannot recover the forward-time trajectory, while the naive method and ACA record the trajectory in memory with "checkpoint".)

Recall $\Phi_{t_i}^t(z_i)$ is the *flow map*, which is the *oracle* solution starting from (t_i, z_i). Define the *variational flow* as:

$$D\Phi_{t_0}^t = \frac{d\Phi_{t_0}^t(z_0)}{dz_0} \tag{3.6}$$

Consider an ODE solver of order p. The local truncation error $L_h(t_i, z_i)$ is of order

$O(h^{p+1})$ and can be written as

$$L_h(t_i, z_i) = \psi_h(t_i, z_i) - \Phi_{t_i}^{t_i+h}(z_i) = h^{p+1} l(t_i, z_i) + O(h^{p+2}) \tag{3.7}$$

where l is some function of $O(1)$. Denote the global error as $G(T)$ at time T, then it satisfies:

$$G(T) = z_{N_t} - \Phi_{t_0}^{T}(z_0) = \sum_{k=0}^{N_t-1} R_k \tag{3.8}$$

Eq. 3.8 is explained by Fig. 3.1: global error is the sum of all local errors propagated to the end time. R_k is the propagated local error defined by Eq. 3.3. For simplicity of analysis, we consider constant stepsize solvers with sufficiently small stepsize h, and let $N_t = N_r = N$.

Theorem 3.3.2. *If the conditions of the Picard-Lindelöf theorem are satisfied, then for an ODE solver of order p, the global error at time T is:*

$$G(t_0 \to T) = \sum_{k=0}^{N-1} \left[h_k^{p+1} D\Phi_{t_k}^{T}(z_k) l(t_k, z_k) \right] + O(h^{p+1}) \tag{3.9}$$

and the error of the reconstructed initial value by the adjoint method is:

$$\begin{aligned} G(t_0 \to T \to t_0) &= G(t_0 \to T) + G(T \to t_0) \\ &= \sum_{k=0}^{N-1} \left[h_k^{p+1} D\Phi_{t_k}^{T}(z_k) l(t_k, z_k) + (-h_k)^{p+1} D\Phi_{T}^{t_k}(\overline{z_k}) \overline{l(t_k, \overline{z_k})} \right] + O(h^{p+1}) \end{aligned} \tag{3.10}$$

where $G(t_0 \to T \to t_0)$ represents the global error of integration from t_0 to T, then from T to t_0. Terms for reverse-time trajectory are overlined ($\bar{l}, \overline{z}$) to differentiate from forward-time trajectory.

Proofs are provided in Sec. 3.7. Eq. 3.10 can be divided into two parts. $G(t_0 \to$

T) corresponds to forward-time error, as shown in Fig. 3.1; $G(T \to t_0)$ corresponds to reverse-time error, as shown in Fig. 3.2. When h is small, assume:

$$\overline{z_k} = z_k + O(h^p) \tag{3.11}$$

$$\overline{l(t_k, \overline{z_k})} = l(t_k, z_k) + O(h^p) \tag{3.12}$$

$$D\Phi_T^{t_k}(\overline{z_k}) = D\Phi_T^{t_k}(z_k) + O(h^p) \tag{3.13}$$

Note that when existence and uniqueness are satisfied, Φ defines a bijective mapping between $z(t_k)$ and $z(T)$, hence

$$D\Phi_T^{t_k} = (D\Phi_{t_k}^T)^{-1} \tag{3.14}$$

Plugging Eq. 3.11-3.14 into Eq. 3.10,

$$G(t_0 \to T \to t_0) = \textstyle\sum_{k=0}^{N-1} h^{p+1} l(t_k, z_k) e_k + O(h^{p+1}) \tag{3.15}$$

$$e_k = D\Phi_{t_k}^T(z_k) + (-1)^{p+1}(D\Phi_{t_k}^T(z_k))^{-1} \tag{3.16}$$

Reverse accuracy for all t_0 requires $e_k = 0$ for all k. If p is odd, the two terms in Eq. 3.16 are of the same sign; thus, e_k cannot be 0. If p is even, $e_k = 0$ requires $D\Phi_{t_k}^T(z_k) = I$, which requires NODE to be an identity function; in this case the model learns nothing. Hence, the adjoint method has numerical errors caused by truncation errors of numerical ODE solvers.

Theorem 3.3.3. *For an ODE solver of order p, the error of the reconstructed initial value by the adjoint method is $\sum_{k=0}^{N-1} \left[h_k^{p+1} D\Phi_{t_k}^T(z_k) l(t_k, z_k) + (-h_k)^{p+1} D\Phi_T^{t_k}(\overline{z_k}) \overline{l(t_k, \overline{z_k})}\right] + O(h^{p+1})$, where Φ is the ideal solution, $D\Phi$ is the Jacobian of Φ, $l(t, z)$ and $\overline{l(t, z)}$ are the local error in forward-time and reverse-time integration respectively.*

Proofs are provided in Sec. 3.7. To summarize, due to inevitable errors with numerical ODE solvers, the reverse-time trajectory (red curve, $\overline{z}(\tau)$) cannot match the forward-time trajectory (blue curve, $z(t)$) accurately. The error in $\overline{z}$ propagates to $\frac{\mathrm{d}L}{\mathrm{d}\theta}$ by Eq. 6.11, hence affects the accuracy in gradient estimation.

3.3.2 Naive Method has Deep Computation Graph

Note that for each step advance in time, there are on average m steps to find an acceptable stepsize, as in Algo. 2. We give an example below:

$$out_1, h_1, error_1 = \psi(t, h_0, z) \tag{3.17}$$

$$out_2, h_2, error_2 = \psi(t, h_1, z) \tag{3.18}$$

$$... \tag{3.19}$$

$$out_m, h_m, error_m = \psi(t, h_m, z) \tag{3.20}$$

Suppose it takes m steps for find an acceptable stepsize such that $error_m < tolerance$. The naive method treats h_m as a recursive function of h_0, and back-propagates through all m steps in the computation graph; while ACA takes h_m as a constant, and back-propagates only through the final accepted step (Eq. 3.20); therefore, the depth of computation graph is $O(N_f \times N_t)$ for ACA, and $O(N_f \times N_t \times m)$ for the naive method. Note that the output of the forward pass is the same for both methods; the backward pass is different.

The very deep computation graph in naive method takes more memory. More importantly, it might cause vanishing or exploding gradient [115], since there's no special structure such as residual connection to deal with the deep structure: specifically, in Eq. 3.17 to Eq. 3.20, only h_i is passed to the next step, and typically in the form $h_{i+1} = h_i/error_i^p$.

3.4 Methods

3.4.1 Adaptive checkpoint adjoint (ACA)

ACA tries to record $z(t)$ to avoid numerical errors, while also controlling memory cost. ACA supports both adaptive and constant stepsize ODE solvers. ACA is summarized in Algo. 3. Note that the forward-pass computation is the same as Algo. 2 for all methods discussed in this chapter.

During the forward-pass, to save memory, ACA deletes redundant computation graphs to search for the optimal stepsize. Instead, ACA applies the "trajectory checkpoint" strategy, recording the discretization points t_i (equivalently, the accepted stepsize $h_i = t_{i+1} - t_i$) and values z_i (not computation graph $\psi_{h_i}(t_i, z_i)$) at a memory cost $O(N_t)$. Considering $O(N_f)$ memory cost for one evaluation of ψ, the total memory cost is $O(N_f + N_t)$.

During the backward-pass, going reverse-time, ACA performs the forward-pass and backward-pass *locally* from t_i to t_{i+1}, and updates λ and $\frac{dL}{d\theta}$. Computations are evaluated at saved discretization points $\{t_0, ...t_{N_t}\}$, using saved values $\{z_0, ..., z_{N_t}\}$, to guarantee accuracy between forward-time and reverse-time trajectory. We only need to search for optimal stepsize during the forward-pass, with m inner iterations in Algo. 2; during the backward-pass we reuse saved stepsizes, so the total computation cost is $O(N_f \times N_t \times (m+1))$.

3.4.2 Asynchronous Leapfrog Integrator

In this section we give a brief introduction to the asynchronous leapfrog (ALF) method [105], and we provide theoretical analysis which is missing in Mutze [105]. For general first-order ODEs in the form of Eq. 3.1, the tuple (z, t) is sufficient for most ODE solvers to take a step numerically. For ALF, the required tuple is (z, v, t), where v is the "approx-

Algorithm 3: ACA: Record $z(t)$ with Minimal Memory

Forward $(f, T, z_0, tolerance)$:

$t = 0, z = z_0$

$state_0 = f.state\_dict()$, cache.save($state_0$)

Select initial step size $h = h_0$ (adaptively with adaptive step-size solver).

$time\_points = empty\_list()$

$z\_values = empty\_list()$

While $t < T$:

$state = f.state\_dict(), \;\; accept\_step = False$

While Not $accept\_step$:

$f.load\_state\_dict(state)$

with $grad\_disabled$:

$z\_new, error\_estimate = \psi(f, z, t, h)$

If $error\_estimate < tolerance$:

$accept\_step = True$

$z = z\_new, \;\; t = t + h,$

$z\_values.append(z), \;\; time\_points.append(t)$

else:

reduce stepsize h according to $error\_estimate$

delete $error\_estimate$ local computation graph

cache.save($time\_points, \;\; z\_values$)

return z, cache

Backward $(f, T, tolerance, cache, \frac{\partial J}{\partial z(T)})$:

Initialize $\lambda = -\frac{\partial J}{\partial z(T)}, \;\; \frac{\partial L}{\partial \theta} = 0$

$\{z_0, z_1, z_2, ...z_{N-1}, z_N\} = cache.z\_values$

$\{t_0, t_1, t_2, ...t_{N-1}, t_N\} = cache.time\_points$

For t_i in $\{t_N, t_{N-1}, ..., t_1, t_0\}$:

Local forward $\hat{z}_i = \psi(f, z_{i-1}, t_{i-1}, h_i = t_i - t_{i-1})$

Local backward

$\frac{\partial L}{\partial \theta} \leftarrow \frac{\partial L}{\partial \theta} - \lambda^\top \frac{\partial \hat{z}_i}{\partial \theta}$

$\lambda \leftarrow \lambda^\top \frac{\partial \hat{z}_i}{\partial z_{i-1}}$

delete local computation graph

return $\frac{\partial L}{\partial \theta}, \;\; \lambda$

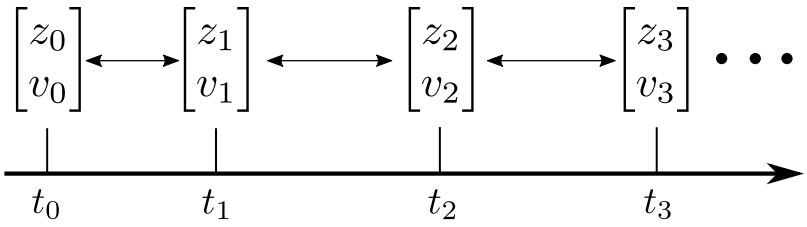


Figure 3.3: With ALF method, given any tuple (z_j, v_j, t_j) and discretized time points $\{t_i\}_{i=1}^{N_t}$, we can reconstruct the entire trajectory accurately due to the reversibility of ALF.

imated derivative". Most numerical ODE solvers such as the Runge-Kutta method [135] track state z evolving with time, while ALF tracks the "augmented state" (z, v). We explain the details of ALF as below.

Algorithm 4: Forward of ψ in ALF

Input $(z_{in}, v_{in}, s_{in}, h)$ where s_{in} is current time, z_{in} and v_{in} are correponding values at time s_{in}, h is stepsize.

Forward

$$s_1 = s_{in} + h/2$$
$$k_1 = z_{in} + v_{in} \times h/2$$
$$u_1 = f(k_1, s_1)$$
$$v_{out} = v_{in} + 2(u_1 - v_{in})$$
$$z_{out} = k_1 + v_{out} \times h/2$$
$$s_{out} = s_1 + h/2$$

Output $(z_{out}, v_{out}, s_{out}, h)$

Algorithm 5: ψ^{-1} (Inverse of ψ) in ALF

Input $(z_{out}, v_{out}, s_{out}, h)$ where s_{out} is current time, z_{out} and v_{out} are corresponding values at s_{out}, h is stepsize.

Inverse

$$s_1 = s_{out} - h/2$$
$$k_1 = z_{out} - v_{out} \times h/2$$
$$u_1 = f(k_1, s_1)$$
$$v_{in} = 2u_1 - v_{out}$$
$$z_{in} = k_1 - v_{in} \times h/2$$
$$s_{in} = s_1 - h/2$$

Output $(z_{in}, v_{in}, s_{in}, h)$

Procedure of ALF Different ODE solvers have different ψ in Algo. 2, hence we only summarize ψ for ALF in Algo. 4. Note that for a complete algorithm of integration for ALF, we need to plug Algo. 4 into Algo. 2. The forward-pass is summarized in Algo. 4. Given stepsize h, with input (z_{in}, v_{in}, s_{in}), a single step of ALF outputs $(z_{out}, v_{out}, s_{out})$.

As in Fig. 3.3, given (z_0, v_0, t_0), the numerical forward-time integration calls Algo. 4

iteratively:

$$(z_i, v_i, t_i, h_i) = \psi(z_{i-1}, v_{i-1}, t_{i-1}, h_i)$$
$$s.t.\ \ h_i = t_i - t_{i-1},\ \ i = 1, 2, ...N_t \tag{3.21}$$

Invertibility of ALF An interesting property of ALF is that ψ defines a bijective mapping; therefore, we can reconstruct $(z_{in}, v_{in}, s_{in}, h)$ from $(z_{out}, v_{out}, s_{out}, h)$, as demonstrated in Algo. 5. As in Fig. 3.3, we can reconstruct the entire trajectory given the state (z_j, v_j) at time t_j, and the discretized time points $\{t_0, ...t_{N_t}\}$. For example, given (z_{N_t}, v_{N_t}) and $\{t_i\}_{i=0}^{N_t}$, the trajectory for Eq. 3.21 is reconstructed:

$$(z_{i-1}, v_{i-1}, t_{i-1}, h_i) = \psi^{-1}(z_i, v_i, t_i, h_i)\ \ s.t.\ \ h_i = t_i - t_{i-1},\ \ i = N_t, N_t - 1, ..., 1 \tag{3.22}$$

In the following sections, we will show the invertibility of ALF is the key to maintain accuracy at a constant memory cost to train Neural ODEs. Note that "inverse" refers to reconstructing the input from the output without computing the gradient, hence is different from "back-propagation".

Initial value For an initial value problem (IVP) such as Eq. 3.1, typically $z_0 = z(t_0)$ is given while v_0 is undetermined. We can construct $v_0 = f(z(t_0), t_0)$, so the initial augmented state is (z_0, v_0).

Difference from midpoint integrator The midpoint integrator [147] is similar to Algo. 4, except that it recomputes $v_{in} = f(z_{in}, s_{in})$ for every step, while ALF directly uses the input v_{in}. Therefore, the midpoint method does not have an explicit form of inverse.

Local truncation error Theorem 3.4.1 indicates that the local truncation error of ALF is of order $O(h^3)$; this implies the global error is $O(h^2)$. Detailed proof is in Ap-

pendix 3.7.4.

Theorem 3.4.1. *For a single step in ALF with stepsize h, the local truncation error of z is $O(h^3)$, and the local truncation error of v is $O(h^2)$.*

A-Stability The ALF solver has a limited stability region, but this can be solved with damping. The damped ALF replaces the update of v_{out} in Algo. 4 with $v_{out} = v_{in} + 2\eta(u_1 - v_{in})$, where η is the "damping coefficient" between 0 and 1. We have the following theorem on its numerical stability.

Theorem 3.4.2. *For the damped ALF integrator with stepsize h, where σ_i is the i-th eigenvalue of the Jacobian $\frac{\partial f}{\partial z}$, then the solver is A-stable if*

$$\left|1 + \eta(h\sigma_i - 1) \pm \sqrt{\eta\left[2h\sigma_i + \eta(h\sigma_i - 1)^2\right]}\right| < 1,\ \forall i$$

Proof is in Appendix 3.7.5 and 3.7.6. Theorem 3.4.2 implies the following: when $\eta = 1$, the damped ALF reduces to ALF, and the stability region is empty; when $0 < \eta < 1$, the stability region is non-empty. However, stability describes the behaviour when T goes to infinity; in practice we always use a bounded T and ALF performs well.

3.4.3 Memory-efficient ALF Integrator (MALI) for gradient estimation in continuous-time models

An ideal solver for Neural ODEs should achieve two goals: accuracy in gradient estimation and constant memory cost *w.r.t* integration time. Adjoint method achieves constant memory at the cost of inaccuracy, ACA guarantees accuracy but requires a linearly growing memory cost. We propose a method based on the ALF solver, which to our knowledge is the first method to achieve the two goals simultaneously.

Procedure of MALI Details of MALI are summarized in Algo. 6. For the forward-pass,

Algorithm 6: MALI to acheive accuracy at a constant memory cost *w.r.t* integration time

Input Initial state z_0, start time t_0, end time T

Forward

Apply the numerical integration in Algo. 2, with the ψ function defined by Algo. 4.

Delete computation graph on the fly, only keep end-time state (z_{N_t}, v_{N_t})

Keep *accepted* discretized time points $\{t_i\}_{i=0}^{N_t}$ (ignore process to search for optimal stepsize)

Backward

Initialize $a(T) = \frac{\partial L}{\partial z(T)}$ by Eq. 6.10, initialize $\frac{\mathrm{d}L}{\mathrm{d}\theta} = 0$

For i in $\{N_t, N_t - 1, ..., 2, 1\}$:

Reconstruct (z_{i-1}, v_{i-1}) from (z_i, v_i) by Algo. 5

Local forward $(z_i, v_i, t_i, h_i) = \psi(z_{i-1}, v_{i-1}, t_{i-1}, h_i)$

Local backward, get $\frac{\partial f(z_{i-1}, t_{i-1}, \theta)}{\partial z_{i-1}}$ and $\frac{\partial f(z_{i-1}, t_{i-1}, \theta)}{\partial \theta}$

Update $a(t)$ and $\frac{\mathrm{d}L}{\mathrm{d}\theta}$ by Eq. 6.11 and Eq. 6.10 discretized at time points t_{i-1} and t_i

Delete local computation graph

Output the adjoint state $a(t_0)$ (gradient *w.r.t* input z_0) and parameter gradient $\frac{\mathrm{d}L}{\mathrm{d}\theta}$

we only keep the end-time state (z_{N_t}, v_{N_t}) and the *accepted* discretized time points (blue curves in Fig. 3.1 and 3.2). We ignore the search process for optimal stepsize (green curve in Fig. 3.1 and 3.2), and delete other variables to save memory. During the backward pass, we can reconstruct the forward-time trajectory as in Eq. 3.22, then calculate the gradient by numerical discretization of Eq. 6.11 and Eq. 6.10.

Constant memory cost *w.r.t* number of solver steps in integration We delete the computation graph and only keep the end-time state to save memory. The memory cost is $N_z(N_f + 1)$, where $N_z N_f$ is due to evaluating $f(z, t)$ and is irreducible for all methods. Compared with the adjoint method, MALI only requires extra N_z memory to record v_{N_t}, and also has a constant memory cost *w.r.t* time step N_t. The memory cost is $N_z(N_f + 1)$.

Accuracy Our method guarantees the accuracy of reverse-time trajectory (e.g. blue curve in Fig. 3.2 matches the blue curve in Fig. 3.1), because ALF is explicitly invertible for free-form f (see Algo. 5). Therefore, the gradient estimation in MALI is more accurate

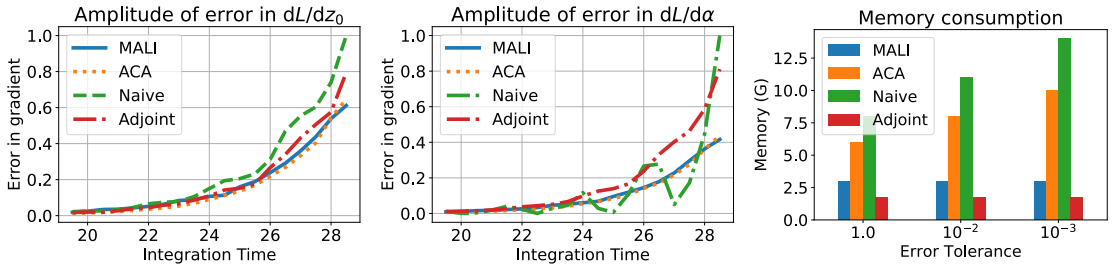


Figure 3.4: Comparison of error in gradient in Eq. 3.23. (a) error in $\frac{\mathrm{d}L}{\mathrm{d}z_0}$. (b) error in $\frac{\mathrm{d}L}{\mathrm{d}\alpha}$. (c) memory cost.

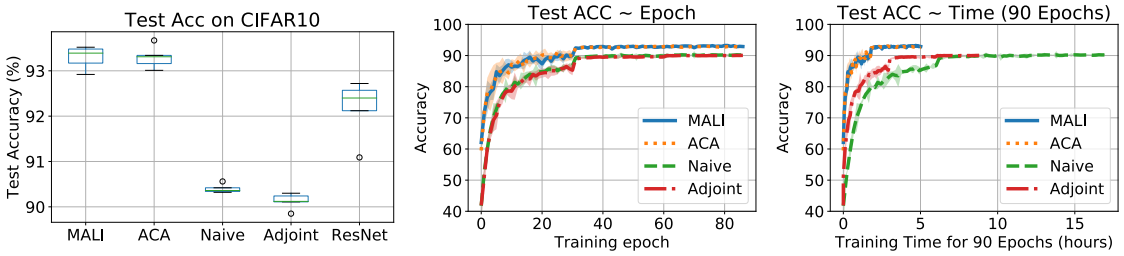


Figure 3.5: Results on Cifar10. From left to right: (1) box plot of test accuracy (first 4 columns are Neural ODEs, last is ResNet); (2) test accuracy $\pm std$ v.s. training epoch for Neural ODE; (3) test accuracy $\pm std$ v.s. training time of 90 epochs for Neural ODE.

compared to the adjoint method.

Computation cost Recall that on average it takes m steps to find an acceptable step-size, whose error estimate is below tolerance. Therefore, the forward-pass with search process has computation burden $N_z \times N_f \times N_t \times m$. Note that we only reconstruct and backprop through the *accepted* step and ignore the search process, hence it takes another $N_z \times N_f \times N_t \times 2$ computation. The overall computation burden is $N_z N_f \times N_t \times (m+2)$ as in Table 3.1.

Shallow computation graph Similar to ACA, MALI only backpropagates through the accepted step (blue curve in Fig. 3.2) and ignores the search process (green curve in Fig. 3.2), hence the depth of computation graph is $N_f \times N_t$. The computation graph of MALI is much shallower than the naive method, hence is more robust to vanishing and exploding gradients [115].

Summary The adjoint method suffers from inaccuracy in reverse-time trajectory, the naive method suffers from exploding or vanishing gradient caused by deep compu-

Table 3.1: Comparison between different methods for gradient estimation in continuous case. MALI achieves reverse accuracy, constant memory *w.r.t* number of solver steps in integration, shallow computation graph and low computation cost.

	Naive	Adjoint	ACA	MALI
Computation	$NzN_f \times N_t \times m \times 2$	$NzN_f \times (N_t + N_r) \times m$	$NzN_f \times N_t \times (m+1)$	$NzN_f \times N_t \times (m+2)$
Memory	$NzN_f \times N_t \times m$	NzN_f	$Nz(N_f + N_t)$	$Nz(N_f + 1)$
Computation graph depth	$N_f \times N_t \times m$	$N_f \times N_r$	$N_f \times N_t$	$N_f \times N_t$
Reverse accuracy	✓	✗	✓	✓

tation graph, and ACA finds a balance but the memory grows linearly with integration time. MALI achieves accuracy in reverse-time trajectory, constant memory *w.r.t* integration time, and a shallow computation graph.

3.5 Experiments

3.5.1 Validation on a toy example

We compare the performance of different methods on a toy example, defined as

$$L(z(T)) = z(T)^2 \;\; s.t. \;\; z(0) = z_0, \;\; \mathrm{d}z(t)/\mathrm{d}t = \alpha z(t) \tag{3.23}$$

The analytical solution is

$$z(t) = z_0 e^{\alpha t}, \;\; L = z_0^2 e^{2\alpha T}, \;\; \mathrm{d}L/\mathrm{d}z_0 = 2z_0 e^{2\alpha T}, \;\; \mathrm{d}L/d\alpha = 2Tz_0^2 e^{2\alpha T} \tag{3.24}$$

We plot the amplitude of error between numerical solution and analytical solution varying with T (integrated under the same error tolerance, rtol $= 10^{-5}$, atol $= 10^{-6}$) in Fig 6.1. ACA and MALI have similar errors, both outperforming other methods. We also plot the memory consumption for different methods on a Neural ODE with the same input in Fig. 6.1. As the error tolerance decreases, the solver evaluates more steps, hence the naive method and ACA increase memory consumption, while MALI and the adjoint method

Table 3.2: Top-1 test accuracy of Neural ODE and ResNet on ImageNet. Neural ODE is trained with MALI, and ResNet is trained as the original model; Neural ODE is tested using different solvers *without* retraining.

	Fixed-stepsize solvers of various stepsizes						Adaptive-stepsize solver of various tolerances			
	Stepsize	1	0.5	0.25	0.15	0.1	Tolerance	1.00E+00	1.00E-01	1.00E-02
Neural ODE	MALI	42.33	66.4	69.59	70.17	69.94	MALI	62.56	69.89	69.87
	Euler	21.94	61.25	67.38	68.69	70.02	Heun-Euler	68.48	69.87	69.88
	RK2	42.33	69	69.72	70.14	69.92	RK23	50.77	69.89	69.93
	RK4	12.6	69.99	69.91	**70.21**	69.96	Dopri5	52.3	68.58	69.71
ResNet	70.09									

Table 3.3: Top-1 accuracy under FGSM attack. ϵ is the perturbation amplitude. For Neural ODE models, row names represent the solvers to derive the gradient for attack, and column names represent solvers for inference on the perturbed image.

		$\epsilon = 1/255$				$\epsilon = 2/255$			
		MALI	Heun-Euler	RK23	Dopri5	MALI	Heun-Euler	RK23	Dopri5
Neural ODE	MALI	14.69	14.72	14.77	15.71	10.38	10.46	10.62	10.62
	Heun-Euler	14.77	14.75	14.80	**15.74**	10.63	10.47	10.44	10.49
	RK23	14.82	14.77	14.79	15.69	**10.78**	10.53	10.48	10.56
	Dopri5	14.82	14.78	14.79	15.15	10.76	10.49	10.48	10.51
ResNet		13.02				9.57			

have a constant memory cost. These results validate our analysis in Sec. 3.4.3 and Table 3.1, and shows MALI achieves accuracy at a constant memory cost.

3.5.2 Image recognition with Neural ODE

We validate MALI on image recognition tasks using Cifar10 and ImageNet datasets. We modify a ResNet18 into its corresponding Neural ODE: the forward function is $y = x + f_\theta(x)$ and $y = x + \int_0^T f_\theta(z)\mathrm{d}t$ for the residual block and Neural ODE respectively, where the same f_θ is shared. We compare MALI and ACA with the naive method and adjoint method.

Results on Cifar10 Results of 5 independent runs on Cifar10 are summarized in Fig. 4.10. MALI achieves comparable accuracy to ACA, and both significantly outperform the naive and the adjoint method. Furthermore, the training speed of MALI is similar to ACA, and both are almost two times faster than the adjoint memthod, and three times faster than the naive method. This validates our analysis on accuracy and computation burden in Table 3.1.

Accuracy on ImageNet Due to the heavy memory burden caused by large images, the

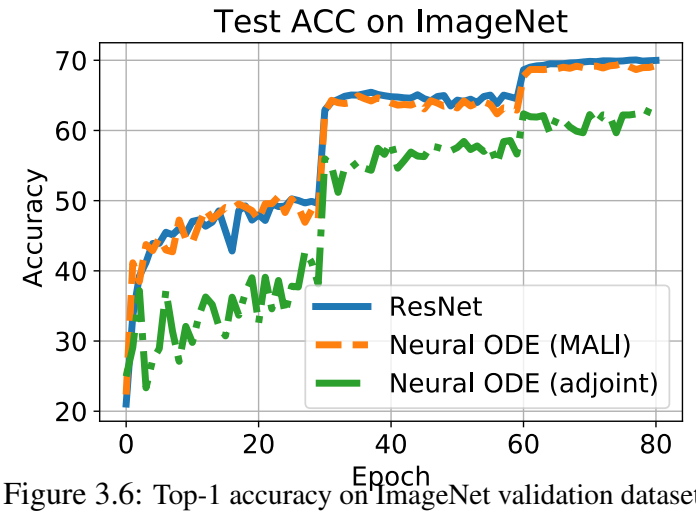


Figure 3.6: Top-1 accuracy on ImageNet validation dataset.

naive method and ACA are unable to train a Neural ODE on ImageNet with 4 GPUs; only MALI and the adjoint method are feasible due to the constant memory. We also compare the Neural ODE to a standard ResNet. As shown in Fig. 3.6, the accuracy of the Neural ODE trained with MALI closely follows ResNet, and significantly outperforms the adjoint method (top-1 validation: 70% v.s. 63%).

Invariance to discretization scheme A continuous model should be invariant to discretization schemes (e.g. different types of ODE solvers) as long as the discretization is sufficiently accurate. We test the Neural ODE using different solvers *without* re-training; since ResNet is often viewed as a one-step Euler discretization of an ODE [51], we perform similar experiments. As shown in Table 4.2, Neural ODE consistently achieves high accuracy ($\sim$70%), while ResNet drops to random guessing ($\sim$0.1%) because ResNet as a one-step Euler discretization fails to be a meaningful dynamical system [124].

Robustness to adversarial attack Hanshu et al. [52] demonstrated that Neural ODE is more robust to adversarial attack than ResNet on small-scale datasets such as Cifar10. We validate this result on the large-scale ImageNet dataset. The top-1 accuracy of Neural ODE and ResNet under FGSM attack [47] are summarized in Table 3.3. For Neural ODE, due to its invariance to discretization scheme, we derive the gradient for attack using a

certain solver (row in Table 3.3), and inference on the perturbed images using various solvers. For different combinations of solvers and perturbation amplitudes, Neural ODE consistently outperforms ResNet.

Summary In image recognition tasks, we demonstrate Neural ODE is accurate, invariant to discretization scheme, and more robust to adversarial attack than ResNet. Note that detailed explanation on the robustness of Neural ODE is out of the scope for this paper, but to our knowledge, MALI is the *first* method to enable training of Neural ODE on large datasets due to constant memory cost.

3.5.3 Time-series modeling

We apply MALI to latent-ODE [131] and Neural Controlled Differential Equation (Neural CDE) [71, 72]. Our experiment is based on the official implementation from the literature. We report the mean squared error (MSE) on the *Mujoco* test set in Table 3.4, which is generated from the "Hopper" model using DeepMind control suite [152]; for all experiments with different ratios of training data, MALI achieves similar MSE to ACA, and both outperform the adjoint and naive method. We report the test accuracy on the *Speech Command* dataset for Neural CDE in Table 3.5; MALI achieves a higher accuracy than competing methods.

3.5.4 Continuous generative models

We apply MALI on FFJORD [48], a free-from continuous generative model, and compare with several variants in the literature [41, 71]. Our experiment is based on the official implementaion of [41]; for a fair comparison, we train with MALI, and test with the same solver as in the literature [48, 41], the *Dopri5* solver with rtol $=$ atol $= 10^{-5}$ from the *torchdiffeq* package [21]. Bits per dim (BPD, lower is better) on validation set for various datasets are reported in Table 3.6. For continuous models, MALI consistently generates

Table 3.4: Test MSE ($\times 0.01$) on Mujoco dataset (lower is better). Results with superscripts correspond to literature in the footnote.

Percentage of training data	RNN[1]	RNN-GRU[1]	Latent-ODE			
			Adjoint[1]	Naive[2]	ACA[2]	MALI
10%	2.45[1]	1.97[2]	0.47[1]	0.36[2]	**0.31**[2]	0.35
20%	1.71[1]	1.42[1]	0.44[1]	0.30[2]	**0.27**[2]	**0.27**
50%	0.79[1]	0.75[1]	0.40[1]	0.29[2]	**0.26**[2]	**0.26**

Table 3.5: Test ACC on Speech Command Dataset

Method	Accuracy (%)
Adjoint[3]	92.8 ± 0.4
SemiNorm[3]	92.9 ± 0.4
Naive	93.2 ± 0.2
ACA	93.2 ± 0.2
MALI	$\mathbf{93.7 \pm 0.3}$

Table 3.6: Bits per dim (BPD) of generative models, *lower* is better. Results marked with superscript numbers correspond to literature in the footnote.

Dataset	Continuous Flow (FFJORD)				Discrete Flow				
	Vanilla[4]	RNODE[5]	SemiNorm[3]	MALI	RealNVP[6]	i-ResNet[7]	Glow[8]	Flow++[9]	Residual Flow[10]
MNIST	0.99[4]	0.97[5]	0.96[3]	**0.87**	1.06[6]	1.05[7]	1.05[8]	-	0.97[10]
CIFAR10	3.40[4]	3.38[5]	3.35[3]	**3.27**	3.49[6]	3.45[7]	3.35[8]	3.28[9]	3.28[10]
ImageNet64	-	3.83[5]	-	**3.71**	3.98[6]	-	3.81[8]	-	3.76[10]

the lowest BPD, and outperforms the Vanilla FFJORD (trained with adjoint), RNODE (regularized FFJORD) and the SemiNorm Adjoint [71]. Furthermore, FFJORD trained with MALI achieves comparable BPD to state-of-the-art discrete-layer flow models in the literature.

3.6 Related works

Besides ALF, the symplectic integrator [159, 167] is also able to reconstruct trajectory accurately, yet it's typically restricted to second order Hamiltonian systems [29], and are unsuitable for general ODEs. Besides aforementioned methods, there are other methods for gradient estimation such as interpolated adjoint [28] and spectral method [123], yet the implementations are involved and not publicly available. Other works focus on the theoretical properties of Neural ODEs [39, 151, 94]. Neural ODE is recently applied to stochastic differential equation [82], jump differential equation [65] and auto-regressive models [162].

[0]1. Rubanova et al. [131]; 2. Zhuang et al. [180]; 3. Kidger et al. [71]; 4. Chen et al. [21]; 5. Finlay et al. [41]; 6. Dinh et al. [35]; 7. Behrmann et al. [8]; 8. Kingma & Dhariwal [75]; 9. Ho et al. [59]; 10. Chen et al. [22]

3.7 Proofs and Theoretical Analysis

3.7.1 Numerical errors for the adjoint method

Lemma 3.7.0.1. *Suppose f is composed of a finite number of ReLU activations and linear transforms,*

$$f(t, z) = Linear_1 \circ ReLU \circ Linear_2 \circ ... \circ Linear_N(z)$$

if the spectral norm of linear transform is bounded, then the IVP defined above has a unique solution on a bounded region.

Proof: f does not depend on t explicitly, hence is continuous in t. ReLU (and other activation functions such as sigmoid, tanh, ...) is uniformly continuous; a linear transform Wz is also uniformly continuous if the spectral norm of W is bounded. From Picard-Lindelöf Theorem, the IVP has a unique solution on a bounded region.

Flow map Denote $\Phi_{t_0}^T(z_0)$ as the *oracle* solution to the IVP at time T, with the initial condition (t_0, z_0). Then $\Phi_{t_0}^T(z_0)$ satisfies:

$$\Phi_{t_2}^{t_3} \circ \Phi_{t_1}^{t_2} = \Phi_{t_1}^{t_3} \tag{3.25}$$

$$\frac{d}{dt}\Phi_{t_0}^t(z_0) = f(t, \Phi_{t_0}^t(z_0)) \tag{3.26}$$

$$\Phi_{t_0}^t(z_0) = z_0 + \int_{t_0}^{t} f\big(s, \Phi_{t_0}^s(z_0)\big)ds \tag{3.27}$$

Variational flow The derivative $w.r.t$ initial condition is called the *variational flow*, denoted as $D\Phi_{t_0}^t$, then it satisfies:

$$D\Phi_{t_0}^{t}(z_0) = \frac{d\Phi_{t_0}^{t}(z_0)}{dz_0}, \;\; D\Phi_{t_0}^{t_0} = I \tag{3.28}$$

$$D\Phi_{t_0}^{t_0+h} = I + O(h), \; if \; h \; is \; small. \tag{3.29}$$

From Eq. 3.25 and 3.29, using the chain rule, we have:

$$D\Phi_{t_0}^{t}(z_0) = \frac{d\Phi_{t_0}^{t}(z_0)}{dz_0} = \frac{d\Phi_{t_0+h}^{t}(\Phi_{t_0}^{t_0+h}(z_0))}{d\Phi_{t_0}^{t_0+h}(z_0)} \frac{d\Phi_{t_0}^{t_0+h}(z_0)}{dz_0} = D\Phi_{t_0+h}^{t} + O(h) \tag{3.30}$$

Local truncation error Denote the step function of a one-step ODE solver as $\psi_h(t, z)$, with step-size h starting from (t, z). Denote the local truncation error as:

$$L_h(t, z) = \psi_h(t, z) - \Phi_t^{t+h}(z) \tag{3.31}$$

For a solver of order p, the error is of order $O(h^{p+1})$, and can be written as

$$L_h(t, z) = h^{p+1} l(t, z) + O(h^{p+2}) \tag{3.32}$$

where l is some function of order $O(1)$.

Global error Denote the global error as $G(T)$ at time T, then it satisfies:

$$G(T) = z_N - \Phi_{t_0}^{T}(z_0) = \sum_{k=0}^{N-1} R_k \tag{3.33}$$

where

$$R_k = \Phi_{t_{k+1}}^{T}(z_{k+1}) - \Phi_{t_k}^{T}(z_k) \tag{3.34}$$

$$= \Phi_{t_{k+1}}^{T}\big(\Phi_{t_k}^{t_{k+1}}(z_k) + L_{h_k}(t_k, z_k)\big) - \Phi_{t_{k+1}}^{T}(\Phi_{t_k}^{t_{k+1}}(z_k)) \tag{3.35}$$

Lemma 3.7.0.2 (Approximation of R_k)**.**

$$R_k = D\Phi_{t_{k+1}}^T\big(\Phi_{t_k}^{t_{k+1}}(z_k)\big)L_{h_k}(t_k, z_k) + O(h_k^{2p+2}) \tag{3.36}$$

Lemma 3.7.0.2 can be viewed as a Taylor expansion of Eq. 3.35, with detailed proof in [111].

Lemma 3.7.0.3. *If* $L_h(t, y) = O(h^{p+1})$*, then* $G_h(T) = O(h^p)$

Proof for Lemma 3.7.0.3 is in [111].

Plug Eq. 3.7 and Eq. 3.30 into Eq. 3.36, we have

$$R_k = \big[D\Phi_{t_k}^T(z_k) + O(h_k)\big]L_{h_k}(t_k, z_k) + O(h_k^{2p+2}) \tag{3.37}$$

$$= \big[D\Phi_{t_k}^T(z_k) + O(h_k)\big]\big[h_k^{p+1}l(t_k, z_k) + O(h_k^{p+2})\big] + O(h_k^{2p+2}) \tag{3.38}$$

$$= h_k^{p+1}D\Phi_{t_k}^T(z_k)l(t_k, z_k) + O(h_k^{p+2}) \tag{3.39}$$

Plug Eq. 3.39 into Eq. 3.8, then we have:

$$G(T) = \sum_{k=0}^{N-1} R_k = \sum_{k=0}^{N-1}\big[h_k^{p+1}D\Phi_{t_k}^T(z_k)l(t_k, z_k) + O(h_k^{p+2})\big] \tag{3.40}$$

$$= \sum_{k=0}^{N-1}\big[h_k^{p+1}D\Phi_{t_k}^T(z_k)l(t_k, z_k)\big] + O(h_{max}^{p+1}) \tag{3.41}$$

Global error of the adjoint method If we solve an IVP forward-in-time from $t = 0$ to T, then take $z(T)$ as the initial condition, and solve it backward-in-time from T to 0, the

numerical error can be written as:

$$G(t_0 \to T \to t_0) = \sum_{k=0}^{N_t-1} \left[h_k^{p+1} D\Phi_{t_k}^T(z_k) l(t_k, z_k)\right] + \sum_{J=0}^{N_r-1} \left[(-h_j)^{p+1} D\Phi_T^{\tau_j}(\overline{z_j}) \overline{l(\tau_j, \overline{z_j})}\right] + O(h_{max}^{p+1}) \tag{3.42}$$

$$= G(t_0 \to T) + G(T \to t_0) + O(h^{p+1}) \tag{3.43}$$

where $G(t_0 \to T)$ represents the numerical error of forward-in-time (t_0 to T) solution (discretized at step k, denoted as z_k); and $G(T \to t_0)$ denotes the numerical error of reverse-in-time solution (T to t_0) (discretized at step j, denoted as $\overline{z_j}$). $G(t_0 \to T \to t_0)$ represents the error in reconstructed initial condition by the adjoint method. Note that generally z does not overlap with $\overline{z}$. The local error of forward-in-time and reverse-in-time numerical integration is represented as l and $\bar{l}$ respectively.

Although going backward is equivalent to a negative stepsize, which might cause the second term to have different signs compared to the first term in Eq. 3.42, we demonstrate that generally their sum cannot cancel.

For the ease of analysis, we assume the forward and reverse-in-time calculation are discretized at the same grid points, with a sufficiently small constant stepsize (For a variable-stepsize solver, we can modify it to a constant-stepsize solver, whose stepsize is the minimal step in variable-stepsize solver. With this modification, the constant stepsize solver

should be *no worse than* adaptive stepsize solver). Then Eq. 3.42 can be written as:

$$G(t_0 \to T \to t_0) = \sum_{k=0}^{N-1} \left[h_k^{p+1} D\Phi_{t_k}^T(z_k) l(t_k, z_k)\right] + \sum_{k=0}^{N-1} \left[(-h_k)^{p+1} D\Phi_T^{t_k}(\overline{z_k}) \overline{l(t_k, \overline{z_k})}\right] + O(h_{max}^{p+1}) \tag{3.44}$$

$$= \sum_{k=0}^{N-1} \left[h_k^{p+1} D\Phi_{t_k}^T(z_k) l(t_k, z_k) + (-h_k)^{p+1} D\Phi_T^{t_k}(\overline{z_k}) \overline{l(t_k, \overline{z_k})}\right] + O(h^{p+1}) \tag{3.45}$$

If the stepsize is sufficiently small, we can assume

$$z_k = \overline{z_k} + O(h) \tag{3.46}$$

$$D\Phi_T^{t_k}(z_k) = D\Phi_T^{t_k}(\overline{z_k}) + O(h) \tag{3.47}$$

$$l(t_k, z_k) = \overline{l(t_k, \overline{z_k})} + O(h) \tag{3.48}$$

Assume the existence and uniqueness conditions are satisfied on $t \in [0, T]$, so $\Phi_{t_0}^T$ defines a homeomorphism, hence:

$$D\Phi_T^{t_k} = (D\Phi_{t_k}^T)^{-1} \tag{3.49}$$

Plug Eq. 3.46 to Eq. 3.49 into Eq. 3.45, we have

$$G(t_0 \to T \to t_0) = \sum_{k=0}^{N-1} h^{p+1} l(t_k, z_k) e_k + O(h^{p+1}) \tag{3.50}$$

$$e_k = D\Phi_{t_k}^T(z_k) + (-1)^{p+1} (D\Phi_{t_k}^T(z_k))^{-1} \tag{3.51}$$

Reverse accuracy for all t_0 requires $e_k = 0$ for all k. If p is odd, then the two terms in e_k are the same sign, and thus cannot cancel to 0; if p is even, then $e_k = 0$ requires $D\Phi_{t_k}^T(z_k) = D\Phi_{t_k}^T(z_k)^{-1} = I$, which is generally not satisfied with a trained network

(otherwise the network is an identity function with a constant bias).

In short, solving an IVP from t_0 to T with $z(0) = z_0$, then taking $z(T)$ as initial condition and solving it from T to t_0 and getting $\overline{z(0)}$, generally $z(0) \neq \overline{z(0)}$ because of numerical errors.

3.7.2 Algorithm of ALF

For the ease of reading, we write the algorithm for ψ in ALF below, which is the same as Algo. 4 in the main paper, but uses slightly different notations for the ease of analysis.

Algorithm 7: Forward of ψ in ALF

Input $(\widehat{z_{in}}, \widehat{v_{in}}, s_{in}, h) = (\widehat{z_0}, \widehat{v_0}, s_0, h)$ where s_0 is current time, $\widehat{z_0}$ and $\widehat{v_0}$ are correponding values at time s_0; stepsize h.

Forward

$$s_1 = s_0 + h/2 \tag{3.52}$$
$$\widehat{z_1} = \widehat{z_0} + \widehat{v_0} \times h/2 \tag{3.53}$$
$$\widehat{v_1} = f(\widehat{z_1}, s_1) \tag{3.54}$$
$$\widehat{v_2} = \widehat{v_1} + (\widehat{v_1} - \widehat{v_0}) \tag{3.55}$$
$$\widehat{z_2} = \widehat{z_1} + \widehat{v_2} \times h/2 \tag{3.56}$$
$$s_2 = s_1 + h/2 \tag{3.57}$$

Output $(\widehat{z_{out}}, \widehat{v_{out}}, s_{out}, h) = (\widehat{z_2}, \widehat{v_2}, s_2, h)$

For simplicity, we can re-write the forward of ALF as

$$\begin{bmatrix} \widehat{z_2} \\ \\ \widehat{v_2} \end{bmatrix} = \begin{bmatrix} \widehat{z_0} + hf(\widehat{z_0} + \frac{h}{2}\widehat{v_0}, s_0 + \frac{h}{2}) \\ \\ 2f(\widehat{z_0} + \frac{h}{2}\widehat{v_0}, s_0 + \frac{h}{2}) - \widehat{v_0} \end{bmatrix} \tag{3.58}$$

Similarly, the inverse of ALF can be written as

$$\begin{bmatrix} \widehat{z}_0 \\ \widehat{v}_0 \end{bmatrix} = \begin{bmatrix} \widehat{z}_2 - hf(\widehat{z}_2 - \frac{h}{2}\widehat{v}_2, s_2 - \frac{h}{2}) \\ \\ 2f(\widehat{z}_2 - \frac{h}{2}\widehat{v}_2, s_2 - \frac{h}{2}) - \widehat{v}_2 \end{bmatrix} \tag{3.59}$$

3.7.3 Expansion of total derivative

For an ODE of the form

$$\frac{\mathrm{d}z(t)}{\mathrm{d}t} = f(z(t), t) \tag{3.60}$$

We have:

$$\frac{\mathrm{d}^2 z(t)}{dt^2} = \frac{\mathrm{d}}{\mathrm{d}t} f(z(t), t) = \frac{\partial f(z(t), t)}{\partial t} + \frac{\partial f(z(t), t)}{\partial z} \frac{\mathrm{d}z(t)}{\mathrm{d}t} \tag{3.61}$$

For the ease of notation, we re-write Eq. 3.61 as

$$\frac{\mathrm{d}^2 z(t)}{dt^2} = f_t + f_z f \tag{3.62}$$

where f_t and f_z represents the partial derivative of f *w.r.t* t and z respectively.

3.7.4 Local truncation error of ALF

Theorem 3.7.1 (Theorem 3.4.1 in the main paper)**.** *For a single step in ALF with stepsize h, the local truncation error of z is $O(h^3)$, and the local truncation errof of v is $O(h^2)$.*

Proof. Under the same notation as Algo. 7, denote the ground-truth state of z and v starting from $(\widehat{z}_0, s_0)$ as $\widetilde{z}$ and $\widetilde{v}$ respectively. Then the local truncation error is

$$L_z = \widetilde{z}(s_0 + h) - \widehat{z}_2, \;\; L_v = \widetilde{v}(s_0 + h) - \widehat{v}_2 \tag{3.63}$$

We estimate L_z and L_v in terms of polynomial of h.

Under mild assumptions that f is smooth up to 2nd order almost everywhere (this is typically satisfied with neural networks with bounded weights), hence Taylor expansion is meaningful for f. By Eq. 3.62, the Taylor expansion of $\widetilde{z}$ around point $(\widehat{z_0}, \widehat{v_0}, s_0)$ is

$$\widetilde{z}(s_0 + h) = \widehat{z_0} + h\frac{\mathrm{d}z}{dt} + \frac{h^2}{2}\frac{\mathrm{d}^2 z}{dt^2} + O(h^3) \tag{3.64}$$

$$= \widehat{z_0} + hf(\widehat{z_0}, s_0) + \frac{h^2}{2}\Big(f_t(\widehat{z_0}, s_0) + f_z(\widehat{z_0}, s_0)f(\widehat{z_0}, s_0)\Big) + O(h^3) \tag{3.65}$$

Next, we analyze accuracy of the numerical approximation. For simplicity, we directly analyze Eq. 3.58 by performing Taylor Expansion on f.

$$f(\widehat{z_0} + \frac{h}{2}\widehat{v_0}, s_0 + \frac{h}{2}) = f(\widehat{z_0}, s_0) + \frac{h}{2}f_t(\widehat{z_0}, s_0) + \frac{h\widehat{v_0}}{2}f_z(\widehat{z_0}, s_0) + O(h^2) \tag{3.66}$$

$$\widehat{z_2} = \widehat{z_0} + hf(\widehat{z_0} + \frac{h}{2}\widehat{v_0}, s_0 + \frac{h}{2}) \tag{3.67}$$

Plug Eq. 3.65, Eq. 3.66 and E.q. 3.67 into the definition of L_z, we get

$$L_z = \widetilde{z}(s_0 + h) - \widehat{z_2} \tag{3.68}$$

$$= \Big[\widehat{z_0} + hf(\widehat{z_0}, s_0) + \frac{h^2}{2}\Big(f_t(\widehat{z_0}, s_0) + f_z(\widehat{z_0}, s_0)f(\widehat{z_0}, s_0)\Big)\Big]$$

$$- \Big[\widehat{z_0} + h\Big(f(\widehat{z_0}, s_0) + \frac{h}{2}f_t(\widehat{z_0}, s_0) + \frac{h\widehat{v_0}}{2}f_z(\widehat{z_0}, s_0)\Big)\Big] + O(h^3) \tag{3.69}$$

$$= \frac{h^2}{2}f_z(\widehat{z_0}, s_0)\Big(f(\widehat{z_0}, s_0) - \widehat{v_0}\Big) + O(h^3) \tag{3.70}$$

Therefore, if $\Big|f(\widehat{z_0}, s_0) - \widehat{v_0}\Big|$ is of order $O(1)$, L_z is of order $O(h^2)$; if $\Big|f(\widehat{z_0}, s_0) - \widehat{v_0}\Big|$ is of order $O(h)$ or smaller, then L_z is of order $O(h^3)$. Specifically, at the start time of integration, we have $\Big|f(\widehat{z_0}, s_0) - \widehat{v_0} = 0\Big|$, by induction, L_z at end time is $O(h^3)$.

Next we analyze the local truncation error in v, denoted as L_v. Denote the ground truth

as $\widetilde{v}(t_0 + h)$, we have

$$\widetilde{v}(s_0 + h) = f\big(\widetilde{z}(s_0 + h), s_0 + h\big) \tag{3.71}$$

$$= f(\widehat{z}_0, s_0) + hf_t(\widehat{z}_0, s_0) + \big(\widetilde{z}(s_0 + h) - \widehat{z}_0\big) f_z(\widehat{z}_0, s_0) + O(h^2) \tag{3.72}$$

Next we analyze the error in the numerical approximation. Plug Eq. 3.66 into Eq. 3.58,

$$\widehat{v}_2 = 2f(\widehat{z}_0 + \frac{h}{2}\widehat{v}_0, s_0 + \frac{h}{2}) - \widehat{v}_0 \tag{3.73}$$

$$= f(\widehat{z}_0, s_0) + \big(f(\widehat{z}_0, s_0) - \widehat{v}_0\big) + hf_t(\widehat{z}_0, s_0) + h\widehat{v}_0 f_z(\widehat{z}_0, s_0) + O(h^2) \tag{3.74}$$

From Eq. 3.65, Eq. 3.72 and Eq. 3.74, we have

$$L_v = \widetilde{v}(s_0 + h) - \widehat{v}_2 \tag{3.75}$$

$$= \Big(f(\widehat{z}_0, s_0) - \widehat{v}_0\Big) + \Big(\widetilde{z}(s_0 + h) - \big(\widehat{z}_0 + h\widehat{v}_0\big)\Big) f_z(\widehat{z}_0, s_0) + O(h^2) \tag{3.76}$$

$$= \Big(f(\widehat{z}_0, s_0) - \widehat{v}_0\Big) + h\Big(f(\widehat{z}_0, s_0) - \widehat{v}_0\Big) f_z(\widehat{z}_0, s_0) + O(h^2) \tag{3.77}$$

The last equation is derived by plugging in Eq. 3.65. Note that Eq. 3.77 holds for every single step forward in time, and at the start time of integration, we have $\big|f(\widehat{z}_0, s_0) - \widehat{v}_0\big| = 0$ due to our initialization as in Sec. 3.4.2 of the main paper. Therefore, by induction, L_v is of order $O(h^2)$ for consecutive steps. □

3.7.5 Stability analysis for ALF

Lemma 3.7.1.1. *For a matrix of the form* $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$*, if* A, B, C, D *are square matrices of the same shape, and* $CD = DC$*, then we have* $\det \begin{bmatrix} A & B \\ C & D \end{bmatrix} = \det(AD - BC)$

Proof. See [144] for a detailed proof. □

Theorem 3.7.2. *For ALF integrator with stepsize h, if $h\sigma_i$ is 0 or is imaginary with norm no larger than 1, where σ_i is the i-th eigenvalue of the Jacobian $\frac{\partial f}{\partial z}$, then the solver is on the critical boundary of A-stability; otherwise, the solver is not A-stable.*

Proof. A solver is A-stable is equivalent to the eigenvalue of the numerical forward has a norm below 1. We calculate the eigenvalue of ψ below.

For the function defined by Eq. 3.58, the Jacobian is

$$J = \begin{bmatrix} \frac{\partial \widehat{z_2}}{\partial z_0} & \frac{\partial \widehat{z_2}}{\partial \widehat{v_0}} \\ \\ \frac{\partial \widehat{v_2}}{\partial z_0} & \frac{\partial \widehat{v_2}}{\partial \widehat{v_0}} \end{bmatrix} = \begin{bmatrix} I + h\frac{\partial f}{\partial z} & \frac{h^2}{2}\frac{\partial f}{\partial z} \\ \\ 2 \times \frac{\partial f}{\partial z} & h\frac{\partial f}{\partial z} - I \end{bmatrix} \tag{3.78}$$

We determine the eigenvalue of J by solving the equation

$$\det(J - \lambda I) = \begin{bmatrix} h\frac{\partial f}{\partial z} + (1-\lambda)I & \frac{h^2}{2}\frac{\partial f}{\partial z} \\ \\ 2 \times \frac{\partial f}{\partial z} & h\frac{\partial f}{\partial z} - (1+\lambda)I \end{bmatrix} = 0 \tag{3.79}$$

It's trivial to check J satisfies conditions for Lemma 3.7.1.1.Therefore, we have

$$\det(J - \lambda I) = \det\Big[\Big(h\frac{\partial f}{\partial z} + (1-\lambda)I\Big)\Big(h\frac{\partial f}{\partial z} - (1+\lambda)I\Big) - \Big(\frac{h^2}{2}\frac{\partial f}{\partial z}\Big)\Big(2 \times \frac{\partial f}{\partial z}\Big)\Big] \tag{3.80}$$

$$= \det\Big[-2\lambda h\frac{\partial f}{\partial z} + (\lambda^2 - 1)I\Big] \tag{3.81}$$

Suppose the eigen-decompostion of $\frac{\partial f}{\partial z}$ can be written as

$$\frac{\partial f}{\partial z} = \Lambda \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \dots & \\ & & & \sigma_N \end{bmatrix} \Lambda^{-1} \tag{3.82}$$

Note that $I = \Lambda I \lambda^{-1}$, hence we have

$$\det(J - \lambda I) = \det\ \Lambda \left\{ -2\lambda h \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \dots & \\ & & & \sigma_N \end{bmatrix} + (\lambda^2 - 1) I \right\} \Lambda^{-1} \tag{3.83}$$

$$= \prod_{i=1}^{N} (\lambda^2 - 2h\sigma_i \lambda - 1) \tag{3.84}$$

Hence the eigenvalues are

$$\lambda_{i\pm} = h\sigma_i \pm \sqrt{h^2 \sigma_i^2 + 1} \tag{3.85}$$

A-stability requires $|\lambda_{i\pm}| < 1, \forall i$, and has no solution.

The critical boundary is $|\lambda_{i\pm}| = 1$, the solution is: $h\sigma_i$ is 0 or on the imaginary line with norm no larger than 1. □

3.7.6 Damped ALF

Algorithm 8: Forward of ψ in Damped ALF ($\eta \in (0, 1]$)

Input $(\widehat{z_{in}}, \widehat{v_{in}}, s_{in}, h) = (\widehat{z}_0, \widehat{v}_0, s_0, h)$ where s_0 is current time, $\widehat{z}_0$ and $\widehat{v}_0$ are correponding values at time s_0; stepsize h.

Forward

$$s_1 = s_0 + h/2 \tag{3.86}$$
$$\widehat{z}_1 = \widehat{z}_0 + \widehat{v}_0 \times h/2 \tag{3.87}$$
$$\widehat{v}_1 = f(\widehat{z}_1, s_1) \tag{3.88}$$
$$\widehat{v}_2 = \widehat{v}_0 + 2\eta(\widehat{v}_1 - \widehat{v}_0) \tag{3.89}$$
$$\widehat{z}_2 = \widehat{z}_1 + \widehat{v}_2 \times h/2 \tag{3.90}$$
$$s_2 = s_1 + h/2 \tag{3.91}$$

Output $(\widehat{z_{out}}, \widehat{v_{out}}, s_{out}, h) = (\widehat{z}_2, \widehat{v}_2, s_2, h)$

Algorithm 9: ψ^{-1} (Inverse of ψ) in Damped ALF ($\eta \in (0, 1]$)

Input $(\widehat{z_{out}}, \widehat{v_{out}}, s_{out}, h)$ where s_{out} is current time, $\widehat{z_{out}}$ and $\widehat{v_{out}}$ are corresponding values at s_{out}, h is stepsize.

Inverse

$$(\widehat{z}_2, \widehat{v}_2, s_2, h) = (\widehat{z_{out}}, \widehat{v_{out}}, s_{out}, h) \tag{3.92}$$
$$s_1 = s_2 - h/2 \tag{3.93}$$
$$\widehat{z}_1 = z_2 - \widehat{v}_2 \times h/2 \tag{3.94}$$
$$\widehat{v}_1 = f(\widehat{z}_1, s_1) \tag{3.95}$$
$$\widehat{v}_0 = (\widehat{v}_2 - 2\eta\widehat{v}_1)/(1 - 2\eta) \tag{3.96}$$
$$\widehat{z}_0 = \widehat{z}_1 - \widehat{v}_0 \times h/2 \tag{3.97}$$
$$s_0 = s_1 - h/2 \tag{3.98}$$

Output $(\widehat{z_{in}}, \widehat{v_{in}}, s_{in}, h) = (\widehat{z}_0, \widehat{v}_0, s_0, h)$

The main difference between ALF and Damped ALF is marked in blue in Algo. 8. In

ALF, the update of $\widehat{v_2}$ is $\widehat{v_2} = (\widehat{v_1} - \widehat{v_0}) + \widehat{v_1} = 2(\widehat{v_1} - \widehat{v_0}) + \widehat{v_0}$; while in Damped ALF, the update is scaled by a factor η between 0 and 1, so the update is $\widehat{v_2} = 2\eta(\widehat{v_1} - \widehat{v_0}) + \widehat{v_0}$. When $\eta = 1$, Damped ALF reduces to ALF.

Similar to Sec. 3.7.2, we can write the forward as For simplicity, we can re-write the forward of ALF as

$$\begin{bmatrix} \widehat{z_2} \\ \\ \widehat{v_2} \end{bmatrix} = \begin{bmatrix} \widehat{z_0} + \eta h f(\widehat{z_0} + \frac{h}{2}\widehat{v_0}, s_0 + \frac{h}{2}) + (1-\eta)h\widehat{v_0} \\ \\ 2\eta f(\widehat{z_0} + \frac{h}{2}\widehat{v_0}, s_0 + \frac{h}{2}) + (1-2\eta)\widehat{v_0} \end{bmatrix} \tag{3.99}$$

Similarly, the inverse of ALF can be written as

$$\begin{bmatrix} \widehat{z_0} \\ \\ \widehat{v_0} \end{bmatrix} = \begin{bmatrix} \widehat{z_2} - h\frac{1-\eta}{1-2\eta}\widehat{v_2} + h\frac{\eta}{1-2\eta} f(\widehat{z_2} - \frac{h}{2}\widehat{v_2}, s_2 - \frac{h}{2}) \\ \\ \frac{1}{1-2\eta}\widehat{v_2} - \frac{2\eta}{1-2\eta} f(\widehat{z_2} - \frac{h}{2}\widehat{v_2}, s_2 - \frac{h}{2}) \end{bmatrix} \tag{3.100}$$

Theorem 3.7.3. *For a single step in Damped ALF with stepsize h, the local truncation error of z is $O(h^2)$, and the local truncation errof of v is $O(h)$.*

Proof. The proof is similar to Thm. 3.7.1. By similar calculations using the Taylor Expansion in Eq. 3.66 and Eq. 3.65, we have

$$\begin{aligned} \widehat{z_2} - \tilde{z}(s_0 + h) &= (1-\eta)h\widehat{v_0} + h\eta\Big[f(\widehat{z_0}, s_0) + \frac{h}{2} f_t(\widehat{z_0}, s_0) + \frac{h\widehat{v_0}}{2} f_z(\widehat{z_0}, s_0)\Big] \\ &\quad - h\Big[f(\widehat{z_0}, s_0) + \frac{h}{2} f_t \widehat{z_0}, s_0 + \frac{h}{2} f_z(\widehat{z_0}, s_0) f(\widehat{z_0}, s_0)\Big] + O(h^2) \qquad (3.101) \\ &= (1-\eta)h\Big(\widehat{v_0} - f(\widehat{z_0}, s_0)\Big) + \frac{\eta - 1}{2} h^2 f_t(\widehat{z_0}, s_0) \\ &\quad + \frac{h^2}{2}\Big(\eta\widehat{v_0} - f(\widehat{z_0}, s_0)\Big) f_z(\widehat{z_0}, s_0) + O(h^2) \qquad (3.102) \end{aligned}$$

Using Eq. 3.72, Eq. 3.66 and Eq. 3.65, we have

$$\begin{aligned}\tilde{v_2} - \widehat{v_2} &= (1-2\eta)\widehat{v_0} + (2\eta-1)f(\widehat{z_0}, s_0) + (1-\eta)hf_t(\widehat{z_0}, s_0) \\ &\quad + \Big(\tilde{z}(s_0+h) - \widehat{z_0} - \eta h\widehat{v_0}\Big) f_z(\widehat{z_0}, s_0) + O(h^2) \qquad (3.103) \\ &= (2\eta-1)\big[f(\widehat{z_0}, s_0) - \widehat{z_0}\big] + (1-\eta)hf_t(\widehat{z_0}, s_0) \\ &\quad + \eta\Big[hf(\widehat{z_0}, s_0) - h\widehat{v_0}\Big] f_z(\widehat{z_0}, s_0) + O(h^2) \qquad (3.104)\end{aligned}$$

Note that when $\eta = 1$, Eq. 3.102 reduces to Eq. 3.70, and Eq. 3.104 reduces to Eq. 3.77. By initialization, we have $|f(\widehat{z_0}, s_0) - \widehat{v_0}| = 0$ at initial time, hence by induction, the local truncation error for z is $O(h^2)$; the local truncation error for v is $O(h)$ when $\eta < 1$, and is $O(h^2)$ when $\eta = 1$. □

Theorem 3.7.4 (Theorem 3.4.2 in the main paper)**.** *For Dampled ALF integrator with stepsize h, where σ_i is the i-th eigenvalue of the Jacobian $\frac{\partial f}{\partial z}$, then the solver is A-stable if* $\Big|1 + \eta(h\sigma - 1) \pm \sqrt{\eta\big[2h\sigma_i + \eta(h\sigma_i - 1)^2\big]}\Big| < 1,\ \forall i$.

Proof. The Jacobian of the forward-pass of a single step damped ALF is

$$J = \begin{bmatrix} I + \eta h\frac{\partial f}{\partial z} & (1-\eta)hI + \eta\frac{h^2}{2}\frac{\partial f}{\partial z} \\ \\ 2\eta\frac{\partial f}{\partial z} & \eta h\frac{\partial f}{\partial z} + (1-2\eta)I \end{bmatrix} \qquad (3.105)$$

when $\eta = 1$, J reduces to Eq. 3.78. We can determine the eigenvalue of J using similar

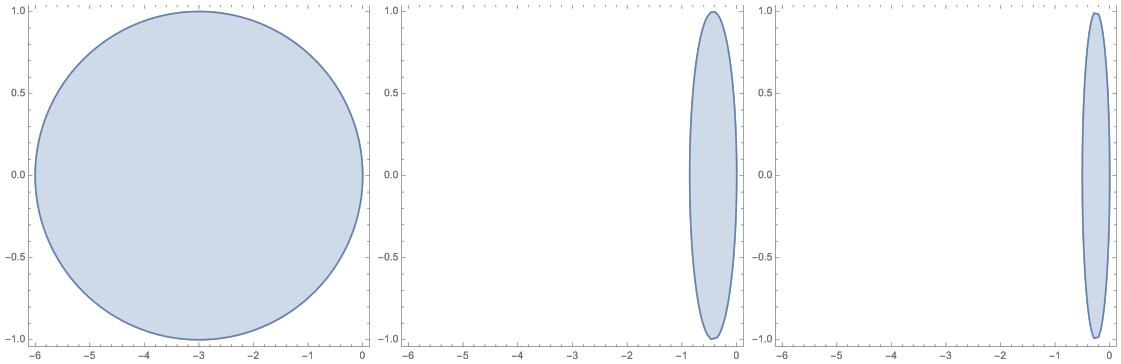


Figure 3.7: Region of A-stability for eigenvalue on the imaginary plane for damped ALF. From left to right, the region of stability for $\eta = 0.25$, $\eta = 0.7$,$\eta = 0.8$ respectively. As η increases to 1, the area of stability region decreases.

techniques. Assume the eigenvalues for $\frac{\partial f}{\partial z}$ are $\{\sigma_i\}$, then we have

$$\det(J - \lambda I) = \det \begin{bmatrix} (1-\lambda)I + \eta h \frac{\partial f}{\partial z} & (1-\eta)hI + \eta \frac{h^2}{2} \frac{\partial f}{\partial z} \\ 2\eta \frac{\partial f}{\partial z} & \eta h \frac{\partial f}{\partial z} + (1 - 2\eta - \lambda)I \end{bmatrix} \tag{3.106}$$

$$= \det \Big[\Big((1-\lambda)I + \eta h \frac{\partial f}{\partial z} \Big) \Big(\eta h \frac{\partial f}{\partial z} + (1 - 2\eta - \lambda)I \Big) - \Big((1-\eta)hI + \eta \frac{h^2}{2} \frac{\partial f}{\partial z} \Big) \Big(2\eta \frac{\partial f}{\partial z} \Big) \Big] \tag{3.107}$$

$$= \prod_{i=1}^{N} \Big[1 + \eta(h\sigma_i - 1) \pm \sqrt{\eta \big[2h\sigma_i + \eta(h\sigma_i - 1)^2 \big]} \Big] \tag{3.108}$$

when $\eta < 1$, it's easy to check that $\Big| 1 + \eta(h\sigma_i - 1) \pm \sqrt{\eta \big[2h\sigma_i + \eta(h\sigma_i - 1)^2 \big]} \Big| < 1$ has non-empty solutions for $h\sigma$. □

For a quick validation, we plot the region of A-stability on the imaginary plane for a single eigenvalue in Fig. 3.7. As η increases, the area of stability decreases. When $\eta = 1$, the system is no-where A-stable, and the boundary for A-stability is on the imaginary axis $[-i, i]$ where i is the imaginary unit.

Chapter 4

AdaBelief optimizer: scale stepsize by the belief in observed gradients

4.1 Introduction

Modern neural networks are typically trained with first-order gradient methods, which can be broadly categorized into two branches: the accelerated stochastic gradient descent (SGD) family [129], such as Nesterov accelerated gradient (NAG) [108], SGD with momentum [148] and heavy-ball method (HB) [120]; and the adaptive learning rate methods, such as Adagrad [37], AdaDelta [170], RMSProp [49] and Adam [74]. SGD methods use a global learning rate for all parameters, while adaptive methods compute an individual learning rate for each parameter.

Compared to the SGD family, adaptive methods typically converge fast in the early training phases, but have poor generalization performance [165, 93]. Recent progress tries to combine the benefits of both, such as switching from Adam to SGD either with a hard schedule as in SWATS [69], or with a smooth transition as in AdaBound [92]. Other modifications of Adam are also proposed: AMSGrad [127] fixes the error in convergence analysis of Adam, Yogi [169] considers the effect of minibatch size, MSVAG [5] dis-

sects Adam as sign update and magnitude scaling, RAdam [88] rectifies the variance of learning rate, Fromage [10] controls the distance in the function space, and AdamW [90] decouples weight decay from gradient descent. Although these modifications achieve better accuracy compared to Adam, their generalization performance is typically worse than SGD on large-scale datasets such as ImageNet [136]; furthermore, compared with Adam, many optimizers are empirically unstable when training generative adversarial networks (GAN) [46].

To solve the problems above, we propose "AdaBelief", which can be easily modified from Adam. Denote the observed gradient at step t as g_t and its exponential moving average (EMA) as m_t. Denote the EMA of g_t^2 and $(g_t - m_t)^2$ as v_t and s_t, respectively. m_t is divided by $\sqrt{v_t}$ in Adam, while it is divided by $\sqrt{s_t}$ in AdaBelief. Intuitively, $\frac{1}{\sqrt{s_t}}$ is the "belief" in the observation: viewing m_t as the prediction of the gradient, if g_t deviates much from m_t, we have weak belief in g_t, and take a small step; if g_t is close to the prediction m_t, we have a strong belief in g_t, and take a large step. We validate the performance of AdaBelief with extensive experiments. Our contributions can be summarized as:

- We propose AdaBelief, which can be easily modified from Adam without extra parameters. AdaBelief has three properties: (1) fast convergence as in adaptive gradient methods, (2) good generalization as in the SGD family, and (3) training stability in complex settings such as GAN.
- We theoretically analyze the convergence property of AdaBelief in both convex optimization and non-convex stochastic optimization.
- We show that an asynchronous version of AdaBelief achieves both a weak convergence condition and the oracle convergence speed for first-order gradient optimizers in the stochastic non-convex optimization.
- We validate the performance of AdaBelief with extensive experiments: AdaBelief achieves

fast convergence as Adam and good generalization as SGD in image classification tasks on CIFAR and ImageNet; AdaBelief outperforms other methods in language modeling; in the training of a W-GAN [4], compared to a well-tuned Adam optimizer, AdaBelief significantly improves the quality of generated images, while several recent adaptive optimizers fail the training.

4.2 Methods

4.2.1 Details of AdaBelief Optimizer

Notations By the convention in [74], we use the following notations:

- $f(\theta) \in \mathbb{R}, \theta \in \mathbb{R}^d$: f is the loss function to minimize, θ is the parameter in $\mathbb{R}^d$
- $\prod_{\mathcal{F},M}(y) = \text{argmin}_{x \in \mathcal{F}} ||M^{1/2}(x-y)||$: projection of y onto a convex feasible set $\mathcal{F}$
- g_t: the gradient and step t
- m_t: exponential moving average (EMA) of g_t
- v_t, s_t: v_t is the EMA of g_t^2, s_t is the EMA of $(g_t - m_t)^2$
- α, ϵ: α is the learning rate, default is 10^{-3}; ϵ is a small number, typically set as 10^{-8}
- β_1, β_2: smoothing parameters, typical values are $\beta_1 = 0.9, \beta_2 = 0.999$
- β_{1t}, β_{2t} are the momentum for m_t and v_t respectively at step t, and typically set as constant (e.g. $\beta_{1t} = \beta_1, \beta_{2t} = \beta_2, \forall t \in \{1, 2, ...T\}$

Algorithm 10: Adam	**Algorithm 11:** AdaBelief
Initialize θ_0, $m_0 \leftarrow 0$, $v_0 \leftarrow 0$,	**Initialize** θ_0, $m_0 \leftarrow 0$, $s_0 \leftarrow 0$,
$t \leftarrow 0$	$t \leftarrow 0$
While θ_t not converged	**While** θ_t not converged
$t \leftarrow t + 1$	$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$	$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$	$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$	$s_t \leftarrow \beta_2 s_{t-1} + (1-\beta_2)(g_t - m_t)^2$
Bias Correction	**Bias Correction**
$\widehat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$, $\widehat{v_t} \leftarrow \frac{v_t}{1-\beta_2^t}$	$\widehat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$, $\widehat{s_t} \leftarrow \frac{s_t+\epsilon}{1-\beta_2^t}$
Update	**Update**
$\theta_t \leftarrow \prod_{\mathcal{F}, \sqrt{\widehat{v_t}}} \left(\theta_{t-1} - \frac{\alpha \widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon} \right)$	$\theta_t \leftarrow \prod_{\mathcal{F}, \sqrt{\widehat{s_t}}} \left(\theta_{t-1} - \frac{\alpha \widehat{m_t}}{\sqrt{\widehat{s_t}} + \epsilon} \right)$

Comparison with Adam Adam and AdaBelief are summarized in Algo. 10 and Algo. 14, where all operations are element-wise, with differences marked in blue. Note that no extra parameters are introduced in AdaBelief. Specifically, in Adam, the update direction is $m_t/\sqrt{v_t}$, where v_t is the EMA of g_t^2; in AdaBelief, the update direction is $m_t/\sqrt{s_t}$, where s_t is the EMA of $(g_t - m_t)^2$. Intuitively, viewing m_t as the prediction of g_t, AdaBelief takes a large step when observation g_t is close to prediction m_t, and a small step when the observation greatly deviates from the prediction. $\widehat{\cdot}$ represents bias-corrected value. Note that an extra ϵ is added to s_t during bias-correction, in order to better match the assumption that s_t is bouded below (the lower bound is at leat ϵ). For simplicity, we omit the bias correction step in theoretical analysis.

4.2.2 Intuitive explanation for benefits of AdaBelief

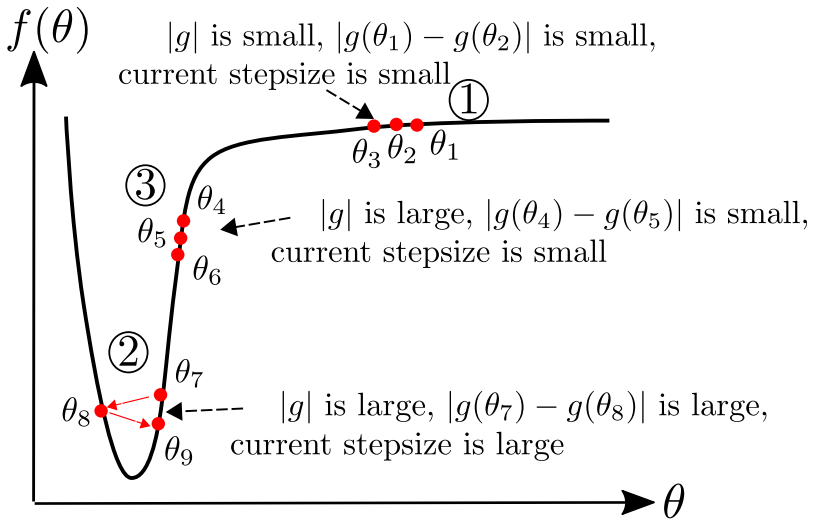


Figure 4.1: An ideal optimizer considers curvature of the loss function, instead of taking a large (small) step where the gradient is large (small) [154].

AdaBelief uses curvature information Update formulas for SGD, Adam and AdaBelief are:

$$\Delta\theta_t^{SGD} = -\alpha m_t, \tag{4.1}$$

$$\Delta\theta_t^{Adam} = -\alpha m_t / \sqrt{v_t},$$

$$\Delta\theta_t^{AdaBelief} = -\alpha m_t / \sqrt{s_t} \tag{4.2}$$

Note that we name α as the "learning rate" and $|\Delta\theta_t^i|$ as the "stepsize" for the ith parameter. With a 1D example in Fig. 4.1, we demonstrate that AdaBelief uses the curvature of loss functions to improve training as summarized in Table 4.1, with a detailed description below:

(1) In region ① in Fig. 4.1, the loss function is flat, hence the gradient is close to 0. In this case, an ideal optimizer should take a large stepsize. The stepsize of SGD is proportional to the EMA of the gradient, hence is small in this case; while both Adam and AdaBelief take a large stepsize, because the denominator ($\sqrt{v_t}$ and $\sqrt{s_t}$) is a small value.

(2) In region ②, the algorithm oscillates in a "steep and narrow" valley, hence both

Table 4.1: Comparison of optimizers in various cases in Fig. 4.1. "S" and "L" represent "small" and "large" stepsize, respectively. $|\Delta\theta_t|_{ideal}$ is the stepsize of an ideal optimizer. Note that only AdaBelief matches the behaviour of an ideal optimizer in all three cases.

	Case 1			Case 2			Case 3		
$\lvert g_t\rvert$, v_t	S			L			L		
$\lvert g_t - g_{t-1}\rvert$, s_t	S			L			S		
$\lvert\Delta\theta_t\rvert_{ideal}$	L			S			L		
$\lvert\Delta\theta_t\rvert$	SGD	Adam	AdaBelief	SGD	Adam	AdaBelief	SGD	Adam	AdaBelief
	S	L	L	L	S	S	L	S	L

$|g_t|$ and $|g_t - g_{t-1}|$ is large. An ideal optimizer should decrease its stepsize, while SGD takes a large step (proportional to m_t). Adam and AdaBelief take a small step because the denominator ($\sqrt{s_t}$ and $\sqrt{v_t}$) is large.

(3) In region ③, we demonstrate AdaBelief's advantage over Adam in the "large gradient, small curvature" case. In this case, $|g_t|$ and v_t are large, but $|g_t - g_{t-1}|$ and s_t are small; this could happen because of a small learning rate α. In this case, an ideal optimizer should increase its stepsize. SGD uses a large stepsize ($\sim \alpha|g_t|$); in Adam, the denominator $\sqrt{v_t}$ is large, hence the stepsize is small; in AdaBelief, denominator $\sqrt{s_t}$ is small, hence the stepsize is large as in an ideal optimizer.

To sum up, AdaBelief scales the update direction by the change in gradient, which is related to the Hessian. Therefore, AdaBelief considers curvature information and performs better than Adam.

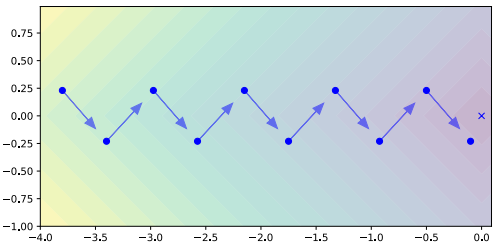
AdaBelief considers the sign of gradient in denominator We show the advantages of AdaBelief with a 2D example in this section, which gives us more intuition for high dimensional cases. In Fig. 4.2, we consider the loss function: $f(x, y) = |x| + |y|$. Note that in this simple problem, the gradient in each axis can only take $\{1, -1\}$. Suppose the start point is near the $x-$axis, e.g. $y_0 \approx 0, x_0 \ll 0$. Optimizers will oscillate in the y direction, and keep increasing in the x direction.

Suppose the algorithm runs for a long time (t is large), so the bias of EMA ($\beta_1^t \mathbb{E} g_t$) is

small:

$$m_t = EMA(g_0, g_1, ...g_t) \approx \mathbb{E}(g_t), \;\; m_{t,x} \approx \mathbb{E}g_{t,x} = 1, \;\; m_{t,y} \approx \mathbb{E}g_{t,y} = 0 \tag{4.3}$$

$$v_t = EMA(g_0^2, g_1^2, ...g_t^2) \approx \mathbb{E}(g_t^2), \;\; v_{t,x} \approx \mathbb{E}g_{t,x}^2 = 1, \;\; v_{t,y} \approx \mathbb{E}g_{t,y}^2 = 1. \tag{4.4}$$



	Step	1	2	3	4	5
	g_x	1	1	1	1	1
	g_y	-1	1	-1	1	-1
Adam	v_x	1	1	1	1	1
	v_y	1	1	1	1	1
AdaBelief	s_x	0	0	0	0	0
	s_y	1	1	1	1	1

Figure 4.2: *Left:* Consider $f(x, y) = |x| + |y|$. Blue vectors represent the gradient, and the cross represents the optimal point. The optimizer oscillates in the y direction, and keeps moving forward in the x direction. *Right:* Optimization process for the example on the left. Note that denominator $\sqrt{v_{t,x}} = \sqrt{v_{t,y}}$ for Adam, hence the same stepsize in x and y direction; while $\sqrt{s_{t,x}} < \sqrt{s_{t,y}}$, hence AdaBelief takes a large step in the x direction, and a small step in the y direction.

In practice, the bias correction step will further reduce the error between the EMA and its expectation if g_t is a stationary process [74]. Note that:

$$s_t = EMA\big((g_0 - m_0)^2, ...(g_t - m_t)^2\big) \approx \mathbb{E}\big[(g_t - \mathbb{E}g_t)^2\big] = \mathbf{Var}g_t, \;\; s_{t,x} \approx 0, \;\; s_{t,y} \approx 1 \tag{4.5}$$

An example of the analysis above is summarized in Fig. 4.2. From Eq. 4.4 and Eq. 4.5, note that in Adam, $v_x = v_y$; this is because the update of v_t only uses the amplitude of g_t and ignores its sign, hence the stepsize for the x and y direction is the same $1/\sqrt{v_{t,x}} = 1/\sqrt{v_{t,y}}$. AdaBelief considers both the magnitude and sign of g_t, and $1/\sqrt{s_{t,x}} \gg 1/\sqrt{s_{t,y}}$, hence takes a large step in the x direction and a small step in the y direction, which matches the behaviour of an ideal optimizer.

Update direction in Adam is close to "sign descent" in low-variance case In this section, we demonstrate that when the gradient has low variance, the update direction in Adam is close to "sign descent", hence deviates from the gradient. This is also mentioned in [5].

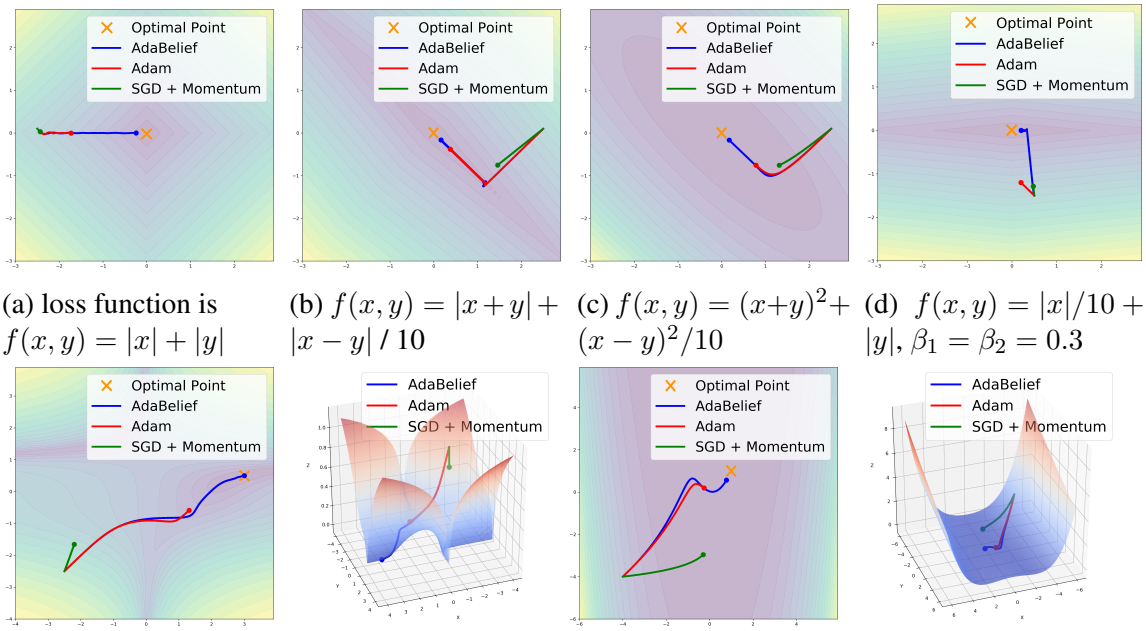
Under the following assumptions: (1) assume g_t is drawn from a stationary distribution, hence after bias correction, $\mathbb{E}v_t = (\mathbb{E}g_t)^2 + \mathbf{Var}g_t$. (2) low-noise assumption, assume $(\mathbb{E}g_t)^2 \gg \mathbf{Var}g_t$, hence we have $\mathbb{E}g_t/\sqrt{\mathbb{E}v_t} \approx \mathbb{E}g_t/\sqrt{(\mathbb{E}g_t)^2} = sign(\mathbb{E}g_t)$. (3) low-bias assumption, assume β_1^t (β_1 to the power of t) is small, hence m_t as an estimator of $\mathbb{E}g_t$ has a small bias $\beta_1^t \mathbb{E}g_t$. Then

$$\Delta\theta_t^{Adam} = -\alpha\frac{m_t}{\sqrt{v_t}+\epsilon} \approx -\alpha\frac{\mathbb{E}g_t}{\sqrt{(\mathbb{E}g_t)^2+\mathbf{Var}g_t}+\epsilon} \approx -\alpha\frac{\mathbb{E}g_t}{||\mathbb{E}g_t||} = -\alpha\ \mathrm{sign}(\mathbb{E}g_t) \tag{4.6}$$

In this case, Adam behaves like a "sign descent"; in 2D cases the update is $\pm 45^\circ$ to the axis, hence deviates from the true gradient direction. The "sign update" effect might cause the generalization gap between adaptive methods and SGD (e.g. on ImageNet) [9, 165]. For AdaBelief, when the variance of g_t is the same for all coordinates, the update direction matches the gradient direction; when the variance is not uniform, AdaBelief takes a small (large) step when the variance is large (small).

Numerical experiments In this section, we validate intuitions in Sec. 4.2.2. Examples are shown in Fig. 6.1, and we refer readers to more *video examples*[1] for better visualization. In all examples, compared with SGD with momentum and Adam, AdaBelief reaches the optimal point at the fastest speed. Learning rate is $\alpha = 10^{-3}$ for all optimizers. For all examples except Fig. 6.1(d), we set the parameters of AdaBelief to be the same as the default in Adam [74], $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$, and set momentum as 0.9 for SGD. For Fig. 6.1(d), to match the assumption in Sec. 4.2.2, we set $\beta_1 = \beta_2 = 0.3$ for both

[1] `https://www.youtube.com/playlist?list=PL7KkG3n9bER6YmMLrKJ5wocjlvP7aWoOu`



(a) loss function is $f(x, y) = |x| + |y|$

(b) $f(x, y) = |x+y| + |x-y|\,/\,10$

(c) $f(x, y) = (x+y)^2 + (x-y)^2/10$

(d) $f(x, y) = |x|/10 + |y|$, $\beta_1 = \beta_2 = 0.3$

(e) Trajectory for Beale function in 2D.

(f) Trajectory for Beale function in 3D.

(g) Rosenbrock function in 2D.

(h) Rosenbrock function in 3D.

Figure 4.3: Trajectories of SGD, Adam and AdaBelief. AdaBelief reaches optimal point (marked as orange cross in 2D plots) the fastest in all cases. We refer readers to *video examples*.

Adam and AdaBelief, and set momentum as 0.3 for SGD.

(a) Consider the loss function $f(x, y) = |x| + |y|$ and a starting point near the x axis. This setting corresponds to Fig. 4.2. Under the same setting, AdaBelief takes a large step in the x direction, and a small step in the y direction, validating our analysis. More examples such as $f(x, y) = |x|/10 + |y|$ are in the supplementary videos.

(b) For an inseparable L_1 loss, AdaBelief outperforms other methods under the same setting.

(c) For an inseparable L_2 loss, AdaBelief outperforms other methods under the same setting.

(d) We set $\beta_1 = \beta_2 = 0.3$ for Adam and AdaBelief, and set momentum as 0.3 in SGD. This corresponds to settings of Eq. 4.6. For the loss $f(x, y) = |x|/10 + |y|$, g_t is a constant for a large region, hence $||\mathbb{E}g_t|| \gg \mathbf{Var}g_t$. As mentioned in [74],

$\mathbb{E}m_t = (1-\beta^t)\mathbb{E}g_t$, hence a smaller β decreases $||m_t - \mathbb{E}g_t||$ faster to 0. Adam behaves like a sign descent (45° to the axis), while AdaBelief and SGD update in the direction of the gradient.

(e)-(f) Optimization trajectory under default setting for the Beale [7] function in 2D and 3D.

(g)-(h) Optimization trajectory under default setting for the Rosenbrock [130] function.

Above cases occur frequently in deep learning Although the above cases are simple, they give hints to local behavior of optimizers in deep learning, and we expect them to occur frequently in deep learning. Hence, we expect AdaBelief to outperform Adam in *general cases*. Other works in the literature [127, 92] claim advantages over Adam, but are typically substantiated with *carefully-constructed examples*. Note that most deep networks use ReLU activation [45], which behaves like an absolute value function as in Fig. 6.1(a). Considering the interaction between neurons, most networks behave like case Fig. 6.1(b), and typically are ill-conditioned (the weight of some parameters are far larger than others) as in the figure. Considering a smooth loss function such as cross entropy or a smooth activation, this case is similar to Fig. 6.1(c). The case with Fig. 6.1(d) requires $|m_t| \approx |\mathbb{E}g_t| \gg \mathbf{Var}g_t$, and this typically occurs at the late stages of training, where the learning rate α is decayed to a small value, and the network reaches a stable region.

4.2.3 Convergence rate of AdaBelief in convex and non-convex optimization

Similar to [127, 92, 24], for simplicity, we omit the de-biasing step (analysis applicable to de-biased version). Proof for convergence in convex and non-convex cases is in the appendix.

Optimization problem For deterministic problems, the problem to be optimized is

$\min_{\theta \in \mathcal{F}} f(\theta)$; for online optimization, the problem is $\min_{\theta \in \mathcal{F}} \sum_{t=1}^{T} f_t(\theta)$, where f_t can be interpreted as loss of the model with the chosen parameters in the t-th step.

Theorem 4.2.1. *(Convergence in convex optimization) Let $\{\theta_t\}$ and $\{s_t\}$ be the sequence obtained by AdaBelief, let $0 \leq \beta_2 < 1, \alpha_t = \frac{\alpha}{\sqrt{t}}$, $\beta_{11} = \beta_1$, $0 \leq \beta_{1t} \leq \beta_1 < 1, s_t \leq s_{t+1}, \forall t \in [T]$. Let $\theta \in \mathcal{F}$, where $\mathcal{F} \subset \mathbb{R}^d$ is a convex feasible set with bounded diameter D_∞. Assume $f(\theta)$ is a convex function and $||g_t||_\infty \leq G_\infty/2$ (hence $||g_t - m_t||_\infty \leq G_\infty$) and $s_{t,i} \geq c > 0, \forall t \in [T], \theta \in \mathcal{F}$. Denote the optimal point as θ^*. For θ_t generated with AdaBelief, we have the following bound on the regret:*

$$\sum_{t=1}^{T}[f_t(\theta_t) - f_t(\theta^*)] \leq \frac{D_\infty^2 \sqrt{T}}{2\alpha(1-\beta_1)} \sum_{i=1}^{d} s_{T,i}^{1/2} + \frac{(1+\beta_1)\alpha\sqrt{1+\log T}}{2\sqrt{c}(1-\beta_1)^3} \sum_{i=1}^{d} \left|\left|g_{1:T,i}^2\right|\right|_2 + \frac{D_\infty^2}{2(1-\beta_1)} \sum_{t=1}^{T}\sum_{i=1}^{d} \frac{\beta_{1t} s_{t,i}^{1/2}}{\alpha_t}$$

Corollary 4.2.1.1. *Suppose $\beta_{1,t} = \beta_1 \lambda^t, \ \ 0 < \lambda < 1$ in Theorem equation 4.2.1, then we have:*

$$\sum_{t=1}^{T}[f_t(\theta_t) - f_t(\theta^*)] \leq \frac{D_\infty^2 \sqrt{T}}{2\alpha(1-\beta_1)} \sum_{i=1}^{d} s_{T,i}^{1/2} + \frac{(1+\beta_1)\alpha\sqrt{1+\log T}}{2\sqrt{c}(1-\beta_1)^3} \sum_{i=1}^{d} \left|\left|g_{1:T,i}^2\right|\right|_2 + \frac{D_\infty^2 \beta_1 G_\infty}{2(1-\beta_1)(1-\lambda)^2\alpha}$$

For the convex case, Theorem 4.2.1 implies the regret of AdaBelief is upper bounded by $O(\sqrt{T})$. Conditions for Corollary 4.2.1.1 can be relaxed to $\beta_{1,t} = \beta_1/t$ as in [127], which still generates $O(\sqrt{T})$ regret. Similar to Theorem 4.1 in [74] and corollary 1 in [127], where the term $\sum_{i=1}^{d} v_{T,i}^{1/2}$ exists, we have $\sum_{i=1}^{d} s_{T,i}^{1/2}$. Without further assumption, $\sum_{i=1}^{d} s_{T,i}^{1/2} < dG_\infty$ since $||g_t - m_t||_\infty < G_\infty$ as assumed in Theorem 2.1, and dG_∞ is constant. The literature [74, 127, 37] exerts a stronger assumption that $\sum_{i=1}^{d} \sqrt{T} v_{T,i}^{1/2} \ll dG_\infty \sqrt{T}$. Our assumption could be similar or weaker, because $\mathbb{E}s_t = \mathrm{Var} g_t \leq \mathbb{E}g_t^2 = \mathbb{E}v_t$, then we get better regret than $O(\sqrt{T})$.

Theorem 4.2.2. *(Convergence for non-convex stochastic optimization) Under the assumptions:*

- *f is differentiable; $||\nabla f(x) - \nabla f(y)|| \leq L||x-y||, \ \forall x, y$; f is also lower bounded.*

- *The noisy gradient is unbiased, and has independent noise,* $i.e.$ $g_t = \nabla f(\theta_t) + \zeta_t, \mathbb{E}\zeta_t = 0, \zeta_t \perp \zeta_j,\ \forall t, j \in \mathbb{N}, t \neq j$.

- *At step* t*, the algorithm can access a bounded noisy gradient, and the true gradient is also bounded.* $i.e.$ $||\nabla f(\theta_t)|| \leq H,\ ||g_t|| \leq H,\ \forall t > 1$.

Assume $\min_{j\in[d]}(s_1)_j \geq c > 0$*, noise in gradient has bounded variance,* $\mathrm{Var}(g_t) = \sigma_t^2 \leq \sigma^2, s_t \leq s_{t+1}, \forall t \in \mathbb{N}$*, then the proposed algorithm satisfies:*

$$\min_{t\in[T]} \mathbb{E}\left|\left|\nabla f(\theta_t)\right|\right|^2 \leq \frac{H}{\sqrt{T}\alpha}\left[\frac{C_1\alpha^2(H^2+\sigma^2)(1+\log T)}{c} + C_2\frac{d\alpha}{\sqrt{c}} + C_3\frac{d\alpha^2}{c} + C_4\right]$$

as in [24], C_1, C_2, C_3 *are constants independent of* d *and* T*, and* C_4 *is a constant independent of* T.

Corollary 4.2.2.1. *If* $c > C_1 H$ *and assumptions for Theorem 4.2.2 are satisfied, we have:*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left[\alpha_t^2\left|\left|\nabla f(\theta_t)\right|\right|^2\right] \leq \frac{1}{T}\frac{1}{\frac{1}{H}-\frac{C_1}{c}}\left[\frac{C_1\alpha^2\sigma^2}{c}\big(1+\log T\big) + C_2\frac{d\alpha}{\sqrt{c}} + C_3\frac{d\alpha^2}{c} + C_4\right]$$

Theorem 4.2.2 implies the convergence rate for AdaBelief in the non-convex case is $O(\log T/\sqrt{T})$, which is similar to Adam-type optimizers [127, 24]. Note that regret bounds are derived in the *worst possible case*, while empirically AdaBelief outperforms Adam mainly because the cases in Sec. 4.2.2 occur more frequently. It is possible that the above bounds are loose. Also note that we assume $s_t \leq s_{t+1}$, in code this requires to use element wise maximum between s_t and s_{t+1} in the denominator.

Algorithm 12: Adam (Sync-Uncenter)

Initialize x_0, $m_0 \leftarrow 0$, $s_0 \leftarrow 0$,
$t \leftarrow 0$
While x_t not converged
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_x f_t(x_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
$x_t \leftarrow x_{t-1} - \frac{\alpha}{\sqrt{v_t + \epsilon}} m_t$

Algorithm 13: AdaShift (Async-Uncenter)

Initialize x_0, $m_0 \leftarrow 0$, $s_0 \leftarrow 0$,
$t \leftarrow 0$
While x_t not converged
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_x f_t(x_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$x_t \leftarrow x_{t-1} - \frac{\alpha}{\sqrt{v_{t-1} + \epsilon}} g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$

Algorithm 14: AdaBelief (Sync-Center)

Initialize x_0, $m_0 \leftarrow 0$, $s_0 \leftarrow 0$,
$t \leftarrow 0$
While x_t not converged
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_x f_t(x_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$s_t \leftarrow \beta_2 s_{t-1} + (1-\beta_2)(g_t - m_t)^2$
$x_t \leftarrow x_{t-1} - \frac{\alpha}{\sqrt{s_t + \epsilon}} m_t$

Algorithm 15: ACProp (Async-Center)

Initialize x_0, $m_0 \leftarrow 0$, $s_0 \leftarrow 0$,
$t \leftarrow 0$
While x_t not converged
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_x f_t(x_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$x_t \leftarrow x_{t-1} - \frac{\alpha}{\sqrt{s_{t-1} + \epsilon}} g_t$
$s_t \leftarrow \beta_2 s_{t-1} + (1-\beta_2)(g_t - m_t)^2$

4.3 Asynchronous version of AdaBelief

4.3.1 Algorithms

In this section, we summarize the AdaBelief [181] method in Algo. 14 and ACProp in Algo. 15. For the ease of notations, all operations in Algo. 14 and Algo. 15 are element-wise, and we omit the bias-correction step of m_t and s_t for simplicity.

We first introduce the notion of "sync (async)" and "center (uncenter)". *(a) Sync vs Async* The update on parameter x_t can be generally split into a numerator (e.g. m_t, g_t) and a denominator (e.g. $\sqrt{s_t}, \sqrt{v_t}$). We call it "sync" if the denominator depends on g_t, such as in Adam and RMSProp; and call it "async" if the denominator is independent of g_t, for example, denominator uses information up to step $t-1$ for the t-th step. *(b) Center vs Uncenter* The "uncentered" update uses v_t, the exponential moving average (EMA) of g_t^2; while the "centered" update uses s_t, the EMA of $(g_t - m_t)^2$.

Adam (Sync-Uncenter) The Adam optimizer [74] stores the EMA of the gradient in m_t, and stores the EMA of g_t^2 in v_t. For each step of the update, Adam performs element-wise division between m_t and $\sqrt{v_t}$. Therefore, the term $\alpha_t \frac{1}{\sqrt{v_t}}$ can be viewed as the element-wise learning rate. Note that β_1 and β_2 are two scalars controlling the smoothness of the EMA for the first and second moment, respectively. When $\beta_1 = 0$, Adam reduces to RMSProp [58].

AdaBelief (Sync-Center) AdaBelief optimizer [181] is summarized in Algo. 14. Compared with Adam, the key difference is that it replaces the uncentered second moment v_t (EMA of g_t^2) by an estimate of the centered second moment s_t (EMA of $(g_t - m_t)^2$). The intuition is to view m_t as an estimate of the expected gradient: if the observation g_t deviates much from the prediction m_t, then it takes a small step; if the observation g_t is close to the prediction m_t, then it takes a large step.

AdaShift (Async-Uncenter) AdaShift [178] performs temporal decorrelation between numerator and denominator. It uses information of $\{g_{t-n}, ...g_t\}$ for the numerator, and uses $\{g_0, ...g_{t-n-1}\}$ for the denominator, where n is the "delay step" controlling where to split sequence $\{g_i\}_{i=0}^t$. The numerator is independent of denominator because each g_i is only used in either numerator or denominator.

ACProp (Async-Center) Our proposed ACProp is the asynchronous version of AdaBelief and is summarized in Algo. 15. Compared to AdaBelief, the key difference is that ACProp uses s_{t-1} in the denominator for step t, while AdaBelief uses s_t. Note that s_t depends on g_t, while s_{t-1} uses history up to step $t-1$. This modification is important to ensure that $\mathbb{E}(g_t/\sqrt{s_{t-1}}|g_0, ...g_{t-1}) = (\mathbb{E}g_t)/\sqrt{s_{t-1}}$. It's also possible to use a delay step larger than 1 similar to AdaShift, for example, use $EMA(\{g_i\}_{i=t-n}^t)$ as numerator, and $EMA(\{(g_i - m_i)^2\}_{i=0}^{t-n-1})$ for denominator.

4.3.2 Async AdaBelief has a weaker convergence condition

We analyze the convergence conditions for different methods in this section. We first analyze the counter example by Reddi et al. (2018) and show that async-optimizers (AdaShift, ACProp) always converge $\forall \beta_1, \beta_2 \in (0, 1)$, while sync-optimizers (Adam, AdaBelief, RMSProp et al.) would diverge if (β_1, β_2) are not carefully chosen; hence, async-optimizers have weaker convergence conditions than sync-optimizers. Next, we compare async-uncenter (AdaShift) with async-center (ACProp) and show that momentum centering further weakens the convergence condition for sparse-gradient problems. Therefore, ACProp has weaker convergence conditions than AdaShift and other sync-optimizers.

Sync vs Async

We show that for the example in [127], async-optimizers (ACProp, AdaShift) have weaker convergence conditions than sync-optimizers (Adam, RMSProp, AdaBelief).

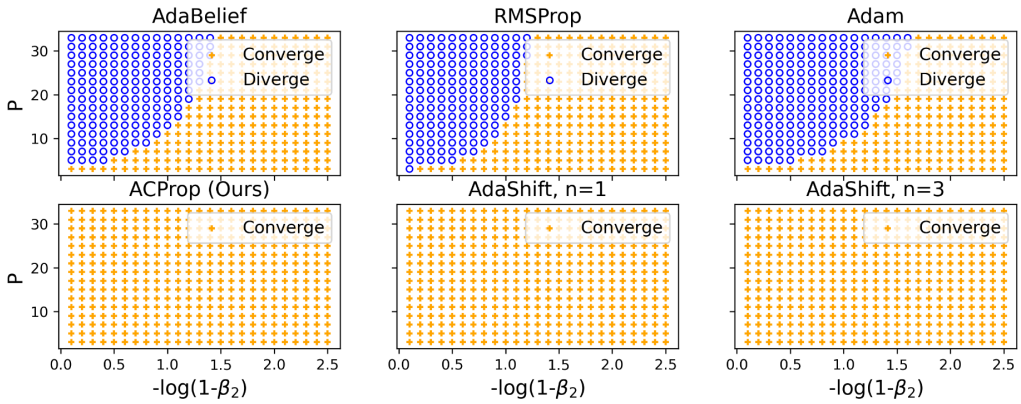


Figure 4.4: Numerical results for the example defined by Eq. equation 4.7. We set the initial value as $x_0 = 0$, and run each optimizer for 10^4 steps trying different initial learning rates in $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1.0\}$, and set the learning rate decays with $1/\sqrt{t}$. If there's a proper initial learning rate, such that the average distance between the parameter and its optimal value $x^* = -1$ for the last 1000 steps is below 0.01, then it's marked as "converge" (orange plus symbol), otherwise as "diverge" (blue circle). For each optimizer, we sweep through different β_2 values in a log grid (x-axis), and sweep through different values of P in the definition of problem (y-axis). We plot the result for $\beta_1 = 0.9$ here; for results with different β_1 values, please refer to appendix. Our results indicate that in the (P, β_2) plane, there's a threshold curve beyond which sync-optimizers (Adam, RMSProp, AdaBelief) will diverge; however, async-optimizers (ACProp, AdaShift) always converge for any point in the (P, β_2) plane. Note that for AdaShift, a larger delay step n is possible to cause divergence (see example in Fig. 4.5 with $n = 10$). To validate that the "divergence" is not due to numerical issues and sync-optimizers are drifting away from optimal, we plot trajectories in Fig. 4.5

Lemma 4.3.0.1 (Thm.1 in [127]). *There exists an online convex optimization problem where sync-optimizers (e.g. Adam, RMSProp) have non-zero average regret, and one example is*

$$f_t(x) = \begin{cases} Px, & \text{if } t\%P = 1 \\ -x, & \text{Otherwise} \end{cases} \quad x \in [-1, 1], P \in \mathbb{N}, P \geq 3 \tag{4.7}$$

Lemma 4.3.0.2 ([143]). *For problem (1) with any fixed P, there's a threshold of β_2 above which RMSProp converges.*

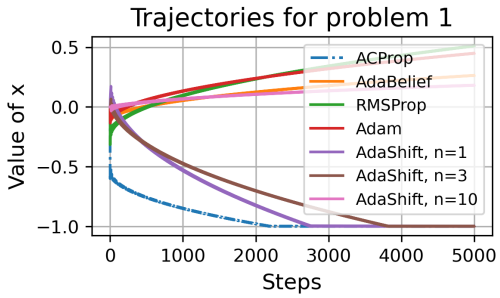


Figure 4.5: Trajectories of x for different optimizers in Problem by Eq. 4.7. Initial point is $x_0 = 0$, the optimal is $x^* = -1$, the trajectories show that sync-optimizers (Adam, AdaBelief, RMSProp) diverge from the optimal, validating the divergent area in Fig. 4.4 is correct rather than artifacts of numerical issues. Async-optimizers (ACProp, AdaShift) converge to optimal value, but large delay step n in AdaShift could cause non-convergence.

In order to better explain the two lemmas above, we conduct numerical experiments on the problem by Eq. equation 4.7, and show results in Fig. 4.4. Note that $\sum_{t=k}^{k+P} f_t(x) = x$, hence the optimal point is $x^* = -1$ since $x \in [-1, 1]$. Starting from initial value $x_0 = 0$, we sweep through the plane of (P, β_2) and plot results of convergence in Fig. 4.4, and plot example trajectories in Fig. 4.5.

Lemma. 4.3.0.1 tells half of the story: looking at each vertical line in the subfigure of Fig. 4.4, that is, for each fixed hyper-parameter β_2, there exists sufficiently large P such that Adam (and RMSProp) would diverge. Lemma. 4.3.0.2 tells the other half of the story: looking at each horizontal line in the subfigure of Fig. 4.4, for each problem with a fixed period P, there exists sufficiently large β_2s beyond which Adam can converge.

The complete story is to look at the (P, β_2) plane in Fig. 4.4. There is a boundary between convergence and divergence area for sync-optimizers (Adam, RMSProp, Ad-

aBelief), while async-optimizers (ACProp, AdaShift) always converge.

Lemma 4.3.0.3. *For the problem defined by Eq. equation 4.7, using learning rate schedule of $\alpha_t = \frac{\alpha_0}{\sqrt{t}}$, async-optimizers (ACProp and AdaShift with $n = 1$) always converge* $\forall \beta_1, \beta_2 \in (0,1), \forall P \in \mathbb{N}, P \geq 3$.

The proof is in the appendix. Note that for AdaShift, proof for the always-convergence property only holds when $n = 1$; larger n could cause divergence (e.g. $n = 10$ causes divergence as in Fig. 4.5). The always-convergence property of ACProp and AdaShift comes from the un-biased stepsize, while the stepsize for sync-optimizers are biased due to correlation between numerator and denominator. Taking RMSProp as example of sync-optimizer, the update is $-\alpha_t \frac{g_t}{\sqrt{v_t}} = -\alpha_t \frac{g_t}{\sqrt{\beta_2^t g_0^2 + \ldots + \beta_2 g_{t-1}^2 + g_t^2}}$. Note that g_t is used both in the numerator and denominator, hence a large g_t does not necessarily generate a large stepsize. For the example in Eq. equation 4.7, the optimizer observes a gradient of -1 for $P-1$ times and a gradient of P once; due to the biased stepsize in sync-optimizers, the gradient of P does not generate a sufficiently large stepsize to compensate for the effect of wrong gradients -1, hence cause non-convergence. For async-optimizers, g_t is not used in the denominator, therefore, the stepsize is not biased and async-optimizers has the always-convergence property.

Remark Reddi et al. (2018) proposed AMSGrad to track the element-wise maximum of v_t in order to achieve the always-convergence property. However, tracking the maximum in the denominator will in general generate a small stepsize, which often harms empirical performance. We demonstrate this through experiments in later sections in Fig. 4.9.

Async-Uncenter vs Async-Center

In the last section, we demonstrated that async-optimizers have weaker convergence conditions than sync-optimizers. In this section, within the async-optimizer family, we analyze

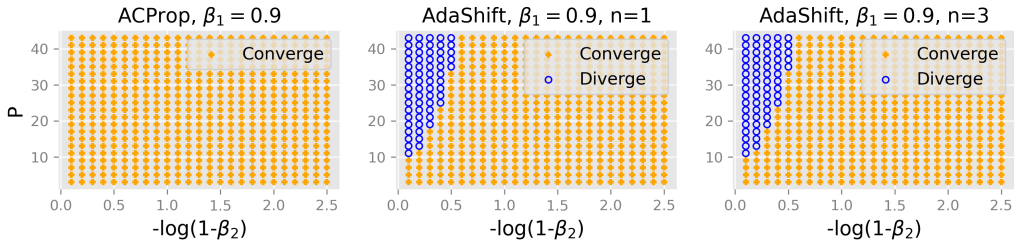


Figure 4.6: Area of convergence for the problem in Eq. equation 4.8. The numerical experiment is performed under the same setting as in Fig. 4.4.Our results experimentally validated the claim that compared with async-uncenter (AdaShift), async-center (ACProp) has a larger convergence area in the hyper-parameter space.

the effect of centering second momentum. We show that compared with async-uncenter (AdaShift), async-center (ACProp) has weaker convergence conditions. We consider the following online convex problem:

$$f_t(x) = \begin{cases} P/2 \times x, & t\%P == 1 \\ -x, & t\%P == P-2 \\ 0, & \textit{otherwise} \end{cases} \quad P > 3, P \in \mathbb{N}, x \in [0,1]. \tag{4.8}$$

Initial point is $x_0 = 0.5$. Optimal point is $x^* = 0$. We have the following results:

Lemma 4.3.0.4. *For the problem defined by Eq. equation 4.8, consider the hyper-parameter tuple (β_1, β_2, P), there exists cases where ACProp converges but AdaShift with $n = 1$ diverges, but not vice versa.*

We provide the proof in the appendix. Lemma. 4.3.0.4 implies that ACProp has a larger area of convergence than AdaShift, hence the centering of second momentum further weakens the convergence conditions. We first validate this claim with numerical experiments in Fig. 4.6; for sanity check, we plot the trajectories of different optimizers in Fig. 4.7. We observe that the convergence of AdaShift is influenced by delay step n, and there's no good criterion to select a good value of n, since Fig. 4.5 requires a small n for convergence in problem equation 4.7, while Fig. 4.7 requires a large n for convergence in

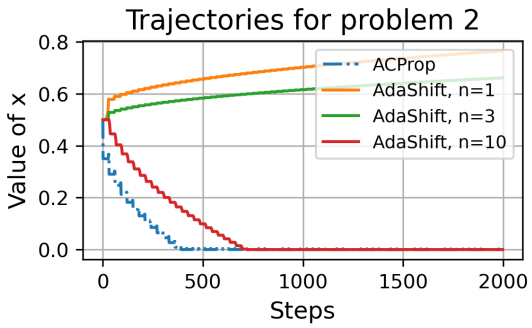


Figure 4.7: Trajectories for problem defined by Eq. equation 4.8. Note that the optimal point is $x^* = 0$.

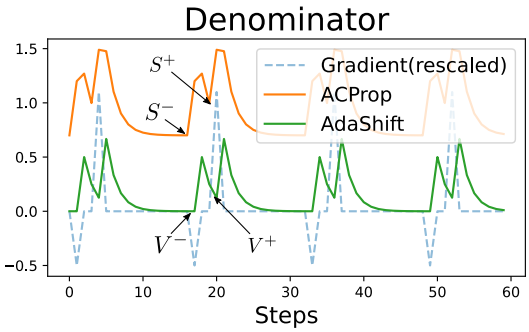


Figure 4.8: Value of uncentered second momentum v_t and centered momentum s_t for problem equation 4.8.

problem equation 4.8. ACProp has a larger area of convergence, indicating that both async update and second momentum centering helps weaken the convergence conditions.

We provide an intuitive explanation on why momentum centering helps convergence. Due to the periodicity of the problem, the optimizer behaves almost periodically as $t \to \infty$. Within each period, the optimizer observes one positive gradient $P/2$ and one negative gradient -1. As in Fig. 4.8, between observing non-zero gradients, the gradient is always 0. Within each period, ACprop will perform a positive update $P/(2\sqrt{s^+})$ and a negative update $-1/\sqrt{s^-}$, where s^+ (s^-) is the value of denominator before observing positive (negative) gradient. Similar notations for v^+ and v^- in AdaShift. A net update in the correct direction requires $\frac{P}{2\sqrt{s^+}} > \frac{1}{\sqrt{s^-}}$, (or $s^+/s^- < P^2/4$).

When observing 0 gradient, for AdaShift, $v_t = \beta_2 v_{t-1} + (1-\beta_2)0^2$; for ACProp, $s_t = \beta_2 s_{t-1} + (1-\beta_2)(0-m_t)^2$ where $m_t \neq 0$. Therefore, v^- decays exponentially to 0, but s^- decays to a non-zero constant, hence $\frac{s^+}{s^-} < \frac{v^+}{v^-}$, hence ACProp is easier to satisfy $s^+/s^- < P^2/4$ and converge.

4.3.3 Async AdaBelief matches the oracle convergence rate

In this section, we show that ACProp converges at a rate of $O(1/\sqrt{T})$ in the stochastic non-convex case, which matches the oracle [3] for first-order optimizers and outperforms the $O(logT/\sqrt{T})$ rate for sync-optimizers (Adam, RMSProp and AdaBelief) [24, 143, 181]. We further show that the upper bound on regret of async-center (ACProp) outperforms async-uncenter (AdaShift) by a constant.

For the ease of analysis, we denote the update as: $x_t = x_{t-1} - \alpha_t A_t g_t$, where A_t is the diagonal preconditioner. For SGD, $A_t = I$; for sync-optimizers (RMSProp), $A_t = \frac{1}{\sqrt{v_t}+\epsilon}$; for AdaShift with $n = 1$, $A_t = \frac{1}{\sqrt{v_{t-1}}+\epsilon}$; for ACProp, $A_t = \frac{1}{\sqrt{s_{t-1}}+\epsilon}$. For async optimizers, $\mathbb{E}[A_t g_t | g_0, ...g_{t-1}] = A_t \mathbb{E} g_t$; for sync-optimizers, this does not hold because g_t is used in A_t

Theorem 4.3.1 (convergence for stochastic non-convex case)**.** *Under the following assumptions:*

- *f is continuously differentiable, f is lower-bounded by f^* and upper bounded by M_f. $\nabla f(x)$ is globally Lipschitz continuous with constant L:*

$$||\nabla f(x) - \nabla f(y)|| \leq L||x - y|| \tag{4.9}$$

- *For any iteration t, g_t is an unbiased estimator of $\nabla f(x_t)$ with variance bounded by σ^2. Assume norm of g_t is bounded by M_g.*

$$\mathbb{E}\big[g_t\big] = \nabla f(x_t) \quad \mathbb{E}\big[||g_t - \nabla f(x_t)||^2\big] \leq \sigma^2 \tag{4.10}$$

then for $\beta_1, \beta_2 \in [0, 1)$, with learning rate schedule as: $\alpha_t = \alpha_0 t^{-\eta}$, $\alpha_0 \leq \frac{C_l}{LC_u^2}$, $\eta \in [0.5, 1)$

for the sequence $\{x_t\}$ generated by ACProp, we have

$$\frac{1}{T}\sum_{t=1}^{T}\Big|\Big|\nabla f(x_t)\Big|\Big|^2 \leq \frac{2}{C_l}\Big[(M_f - f^*)\alpha_0 T^{\eta-1} + \frac{LC_u^2\sigma^2\alpha_0}{2(1-\eta)}T^{-\eta}\Big] \tag{4.11}$$

where C_l and C_u are scalars representing the lower and upper bound for A_t, e.g. $C_l I \preceq A_t \preceq C_u I$, where $A \preceq B$ represents $B - A$ is semi-positive-definite.

Note that there's a natural bound for C_l and C_u: $C_u \leq \frac{1}{\epsilon}$ and $C_l \geq \frac{1}{2M_g}$ because ϵ is added to denominator to avoid division by 0, and g_t is bounded by M_g. Thm. 4.3.1 implies that ACProp has a convergence rate of $O(1/\sqrt{T})$ when $\eta = 0.5$; equivalently, in order to have $||\nabla f(x)||^2 \leq \delta^2$, ACProp requires at most $O(\delta^{-4})$ steps.

Theorem 4.3.2 (Oracle complexity [3])**.** *For a stochastic non-convex problem satisfying assumptions in Theorem. 4.3.1, using only up to first-order gradient information, in the worst case any algorithm requires at least $O(\delta^{-4})$ queries to find a δ-stationary point x such that $||\nabla f(x)||^2 \leq \delta^2$.*

Optimal rate in big O Thm. 4.3.1 and Thm. 4.3.2 imply that async-optimizers achieves a convergence rate of $O(1/\sqrt{T})$ for the stochastic non-convex problem, which matches the oracle complexity and outperforms the $O(logT/\sqrt{T})$ rate of sync-optimizers (Adam [127], RMSProp[143], AdaBelief [181]). Adam and RMSProp are shown to achieve $O(1/\sqrt{T})$ rate under the stricter condition that $\beta_{2,t} \to 1$ [182]. A similar rate has been achieved in AVAGrad [141], and AdaGrad is shown to achieve a similar rate [81]. Despite the same convergence rate, we show that ACProp has better empirical performance.

Constants in the upper bound of regret Though both async-center and async-uncenter optimizers have the same convergence rate with matching upper and lower bound in big O notion, the constants of the upper bound on regret is different. Thm. 4.3.1 implies that the upper bound on regret is an increasing function of $1/C_l$ and C_u, and

$$1/C_l = \sqrt{K_u} + \epsilon, \;\; C_u = 1/(\sqrt{K_l} + \epsilon)$$

where K_l and K_u are the lower and upper bound of second momentum, respectively.

We analyze the constants in regret by analyzing K_l and K_u. If we assume the observed gradient g_t follows some independent stationary distribution, with mean μ and variance σ^2, then approximately

$$\textit{Uncentered second momentum: } 1/C_l^v = \sqrt{K_u^v} + \epsilon \approx \sqrt{\mu^2 + \sigma^2} + \epsilon \tag{4.12}$$

$$\textit{Centered second momentum: } 1/C_l^s = \sqrt{K_u^s} + \epsilon \approx \sqrt{\sigma^2} + \epsilon \tag{4.13}$$

During early phase of training, in general $|\mu| \gg \sigma$, hence $1/C_l^s \ll 1/C_l^v$, and the centered version (ACProp) can converge faster than uncentered type (AdaShift) by a constant factor of around $\frac{\sqrt{\mu^2+\sigma^2}+\epsilon}{\sqrt{\sigma^2}+\epsilon}$. During the late phase, g_t is centered around 0, and $|\mu| \ll \sigma$, hence K_l^v (for uncentered version) and K_l^s (for centered version) are both close to 0, hence C_u term is close for both types.

Remark We emphasize that ACProp rarely encounters numerical issues caused by a small s_t as denominator, even though Eq. equation 4.13 implies a lower bound for s_t around σ^2 which could be small in extreme cases. Note that s_t is an estimate of mixture of two aspects: the change in true gradient $||\nabla f_t(x) - \nabla f_{t-1}(x)||^2$, and the noise in g_t as an observation of $\nabla f(x)$. Therefore, two conditions are essential to achieve $s_t = 0$: the true gradient $\nabla f_t(x)$ remains constant, and g_t is a noise-free observation of $\nabla f_t(x)$. Eq. equation 4.13 is based on assumption that $||\nabla f_t(x) - \nabla f_{t-1}(x)||^2 = 0$, if we further assume $\sigma = 0$, then the problem reduces to a trivial ideal case: a linear loss surface with clean observations of gradient, which is rarely satisfied in practice. More discussions are in appendix.

Empirical validations We conducted experiments on the MNIST dataset using a 2-layer MLP. We plot the average value of v_t for uncentered-type and s_t for centered-type

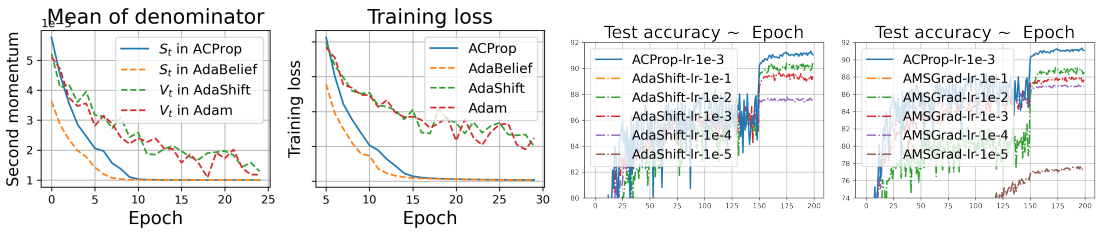


Figure 4.9: From left to right: (a) Mean value of denominator for a 2-layer MLP on MNIST dataset. (b) Training loss of different optimizers for the 2-layer MLP model. (c) Performance of AdaShift for VGG-11 on CIFAR10 varying with learning rate ranging from 1e-1 to 1e-5, we plot the performance of ACProp with learning rate 1e-3 as reference. Missing lines are because their accuracy are below display threshold. All methods decay learning rate by a factor of 10 at 150th epoch. (d) Performance of AMSGrad for VGG-11 on CIFAR10 varying with learning rate under the same setting in (c).

optimizers; as Fig. 4.9(a,b) shows, we observe $s_t \leq v_t$ and the centered-type (ACProp, AdaBelief) converges faster, validating our analysis for early phases. For epochs > 10, we observe that $\min s_t \approx \min v_t$, validating our analysis for late phases.

As in Fig. 4.9(a,b), the ratio v_t/s_t decays with training, and in fact it depends on model structure and dataset noise. Therefore, empirically it's hard to compensate for the constants in regret by applying a larger learning rate for async-uncenter optimizers. As shown in Fig. 4.9(c,d), for VGG network on CIFAR10 classification task, we tried different initial learning rates for AdaShift (async-uncenter) and AMSGrad ranging from 1e-1 to 1e-5, and their performances are all inferior to ACProp with a learning rate 1e-3. Please see Fig.4.11 for a complete table varying with hyper-parameters.

4.4 Experiments

We validate the performance of ACProp in various experiments, including image classification with convolutional neural networks (CNN), reinforcement learning with deep Q-network (DQN), machine translation with transformer and generative adversarial networks (GANs). We aim to test both the generalization performance and training stability: SGD family optimizers typically are the default for CNN models such as in image recog-

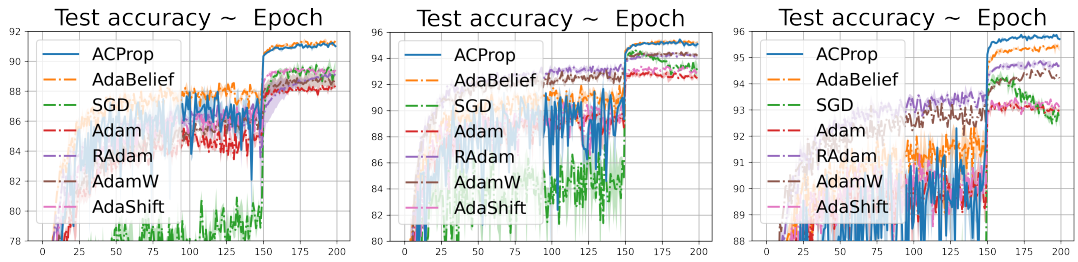


Figure 4.10: Test accuracy ($mean \pm std$) on CIFAR10 datset. Left to right: VGG-11, ResNet-34, DenseNet-121.

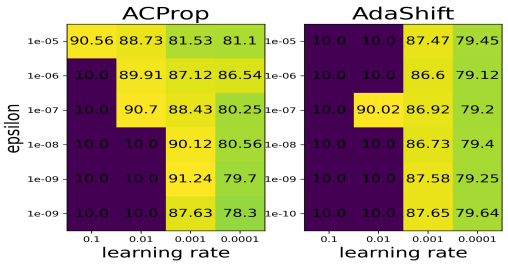


Figure 4.11: Test accuracy (%) of VGG network on CIFAR10 under different hyper-parameters. We tested learning rate in $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$ and $\epsilon \in \{10^{-5}, ..., 10^{-9}\}$.

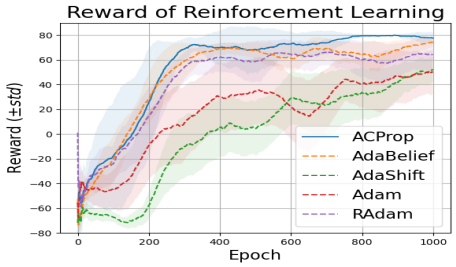


Figure 4.12: The reward (higher is better) curve of a DQN-network on the four-rooms problem. We report the mean and standard deviation across 10 independent runs.

Table 4.2: Top-1 accuracy of ResNet18 on ImageNet. $^\diamond$ is reported in PyTorch Documentation, † is reported in [20], $*$ is reported in [88].

SGD	Padam	Adam	AdamW	RAdam	AdaShift	AdaBelief	ACProp
69.76$^\diamond$ (70.23†)	70.07†	66.54*	67.93†	67.62*	65.28	70.08	**70.46**

nition [53] and object detection [128] due to their better generalization performance than Adam; and Adam is typically the default for GANs [46], reinforcement learning [101] and transformers [158], mainly due to its better numerical stability and faster convergence than SGD. We aim to validate that ACProp can perform well for both cases.

Image classification with CNN We first conducted experiments on CIFAR10 image classification task with a VGG-11 [145], ResNet34 [53] and DenseNet-121 [61]. We performed extensive hyper-parameter tuning in order to better compare the performance of different optimizers: for SGD we set the momentum as 0.9 which is the default for many cases [53, 61], and search the learning rate between 0.1 and 10^{-5} in the log-grid; for

Table 4.3: BLEU score (higher is better) on machine translation with Transformer

	Adam	RAdam	AdaShift	AdaBelief	ACProp
DE-EN	34.66±0.014	34.76±0.003	30.18±0.020	35.17±0.015	**35.35±0.012**
EN-VI	21.83±0.015	22.54±0.005	20.18±0.231	22.45±0.003	**22.62±0.008**
JA-EN	33.33±0.008	32.23±0.015	25.24±0.151	**34.38±0.009**	33.70±0.021
RO-EN	29.78± 0.003	30.26 ± 0.011	27.86±0.024	30.03±0.012	**30.27±0.007**

Table 4.4: FID (lower is better) for GANs

	Adam	RAdam	AdaShift	AdaBelief	ACProp
DCGAN	49.29±0.25	48.24±1.38	99.32±3.82	47.25±0.79	**43.43±4.38**
RLGAN	38.18±0.01	40.61±0.01	56.18 ±0.23	**36.58±0.12**	37.15±0.13
SNGAN	13.14±0.10	13.00±0.04	26.62±0.21	12.70±0.17	**12.44±0.02**
SAGAN	13.98±0.02	14.25±0.01	22.11±0.25	14.17±0.14	**13.54±0.15**

other adaptive optimizers, including AdaBelief, Adam, RAdam, AdamW and AdaShift, we search the learning rate between 0.01 and 10^{-5} in the log-grid, and search ϵ between 10^{-5} and 10^{-10} in the log-grid. We use a weight decay of 5e-2 for AdamW, and use 5e-4 for other optimizers. We report the $mean \pm std$ for the best of each optimizer in Fig. 4.10: for VGG and ResNet, ACProp achieves comparable results with AdaBelief and outperforms other optimizers; for DenseNet, ACProp achieves the highest accuracy and even outperforms AdaBelief by 0.5%. As in Table 4.2, for ResNet18 on ImageNet, ACProp outperforms other methods and achieves comparable accuracy to the best of SGD in the literature, validating its generalization performance.

To evaluate the robustness to hyper-parameters, we test the performance of various optimizers under different hyper-parameters with VGG network. We plot the results for ACProp and AdaShift as an example in Fig. 4.11 and find that ACProp is more robust to hyper-parameters and typically achieves higher accuracy than AdaShift.

Table 4.5: Performance comparison between AVAGrad and ACProp. ↑ (↓) represents metrics that upper (lower) is better. * are reported in the AVAGrad paper [141]

	WideResNet Test Error (↓)		Transformer BLEU (↑)		GAN FID (↓)	
	CIFAR10	CIFAR100	DE-EN	RO-EN	DCGAN	SNGAN
AVAGrad	3.80*±0.02	18.76*±0.20	30.23±0.024	27.73±0.134	59.32±3.28	21.02±0.14
ACProp	**3.67±0.04**	**18.72±0.01**	**35.35±0.012**	**30.27±0.007**	**43.34±4.38**	**12.44±0.02**

Reinforcement learning with DQN We evaluated different optimizers on reinforcement learning with a deep Q-network (DQN) [101] on the four-rooms task [149]. We tune the hyper-parameters in the same setting as previous section. We report the mean and standard deviation of reward (higher is better) across 10 runs in Fig. 4.12. ACProp achieves the highest mean reward, validating its numerical stability and good generalization.

Neural machine translation with Transformer We evaluated the performance of ACProp on neural machine translation tasks with a transformer model [158]. For all optimizers, we set learning rate as 0.0002, and search for $\beta_1 \in \{0.9, 0.99, 0.999\}$, $\beta_2 \in \{0.98, 0.99, 0.999\}$ and $\epsilon \in \{10^{-5}, 10^{-6}, ...10^{-16}\}$. As shown in Table. 4.3, ACProp achieves the highest BLEU score in 3 out 4 tasks, and consistently outperforms a well-tuned Adam.

Generative Adversarial Networks (GAN) The training of GANs easily suffers from mode collapse and numerical instability [139], hence is a good test for the stability of optimizers. We conducted experiments with Deep Convolutional GAN (DCGAN) [125], Spectral-Norm GAN (SNGAN) [100], Self-Attention GAN (SAGAN) [172] and Relativistic-GAN (RLGAN) [67]. We set $\beta_1 = 0.5$, and search for β_2 and ϵ with the same schedule as previous section. We report the FID [56] on CIFAR10 dataset in Table. 4.4, where a lower FID represents better quality of generated images. ACProp achieves the best overall FID score and outperforms well-tuned Adam.

Remark Besides AdaShift, we found another async-optimizer named AVAGrad in [141]. Unlike other adaptive optimizers, AVAGrad is not scale-invariant hence the default hyper-parameters are very different from Adam-type ($lr = 0.1, \epsilon = 0.1$). We searched for hyper-parameters for AVAGrad for a much larger range, with ϵ between 1e-8 and 100 in the log-grid, and lr between 1e-6 and 100 in the log-grid. For experiments with a WideResNet, we replace the optimizer in the official implementation for AVAGrad by ACProp, and cite results in the AVAGrad paper. As in Table 4.5, ACProp consistently outperforms

AVAGrad in CNN, Transformer, and GAN training.

4.5 Proofs and theoretical analysis

4.5.1 Convergence of AdaBelief in convex online learning case

For the ease of notation, we absorb ϵ into s_t. Equivalently, $s_t \geq c > 0, \forall t \in [T]$. For simplicity, we omit the debiasing step in theoretical analysis as in [127]. Our analysis can be applied to the de-biased version as well.

Lemma 4.5.0.1. *[98] For any $Q \in S_+^d$ and convex feasible set $\mathcal{F} \subset \mathbb{R}^d$, suppose $u_1 = \min_{x \in \mathcal{F}} \left|\left|Q^{1/2}(x - z_1)\right|\right|$ and $u_2 = \min_{x \in \mathcal{F}} \left|\left|Q^{1/2}(x - z_2)\right|\right|$, then we have $\left|\left|Q^{1/2}(u_1 - u_2)\right|\right| \leq \left|\left|Q^{1/2}(z_1 - z_2)\right|\right|$.*

Theorem 4.5.1. *Let $\{\theta_t\}$ and $\{s_t\}$ be the sequence obtained by the proposed algorithm, let $0 \leq \beta_2 < 1, \alpha_t = \frac{\alpha}{\sqrt{t}}$, $\beta_{11} = \beta_1$, $0 \leq \beta_{1t} \leq \beta_1 < 1, s_{t-1} \leq s_t, \forall t \in [T]$. Let $\theta \in \mathcal{F}$, where $\mathcal{F} \subset \mathbb{R}^d$ is a convex feasible set with bounded diameter D_∞. Assume $f(\theta)$ is a convex function and $||g_t||_\infty \leq G_\infty/2$ (hence $||g_t - m_t||_\infty \leq G_\infty$) and $s_{t,i} \geq c > 0, \forall t \in [T], \theta \in \mathcal{F}$. Denote the optimal point as θ^*. For θ_t generated with Algorithm* **??***, we have the following bound on the regret:*

$$\sum_{t=1}^{T} f_t(\theta_t) - f_t(\theta^*) \leq \frac{D_\infty^2 \sqrt{T}}{2\alpha(1-\beta_1)} \sum_{i=1}^{d} s_{T,i}^{1/2} + \frac{(1+\beta_1)\alpha\sqrt{1+\log T}}{2\sqrt{c}(1-\beta_1)^3} \sum_{i=1}^{d} \left|\left| g_{1:T,i}^2 \right|\right|_2 + \frac{D_\infty^2}{2(1-\beta_1)} \sum_{t=1}^{T} \sum_{i=1}^{d} \frac{\beta_{1t} s_{t,i}^{1/2}}{\alpha_t}$$

Proof:

$$\theta_{t+1} = \prod_{\mathcal{F},\sqrt{s_t}} (\theta_t - \alpha_t s_t^{-1/2} m_t) = \min_{\theta \in \mathcal{F}} \left|\left| s_t^{1/4} [\theta - (\theta_t - \alpha_t s_t^{-1/2} m_t)] \right|\right|$$

Note that $\prod_{\mathcal{F},\sqrt{s_t}}(\theta^*) = \theta^*$ since $\theta^* \in \mathcal{F}$. Use θ_i^* and $\theta_{t,i}$ to denote the ith dimension of θ^* and θ_t respectively. From lemma equation 4.5.0.1, using $u_1 = \theta_{t+1}$ and $u_2 = \theta^*$, we have:

$$\begin{aligned}
\left|\left| s_t^{1/4}(\theta_{t+1} - \theta^*) \right|\right|^2 &\leq \left|\left| s_t^{1/4}(\theta_t - \alpha_t s_t^{-1/2} m_t - \theta^*) \right|\right|^2 \\
&= \left|\left| s_t^{1/4}(\theta_t - \theta^*) \right|\right|^2 + \alpha_t^2 \left|\left| s_t^{-1/4} m_t \right|\right|^2 - 2\alpha_t \langle m_t, \theta_t - \theta^* \rangle \\
&= \left|\left| s_t^{1/4}(\theta_t - \theta^*) \right|\right|^2 + \alpha_t^2 \left|\left| s_t^{-1/4} m_t \right|\right|^2 \\
&\quad - 2\alpha_t \langle \beta_{1t} m_{t-1} + (1 - \beta_{1t}) g_t, \theta_t - \theta^* \rangle \qquad (4.14)
\end{aligned}$$

Note that $\beta_1 \in [0, 1)$ and $\beta_2 \in [0, 1)$, rearranging inequality equation 4.14, we have:

$$\begin{aligned}
\langle g_t, \theta_t - \theta^* \rangle &\leq \frac{1}{2\alpha_t(1 - \beta_{1t})} \left[\left|\left| s_t^{1/4}(\theta_t - \theta^*) \right|\right|^2 - \left|\left| s_t^{1/4}(\theta_{t+1} - \theta^*) \right|\right|^2 \right] \\
&\quad + \frac{\alpha_t}{2(1 - \beta_{1t})} \left|\left| s_t^{-1/4} m_t \right|\right|^2 - \frac{\beta_{1t}}{1 - \beta_{1t}} \langle m_{t-1}, \theta_t - \theta^* \rangle \\
&\leq \frac{1}{2\alpha_t(1 - \beta_{1t})} \left[\left|\left| s_t^{1/4}(\theta_t - \theta^*) \right|\right|^2 - \left|\left| s_t^{1/4}(\theta_{t+1} - \theta^*) \right|\right|^2 \right] \\
&\quad + \frac{\alpha_t}{2(1 - \beta_{1t})} \left|\left| s_t^{-1/4} m_t \right|\right|^2 \\
&\quad + \frac{\beta_{1t}}{2(1 - \beta_{1t})} \alpha_t \left|\left| s_t^{-1/4} m_{t-1} \right|\right|^2 + \frac{\beta_{1t}}{2\alpha_t(1 - \beta_{1t})} \left|\left| s_t^{1/4}(\theta_t - \theta^*) \right|\right|^2 \\
&\quad \left(\textit{Cauchy-Schwartz and Young's inequality: } ab \leq \frac{a^2 \epsilon}{2} + \frac{b^2}{2\epsilon}, \forall \epsilon > 0 \right) \qquad (4.15)
\end{aligned}$$

By convexity of f, we have:

$$
\begin{aligned}
\sum_{t=1}^{T} f_t(\theta_t) - f_t(\theta^*) &\leq \sum_{t=1}^{T} \langle g_t, \theta_t - \theta^* \rangle \\
&\leq \sum_{t=1}^{T} \Big\{ \frac{1}{2\alpha_t(1-\beta_{1t})} \Big[\Big|\Big| s_t^{1/4}(\theta_t - \theta^*) \Big|\Big|^2 - \Big|\Big| s_t^{1/4}(\theta_{t+1} - \theta^*) \Big|\Big|^2 \Big] \\
&+ \frac{1}{2(1-\beta_{1t})} \alpha_t \Big|\Big| s_t^{-1/4} m_t \Big|\Big|^2 + \frac{\beta_{1t}}{2(1-\beta_{1t})} \alpha_t \Big|\Big| s_t^{-1/4} m_{t-1} \Big|\Big|^2 \\
&+ \frac{\beta_{1t}}{2\alpha_t(1-\beta_{1t})} \Big|\Big| s_t^{1/4}(\theta_t - \theta^*) \Big|\Big|^2 \Big\} \\
&\Big(By\ formula\ equation\ 4.15 \Big) \\
&\leq \frac{1}{2(1-\beta_1)} \frac{\Big|\Big| s_1^{1/4}(\theta_1 - \theta^*) \Big|\Big|^2}{\alpha_1} \\
&+ \frac{1}{2(1-\beta_1)} \sum_{t=2}^{T} \Big[\frac{\Big|\Big| s_t^{1/4}(\theta_t - \theta^*) \Big|\Big|^2}{\alpha_t} - \frac{\Big|\Big| s_{t-1}^{1/4}(\theta_t - \theta^*) \Big|\Big|^2}{\alpha_{t-1}} \Big] \\
&+ \sum_{t=1}^{T} \Big[\frac{1}{2(1-\beta_1)} \alpha_t \Big|\Big| s_t^{-1/4} m_t \Big|\Big|^2 \Big] + \sum_{t=2}^{T} \Big[\frac{\beta_1}{2(1-\beta_1)} \alpha_{t-1} \Big|\Big| s_{t-1}^{-1/4} m_{t-1} \Big|\Big|^2 \Big] \\
&+ \sum_{t=1}^{T} \frac{\beta_{1t}}{2\alpha_t(1-\beta_{1t})} \Big|\Big| s_t^{1/4}(\theta_t - \theta^*) \Big|\Big|^2 \\
&\Big(0 \leq s_{t-1} \leq s_t, 0 \leq \alpha_t \leq \alpha_{t-1}, 0 \leq \beta_{1t} \leq \beta_1 < 1 \Big) \\
&\leq \frac{1}{2(1-\beta_1)} \frac{\Big|\Big| s_1^{1/4}(\theta_1 - \theta^*) \Big|\Big|^2}{\alpha_1} + \frac{1}{2(1-\beta_1)} \sum_{t=2}^{T} \Big|\Big| \theta_t - \theta^* \Big|\Big|^2 \Big[\frac{s_t^{1/2}}{\alpha_t} - \frac{s_{t-1}^{1/2}}{\alpha_{t-1}} \Big] \\
&+ \frac{1+\beta_1}{2(1-\beta_1)} \sum_{t=1}^{T} \alpha_t \Big|\Big| s_t^{-1/4} m_t \Big|\Big|^2 \\
&+ \sum_{t=1}^{T} \frac{\beta_{1t}}{2\alpha_t(1-\beta_{1t})} \Big|\Big| s_t^{1/4}(\theta_t - \theta^*) \Big|\Big|^2 \\
&\leq \frac{1}{2(1-\beta_1)} \frac{\Big|\Big| s_1^{1/4}(\theta_1 - \theta^*) \Big|\Big|^2}{\alpha_1} + \frac{1}{2(1-\beta_1)} \sum_{t=2}^{T} \Big|\Big| \theta_t - \theta^* \Big|\Big|^2 \Big[\frac{s_t^{1/2}}{\alpha_t} - \frac{s_{t-1}^{1/2}}{\alpha_{t-1}} \Big] \\
&+ \frac{1+\beta_1}{2(1-\beta_1)} \sum_{t=1}^{T} \alpha_t \Big|\Big| s_t^{-1/4} m_t \Big|\Big|^2 \\
&+ \frac{1}{2(1-\beta_1)} \sum_{t=1}^{T} \frac{\beta_{1t}}{\alpha_t} \Big|\Big| s_t^{1/4}(\theta_t - \theta^*) \Big|\Big|^2 \\
&\Big(since\ 0 \leq \beta_{1t} \leq \beta_1 < 1 \Big) \qquad (4.16)
\end{aligned}
$$

Now bound $\sum_{t=1}^{T} \alpha_t ||s_t^{-1/4} m_t||^2$ in Formula equation 4.16, assuming $0 < c \leq s_t, \forall t \in [T]$.

$$
\begin{aligned}
\sum_{t=1}^{T} \alpha_t \Big|\Big| s_t^{-1/4} m_t \Big|\Big|^2 &= \sum_{t=1}^{T-1} \alpha_t \Big|\Big| s_t^{-1/4} m_t \Big|\Big|^2 + \alpha_T \Big|\Big| s_T^{-1/4} m_T \Big|\Big|^2 \\
&\leq \sum_{t=1}^{T-1} \alpha_t \Big|\Big| s_t^{-1/4} m_t \Big|\Big|^2 + \frac{\alpha_T}{\sqrt{c}} \Big|\Big| m_T \Big|\Big|^2 \\
&= \sum_{t=1}^{T-1} \alpha_t \Big|\Big| s_t^{-1/4} m_t \Big|\Big|^2 + \frac{\alpha}{\sqrt{cT}} \sum_{i=1}^{d} \Big(\sum_{j=1}^{T} (1-\beta_{1,j}) g_{j,i} \prod_{k=1}^{T-j} \beta_{1,T-k+1} \Big)^2 \\
&\Big(\textit{since } m_T = \sum_{j=1}^{T} (1-\beta_{1,j}) g_{j,i} \prod_{k=1}^{T-j} \beta_{1,T-k+1} \Big) \\
&\leq \sum_{t=1}^{T-1} \alpha_t \Big|\Big| s_t^{-1/4} m_t \Big|\Big|^2 + \frac{\alpha}{\sqrt{cT}} \sum_{i=1}^{d} \Big(\sum_{j=1}^{T} g_{j,i} \prod_{k=1}^{T-j} \beta_1 \Big)^2 \\
&(\textit{since } \ 0 < \beta_{1,j} \leq \beta_1 < 1) \\
&= \sum_{t=1}^{T-1} \alpha_t \Big|\Big| s_t^{-1/4} m_t \Big|\Big|^2 + \frac{\alpha}{\sqrt{cT}} \sum_{i=1}^{d} \Big(\sum_{j=1}^{T} \beta_1^{T-j} g_{j,i} \Big)^2 \\
&\leq \sum_{t=1}^{T-1} \alpha_t \Big|\Big| s_t^{-1/4} m_t \Big|\Big|^2 + \frac{\alpha}{\sqrt{cT}} \sum_{i=1}^{d} \Big(\sum_{j=1}^{T} \beta_1^{T-j} \Big) \Big(\sum_{j=1}^{T} \beta_1^{T-j} g_{j,i}^2 \Big) \\
&\Big(Cauchy - Schwartz, \langle u, v \rangle^2 \leq \Big|\Big| u \Big|\Big|^2 \Big|\Big| v \Big|\Big|^2, u_j = \sqrt{\beta_1^{T-j}}, v_j = \sqrt{\beta_1^{T-j}} g_{j,i} \Big) \\
&= \sum_{t=1}^{T-1} \alpha_t \Big|\Big| s_t^{-1/4} m_t \Big|\Big|^2 + \frac{\alpha}{\sqrt{cT}} \sum_{i=1}^{d} \frac{1-\beta_1^T}{1-\beta_1} \sum_{j=1}^{T} \beta_1^{T-j} g_{j,i}^2 \\
&\leq \sum_{t=1}^{T-1} \alpha_t \Big|\Big| s_t^{-1/4} m_t \Big|\Big|^2 + \frac{\alpha}{\sqrt{c}(1-\beta_1)} \sum_{i=1}^{d} \sum_{j=1}^{T} \beta_1^{T-j} g_{j,i}^2 \frac{1}{\sqrt{T}} \\
&\Big(\textit{since } 1 - \beta_1^T < 1 \Big) \\
&\leq \frac{\alpha}{\sqrt{c}(1-\beta_1)} \sum_{i=1}^{d} \sum_{t=1}^{T} \sum_{j=1}^{t} \beta_1^{t-j} g_{j,i}^2 \frac{1}{\sqrt{t}} \\
&\Big(\textit{Recursively bound each term in the sum } \sum_{t=1}^{T} * \Big)
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{\alpha}{\sqrt{c}(1-\beta_1)} \sum_{i=1}^{d} \sum_{t=1}^{T} g_{t,i}^2 \sum_{j=t}^{T} \frac{\beta_1^{j-t}}{\sqrt{j}} \\
&\leq \frac{\alpha}{\sqrt{c}(1-\beta_1)} \sum_{i=1}^{d} \sum_{t=1}^{T} g_{t,i}^2 \sum_{j=t}^{T} \frac{\beta_1^{j-t}}{\sqrt{t}} \\
&\leq \frac{\alpha}{\sqrt{c}(1-\beta_1)^2} \sum_{i=1}^{d} \sum_{t=1}^{T} g_{t,i}^2 \frac{1}{\sqrt{t}} \\
&\Big(\textit{since} \ \ \sum_{j=t}^{T} \beta_1^{j-t} = \sum_{j=0}^{T-t} \beta_1^{j} = \frac{1-\beta_1^{T-t+1}}{1-\beta_1} \leq \frac{1}{1-\beta_1} \Big) \\
&\leq \frac{\alpha}{\sqrt{c}(1-\beta_1)^2} \sum_{i=1}^{d} \Big|\Big| g_{1:T,i}^2 \Big|\Big|_2 \sqrt{\sum_{t=1}^{T} \frac{1}{t}} \\
&\Big(Cauchy - Schwartz, \langle u, v \rangle \leq \Big|\Big| u \Big|\Big| \Big|\Big| v \Big|\Big|, u_t = g_{t,i}^2, v_t = \frac{1}{\sqrt{t}} \Big) \\
&\leq \frac{\alpha\sqrt{1+\log T}}{\sqrt{c}(1-\beta_1)^2} \sum_{i=1}^{d} \Big|\Big| g_{1:T,i}^2 \Big|\Big|_2 \ \ \Big(\textit{since} \ \ \sum_{t=1}^{T} \frac{1}{t} \leq 1 + \log T \Big) \qquad (4.17)
\end{aligned}
$$

Apply formula equation 4.17 to equation 4.16, we have:

$$
\begin{aligned}
\sum_{t=1}^{T} f_t(\theta_t) - f_t(\theta^*) &\leq \frac{1}{2(1-\beta_1)} \frac{\Big|\Big| s_1^{1/4}(\theta_1 - \theta^*) \Big|\Big|^2}{\alpha_1} + \frac{1}{2(1-\beta_1)} \sum_{t=2}^{T} \Big|\Big| \theta_t - \theta^* \Big|\Big|^2 \Big[\frac{s_t^{1/2}}{\alpha_t} - \frac{s_{t-1}^{1/2}}{\alpha_{t-1}} \Big] \\
&+ \frac{1+\beta_1}{2(1-\beta_1)} \sum_{t=1}^{T} \alpha_t \Big|\Big| s_t^{-1/4} m_t \Big|\Big|^2 \\
&+ \frac{1}{2(1-\beta_1)} \sum_{t=1}^{T} \frac{\beta_{1t}}{\alpha_t} \Big|\Big| s_t^{1/4}(\theta_t - \theta^*) \Big|\Big|^2 \\
&\leq \frac{1}{2(1-\beta_1)} \frac{\Big|\Big| s_1^{1/4}(\theta_1 - \theta^*) \Big|\Big|^2}{\alpha_1} + \frac{1}{2(1-\beta_1)} \sum_{t=2}^{T} \Big|\Big| \theta_t - \theta^* \Big|\Big|^2 \Big[\frac{s_t^{1/2}}{\alpha_t} - \frac{s_{t-1}^{1/2}}{\alpha_{t-1}} \Big] \\
&+ \frac{(1+\beta_1)\alpha\sqrt{1+\log T}}{2\sqrt{c}(1-\beta_1)^3} \sum_{i=1}^{d} \Big|\Big| g_{1:T,i}^2 \Big|\Big|_2 \\
&+ \frac{1}{2(1-\beta_1)} \sum_{t=1}^{T} \frac{\beta_{1t}}{\alpha_t} \Big|\Big| s_t^{1/4}(\theta_t - \theta^*) \Big|\Big|^2 \\
&\Big(\textit{By formula equation 4.17} \Big)
\end{aligned}
$$

$$
\begin{aligned}
&\leq \frac{1}{2(1-\beta_1)} \sum_{i=1}^{d} \frac{s_{1,i}^{1/2} D_\infty^2}{\alpha_1} + \frac{1}{2(1-\beta_1)} \sum_{t=2}^{T} \sum_{i=1}^{d} D_\infty^2 \Big[\frac{s_{t,i}^{1/2}}{\alpha_t} - \frac{s_{t-1,i}^{1/2}}{\alpha_{t-1}} \Big] \\
&+ \frac{(1+\beta_1)\alpha\sqrt{1+\log T}}{2\sqrt{c}(1-\beta_1)^3} \sum_{i=1}^{d} \Big|\Big| g_{1:T,i}^2 \Big|\Big|_2 \\
&+ \frac{D_\infty^2}{2(1-\beta_1)} \sum_{t=1}^{T} \sum_{i=1}^{d} \frac{\beta_{1t} s_{t,i}^{1/2}}{\alpha_t} \\
&\Big(\textit{since } x \in \mathcal{F}, \textit{with bounded diameter } D_\infty, \textit{and } \frac{s_{t,i}^{1/2}}{\alpha_t} \geq \frac{s_{t-1,i}^{1/2}}{\alpha_{t-1}} \textit{ by assumption.} \Big) \\
&\leq \frac{D_\infty^2 \sqrt{T}}{2\alpha(1-\beta_1)} \sum_{i=1}^{d} s_{T,i}^{1/2} + \frac{(1+\beta_1)\alpha\sqrt{1+\log T}}{2\sqrt{c}(1-\beta_1)^3} \sum_{i=1}^{d} \Big|\Big| g_{1:T,i}^2 \Big|\Big|_2 \\
&+ \frac{D_\infty^2}{2(1-\beta_1)} \sum_{t=1}^{T} \sum_{i=1}^{d} \frac{\beta_{1t} s_{t,i}^{1/2}}{\alpha_t} \\
&\Big(\alpha_t \geq \alpha_{t+1} \textit{ and perform telescope sum} \Big) \qquad (4.18)
\end{aligned}
$$

□

Corollary 4.5.1.1. *Suppose* $\beta_{1,t} = \beta_1 \lambda^t, \;\; 0 < \lambda < 1$ *in Theorem equation 4.5.1, then we have:*

$$
\begin{aligned}
\sum_{t=1}^{T} f_t(\theta_t) - f_t(\theta^*) \leq \frac{D_\infty^2 \sqrt{T}}{2\alpha(1-\beta_1)} \sum_{i=1}^{d} s_{T,i}^{1/2} + \frac{(1+\beta_1)\alpha\sqrt{1+\log T}}{2\sqrt{c}(1-\beta_1)^3} \sum_{i=1}^{d} \Big|\Big| g_{1:T,i}^2 \Big|\Big|_2 \\
+ \frac{D_\infty^2 \beta_1 G_\infty}{2(1-\beta_1)(1-\lambda)^2 \alpha} \qquad (4.19)
\end{aligned}
$$

Proof: By sum of arithmetico-geometric series, we have:

$$
\sum_{t=1}^{T} \lambda^{t-1} \sqrt{t} \leq \sum_{t=1}^{T} \lambda^{t-1} t \leq \frac{1}{(1-\lambda)^2} \qquad (4.20)
$$

Plugging equation 4.20 into equation 4.18, we can derive the results above. □

4.5.2 Convergence of AdaBelief for non-convex stochastic optimization

Assumptions

- A1, f is differentiable and has $L-Lipschitz$ gradient, $||\nabla f(x)-\nabla f(y)|| \leq L||x-y||,\ \forall x, y.\ f$ is also lower bounded.
- A2, at time t, the algorithm can access a bounded noisy gradient, the true gradient is also bounded. $i.e.\ ||\nabla f(\theta_t)|| \leq H,\ ||g_t|| \leq H,\ \ \forall t > 1.$
- A3, The noisy gradient is unbiased, and has independent noise. $i.e.\ g_t = \nabla f(\theta_t) + \zeta_t, \mathbb{E}\zeta_t = 0, \zeta_t \perp \zeta_j,\ \forall j, t \in \mathbb{N}, t \neq j$

Theorem 4.5.2. *[24] Suppose assumptions A1-A3 are satisfied, $\beta_{1,t}$ is chosen such that $0 \leq \beta_{1,t+1} \leq \beta_{1,t} < 1, 0 < \beta_2 < 1, \forall t > 0$. For some constant G, $\left|\left|\alpha_t \frac{m_t}{\sqrt{s_t}}\right|\right| \leq G, \forall t$. Then Adam-type algorithms yield*

$$\mathbb{E}\Big[\sum_{t=1}^{T} \alpha_t \langle \nabla f(\theta_t), \nabla f(\theta_t)/\sqrt{s_t}\rangle\Big] \leq$$
$$\mathbb{E}\left[C_1 \sum_{t=1}^{T} \left|\left|\alpha_t g_t/\sqrt{s_t}\right|\right|^2 + C_2 \sum_{t=1}^{T} \left|\left|\frac{\alpha_t}{\sqrt{s_t}} - \frac{\alpha_{t-1}}{\sqrt{s_{t-1}}}\right|\right|_1 + C_3 \sum_{t=1}^{T} \left|\left|\frac{\alpha_t}{\sqrt{s_t}} - \frac{\alpha_{t-1}}{\sqrt{s_{t-1}}}\right|\right|^2\right] + C_4 \tag{4.21}$$

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 randomness corresponding to $\{g_t\}$.

Furthermore, let $\gamma_t := \min_{j\in[d]} \min_{\{g_i\}_{i=1}^t} \alpha_i/(\sqrt{s_i})_j$ denote the minimum possible value of effective stepsize at time t over all possible coordinate and past gradients $\{g_i\}_{i=1}^t$. The

convergence rate of Adam-type algorithm is given by

$$\min_{t\in[T]} \mathbb{E}\left[\left|\left|\nabla f(\theta_t)\right|\right|^2\right] = O\left(\frac{s_1(T)}{s_2(T)}\right) \tag{4.22}$$

where $s_1(T)$ is defined through the upper bound of RHS of equation 4.21, and $\sum_{t=1}^{T}\gamma_t = \Omega(s_2(T))$

Proof: Full proof is in [24]. □

Theorem 4.5.3. *Assume $\min_{j\in[d]}(s_1)_j \geq c > 0$, noise in gradient has bounded variance, $\mathrm{Var}(g_t) = \sigma_t^2 \leq \sigma^2, \forall t \in \mathbb{N}$, then the AdaBelief algorithm satisfies:*

$$\begin{aligned}\min_{t\in[T]} \mathbb{E}\left|\left|\nabla f(\theta_t)\right|\right|^2 &\leq \frac{H}{\sqrt{T}\alpha}\Big[\frac{C_1\alpha^2(H^2+\sigma^2)(1+\log T)}{c} + C_2\frac{d\alpha}{\sqrt{c}} + C_3\frac{d\alpha^2}{c} + C_4\Big] \\ &= \frac{1}{\sqrt{T}}(Q_1 + Q_2\log T)\end{aligned}$$

where

$$\begin{aligned}Q_1 &= \frac{H}{\alpha}\Big[\frac{C_1\alpha^2(H^2+\sigma^2)}{c} + C_2\frac{d\alpha}{\sqrt{c}} + C_3\frac{d\alpha^2}{c} + C_4\Big] \\ Q_2 &= \frac{HC_1\alpha(H^2+\sigma^2)}{c}\end{aligned}$$

Proof: We first derive an upper bound of the RHS of formula equation 4.21, then derive a lower bound of the LHS of equation 4.21.

$$\begin{aligned}\mathbb{E}\Big[\sum_{t=1}^{T}\left|\left|\alpha_t g_t/\sqrt{s_t}\right|\right|^2\Big] &\leq \frac{1}{c}\mathbb{E}\Big[\sum_{t=1}^{T}\sum_{i=1}^{d}(\alpha_{t,i}g_{t,i})^2\Big] \quad \Big(\textit{since } 0 < c \leq s_t, \forall t \in [T]\Big) \\ &= \frac{1}{c}\sum_{i=1}^{d}\sum_{t=1}^{T}\alpha_t^2\mathbb{E}(g_{t,i})^2\end{aligned}$$

$$= \frac{1}{c}\sum_{t=1}^{T}\alpha_t^2\mathbb{E}\Big[\Big|\Big|\nabla f(\theta_t)\Big|\Big|^2 + \Big|\Big|\sigma_t\Big|\Big|^2\Big] \tag{4.23}$$

$$\begin{aligned}
\mathbb{E}\Big[\sum_{t=1}^{T}\Big|\Big|\frac{\alpha_t}{\sqrt{s_t}} - \frac{\alpha_{t-1}}{\sqrt{s_{t-1}}}\Big|\Big|_1\Big] &= \mathbb{E}\Big[\sum_{i=1}^{d}\sum_{t=1}^{T}\frac{\alpha_{t-1}}{\sqrt{s_{t-1,i}}} - \frac{\alpha_t}{\sqrt{s_t,i}}\Big] \\
&\Big(\textit{since } \alpha_t \le \alpha_{t-1}, s_{t,i} \ge s_{t-1,i}\Big) \\
&= \mathbb{E}\Big[\sum_{i=1}^{d}\frac{\alpha_1}{\sqrt{s_{1,i}}} - \frac{\alpha_T}{\sqrt{s_{T,i}}}\Big] \\
&\le \mathbb{E}\Big[\sum_{i=1}^{d}\frac{\alpha_1}{\sqrt{s_{1,i}}}\Big] \\
&\le \frac{d\alpha}{\sqrt{c}} \quad \Big(\textit{since } 0 < c \le s_t, 0 \le \alpha_t \le \alpha_1 = \alpha, \forall t\Big)
\end{aligned} \tag{4.24}$$

$$\begin{aligned}
\mathbb{E}\Big[\sum_{t=1}^{T}\Big|\Big|\frac{\alpha_t}{\sqrt{s_t}} - \frac{\alpha_{t-1}}{\sqrt{s_{t-1}}}\Big|\Big|^2\Big] &= \mathbb{E}\Big[\sum_{t=1}^{T}\sum_{i=1}^{d}\Big(\frac{\alpha_t}{\sqrt{s_t}} - \frac{\alpha_{t-1}}{\sqrt{s_{t-1}}}\Big)_i^2\Big] \\
&\le \mathbb{E}\Big[\sum_{t=1}^{T}\sum_{i=1}^{d}\Big|\frac{\alpha_t}{\sqrt{s_t}} - \frac{\alpha_{t-1}}{\sqrt{s_{t-1}}}\Big|_i\frac{\alpha}{\sqrt{c}}\Big] \\
&\Big(\textit{Since } \Big|\frac{\alpha_t}{\sqrt{s_t}} - \frac{\alpha_{t-1}}{\sqrt{s_{t-1}}}\Big| = \frac{\alpha_{t-1}}{\sqrt{s_{t-1}}} - \frac{\alpha_t}{\sqrt{s_t}} \le \frac{\alpha_{t-1}}{\sqrt{s_{t-1}}} \le \frac{\alpha}{\sqrt{c}}\Big) \\
&\le \frac{d\alpha^2}{c} \quad \Big(\textit{By equation 5.25}\Big)
\end{aligned} \tag{4.25}$$

Next we derive the lower bound of LHS of equation 4.21.

$$\mathbb{E}\Big[\sum_{t=1}^{T}\alpha_t\langle\nabla f(\theta_t), \frac{\nabla f(\theta_t)}{\sqrt{s_t}}\rangle\Big] \ge \frac{1}{H}\mathbb{E}\Big[\sum_{t=1}^{T}\alpha_t\Big|\Big|\nabla f(\theta_t)\Big|\Big|^2\Big] \ge \frac{\alpha\sqrt{T}}{H}\min_{t\in[T]}\mathbb{E}\Big|\Big|\nabla f(\theta_t)\Big|\Big|^2 \tag{4.26}$$

Combining equation 4.23, equation 5.25, equation 5.83 and equation 5.85 to equa-

tion 4.21, we have:

$$\frac{\alpha\sqrt{T}}{H}\min_{t\in[T]}\mathbb{E}\Big|\Big|\nabla f(\theta_t)\Big|\Big|^2 \leq \mathbb{E}\Big[\sum_{t=1}^{T}\alpha_t\langle\nabla f(\theta_t),\frac{\nabla f(\theta_t)}{\sqrt{s_t}}\rangle\Big]$$

$$\leq \mathbb{E}\Big[C_1\sum_{t=1}^{T}\Big|\Big|\alpha_t g_t/\sqrt{s_t}\Big|\Big|^2 + C_2\sum_{t=1}^{T}\Big|\Big|\frac{\alpha_t}{\sqrt{s_t}}-\frac{\alpha_{t-1}}{\sqrt{s_{t-1}}}\Big|\Big|_1 + C_3\sum_{t=1}^{T}\Big|\Big|\frac{\alpha_t}{\sqrt{s_t}}-\frac{\alpha_{t-1}}{\sqrt{s_{t-1}}}\Big|\Big|^2\Big] + C_4$$

$$\leq \frac{C_1}{c}\sum_{t=1}^{T}\mathbb{E}\Big[\alpha_t^2\Big|\Big|\nabla f(\theta_t)\Big|\Big|^2 + \alpha_t^2\Big|\Big|\sigma_t\Big|\Big|^2\Big] + C_2\frac{d\alpha}{\sqrt{c}} + C_3\frac{d\alpha^2}{c} + C_4 \tag{4.27}$$

$$\leq \frac{C_1}{c}\sum_{t=1}^{T}\mathbb{E}\Big[\alpha_t^2(H^2+\sigma^2)\Big] + C_2\frac{d\alpha}{\sqrt{c}} + C_3\frac{d\alpha^2}{c} + C_4$$

$$\leq \frac{C_1\alpha^2(H^2+\sigma^2)(1+\log T)}{c} + C_2\frac{d\alpha}{\sqrt{c}} + C_3\frac{d\alpha^2}{c} + C_4 \tag{4.28}$$

$$\Big(\textit{since } \alpha_t = \frac{\alpha}{\sqrt{t}},\ \sum_{t=1}^{T}\frac{1}{t}\leq 1+\log T\Big)$$

Re-arranging above inequality, we have

$$\min_{t\in[T]}\mathbb{E}\Big|\Big|\nabla f(\theta_t)\Big|\Big|^2 \leq \frac{H}{\sqrt{T}\alpha}\Big[\frac{C_1\alpha^2(H^2+\sigma^2)(1+\log T)}{c} + C_2\frac{d\alpha}{\sqrt{c}} + C_3\frac{d\alpha^2}{c} + C_4\Big]$$
$$= \frac{1}{\sqrt{T}}(Q_1 + Q_2\log T) \tag{4.29}$$

where

$$Q_1 = \frac{H}{\alpha}\Big[\frac{C_1\alpha^2(H^2+\sigma^2)}{c} + C_2\frac{d\alpha}{\sqrt{c}} + C_3\frac{d\alpha^2}{c} + C_4\Big] \tag{4.30}$$

$$Q_2 = \frac{HC_1\alpha(H^2+\sigma^2)}{c} \tag{4.31}$$

□

Corollary 4.5.3.1. *If $c > C_1 H$ and assumptions for Theorem 4.5.2 are satisfied, we have:*

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\Big[\alpha_t^2\Big|\Big|\nabla f(\theta_t)\Big|\Big|^2\Big] \leq \frac{1}{T}\frac{1}{\frac{1}{H}-\frac{C_1}{c}}\Bigg[\frac{C_1\alpha^2\sigma^2}{c}\big(1+\log T\big) + C_2\frac{d\alpha}{\sqrt{c}} + C_3\frac{d\alpha^2}{c} + C_4\Bigg] \tag{4.32}$$

Proof: From equation 5.85 and equation 4.27, we have

$$\begin{aligned}\frac{1}{H}\mathbb{E}\Big[\sum_{t=1}^{T}\alpha_t\Big|\Big|\nabla f(\theta_t)\Big|\Big|^2\Big] &\leq \mathbb{E}\Big[\sum_{t=1}^{T}\alpha_t\langle\nabla f(\theta_t), \frac{\nabla f(\theta_t)}{\sqrt{s_t}}\rangle\Big] \\ &\leq \frac{C_1}{c}\sum_{t=1}^{T}\mathbb{E}\Big[\alpha_t^2\Big|\Big|\nabla f(\theta_t)\Big|\Big|^2 + \alpha_t^2\Big|\Big|\sigma_t\Big|\Big|^2\Big] + C_2\frac{d\alpha}{\sqrt{c}} + C_3\frac{d\alpha^2}{c} + C_4\end{aligned} \tag{4.33}$$

By re-arranging, we have

$$\begin{aligned}\big(\frac{1}{H}-\frac{C_1}{c}\big)\sum_{t=1}^{T}\mathbb{E}\Big[\alpha_t^2\Big|\Big|\nabla f(\theta_t)\Big|\Big|^2\Big] &\leq \frac{C_1}{c}\sum_{t=1}^{T}\mathbb{E}\Big[\alpha_t^2\Big|\Big|\sigma_t\Big|\Big|^2\Big] + C_2\frac{d\alpha}{\sqrt{c}} + C_3\frac{d\alpha^2}{c} + C_4 \\ &\leq \frac{C_1\alpha^2\sigma^2}{c}\big(1+\log T\big) + C_2\frac{d\alpha}{\sqrt{c}} + C_3\frac{d\alpha^2}{c} + C_4\end{aligned} \tag{4.34}$$

By assumption, $\frac{1}{H}-\frac{C_1}{c} > 0$, then we have

$$\sum_{t=1}^{T}\mathbb{E}\Big[\alpha_t^2\Big|\Big|\nabla f(\theta_t)\Big|\Big|^2\Big] \leq \frac{1}{\frac{1}{H}-\frac{C_1}{c}}\Bigg[\frac{C_1\alpha^2\sigma^2}{c}\big(1+\log T\big) + C_2\frac{d\alpha}{\sqrt{c}} + C_3\frac{d\alpha^2}{c} + C_4\Bigg] \tag{4.35}$$

□

4.5.3 Analysis on convergence conditions of Asynchronous AdaBelief

Convergence analysis for Problem 4.7

Lemma 4.5.3.1. *There exists an online convex optimization problem where Adam (and RMSprop) has non-zero average regret, and one of the problem is in the form*

$$f_t(x) = \begin{cases} Px, & \textit{if } t \bmod P = 1 \\ -x, & \textit{Otherwise} \end{cases} \quad x \in [-1, 1], \exists P \in \mathbb{N}, P \geq 3 \tag{4.36}$$

Proof. See [127] Thm.1 for proof. □

Lemma 4.5.3.2. *For the problem defined above, there's a threshold of β_2 above which RMSprop converge.*

Proof. See [106] for details. □

Lemma 4.5.3.3. *For the problem defined by Eq. equation 4.36, ACProp algorithm converges $\forall \beta_1, \beta_2 \in (0, 1), \forall P \in \mathbb{N}, P \geq 3$.*

Proof. We analyze the limit behavior of ACProp algorithm. Since the observed gradient is periodic with an integer period P, we analyze one period from with indices from kP to $kP + P$, where k is an integer going to $+\infty$.

From the update of ACProp, we observe that:

$$m_{kP} = (1 - \beta_1) \sum_{i=1}^{kP} \beta_1^{kP-i} \times (-1) + (1 - \beta_1) \sum_{j=0}^{k-1} \beta_1^{kP-(jP+1)} (P + 1) \tag{4.37}$$

$$\Big(\textit{For each observation with gradient } P, \textit{we break it into } P = -1 + (P + 1)\Big)$$

$$= -(1 - \beta_1) \sum_{i=1}^{kP} \beta_1^{kP-i} + (1 - \beta_1)(P + 1)\beta_1^{-1} \sum_{j=0}^{k-1} \beta_1^{P(k-j)} \tag{4.38}$$

$$= -(1 - \beta_1^{kP}) + (1 - \beta_1)(P + 1)\beta_1^{P-1} \frac{1 - \beta_1^{(k-1)P}}{1 - \beta_1^P} \tag{4.39}$$

$$\lim_{k\to\infty} m_{kP} = -1 + (P+1)(1-\beta_1)\beta_1^{P-1}\frac{1}{1-\beta_1^P} = \frac{(P+1)\beta_1^{P-1} - P\beta_1^P - 1}{1-\beta_1^P} \tag{4.40}$$

$$\Big(\textit{Since } \beta_1 \in [0,1)\Big)$$

Next, we derive $\lim_{k\to\infty} S_{kP}$. Note that the observed gradient is periodic, and $\lim_{k\to\infty} m_{kP} = \lim_{k\to\infty} m_{kP+P}$, hence $\lim_{k\to\infty} S_{kP} = \lim_{k\to\infty} S_{kP+P}$. Start from index kP, we derive variables up to $kP+P$ with ACProp algorithm.

$$index = kP,$$

$$m_{kP}, S_{kP} \tag{4.41}$$

$$index = kP+1,$$

$$m_{kP+1} = \beta_1 m_0 + (1-\beta_1)P \tag{4.42}$$

$$S_{kP+1} = \beta_2 S_{kP} + (1-\beta_2)(P - m_{kP})^2 \tag{4.43}$$

$$index = kP+2,$$

$$m_{kP+2} = \beta_1 m_{kP+1} + (1-\beta_1)\times(-1) \tag{4.44}$$

$$= \beta_1^2 m_{kP} + (1-\beta_1)\beta_1 P + (1-\beta_1)\times(-1) \tag{4.45}$$

$$S_{kP+2} = \beta_2 S_{kP+1} + (1-\beta_2)(-1 - m_{kP+1})^2 \tag{4.46}$$

$$= \beta_2^2 S_{kP} + (1-\beta_2)\beta_2(P - m_{kP})^2 + (1-\beta_2)\big[\beta_1(P - m_{kP}) - (P+1)\big]^2 \tag{4.47}$$

$$index = kP+3,$$

$$m_{kP+3} = \beta_1 m_{kP+2} + (1-\beta_1)\times(-1) \tag{4.48}$$

$$= \beta_1^3 m_{kP} + (1-\beta_1)\beta_1^2 P + (1-\beta_1)\beta_1\times(-1) + (1-\beta_1)\times(-1) \tag{4.49}$$

$$S_{kP+3} = \beta_2 S_2 + (1-\beta_2)(-1 - m_{kP+2})^2 \tag{4.50}$$

$$= \beta_2^3 S_{kP} + (1-\beta_2)\beta_2^2(P - m_{kP})^2$$

$$+ (1-\beta_2)\beta_2\big[\beta_1(P-m_{kP}) - (P+1)\big]^2(\beta_2+\beta_1^2) \tag{4.51}$$

$index = kP+4$,

$$m_{kP+4} = \beta_1^4 m_{kP} + (1-\beta_1)\beta_1^3 P + (-1)(1-\beta_1)(\beta_1^2+\beta_1+1) \tag{4.52}$$

$$S_{kP+4} = \beta_2 S_{kP+3} + (1-\beta_2)(-1-m_{kP+3})^2 \tag{4.53}$$

$$= \beta_2^4 S_{kP} + (1-\beta_2)\beta_2^3(P-m_{kP})^2$$

$$+ (1-\beta_2)\beta_2\big[\beta_1(P-m_{kP}) - (P+1)\big]^2(\beta_2^2+\beta_2\beta_1^2+\beta_1^4) \tag{4.54}$$

$\cdots$

$index = kP+P$,

$$m_{kP+P} = \beta_1^P m_{kP} + (1-\beta_1)\beta_1^{P-1}P + (-1)(1-\beta_1)\big[\beta_1^{P-2}+\beta_1^{P-3}+...+1\big] \tag{4.55}$$

$$= \beta_1^P m_{kP} + (1-\beta_1)\beta_1^{P-1}P + (\beta_1-1)\frac{1-\beta_1^{P-1}}{1-\beta 1} \tag{4.56}$$

$$S_{kP+P} = \beta_2^P S_{kP} + (1-\beta_2)\beta_2^{P-1}(P-m_{kP})^2$$

$$+ (1-\beta_2)\big[\beta_1(P-m_{kP}) - (P+1)\big]^2\big(\beta_2^{P-2}+\beta_2^{P-3}\beta_1^2+...+\beta_2^0\beta_1^{2P-4}\big) \tag{4.57}$$

$$= \beta_2^P S_{kP} + (1-\beta_2)\beta_2^{P-1}(P-m_{kP})^2$$

$$+ (1-\beta_2)\big[\beta_1(P-m_{kP}) - (P+1)\big]^2\beta_2^{P-2}\frac{1-(\beta_1^2/\beta_2)^{P-1}}{1-(\beta_1^2/\beta_2)} \tag{4.58}$$

As k goes to $+\infty$, we have

$$\lim_{k\to\infty} m_{kP+P} = \lim_{k\to\infty} m_{kP} \tag{4.59}$$

$$\lim_{k\to\infty} S_{kP+P} = \lim_{k\to\infty} S_{kP} \tag{4.60}$$

From Eq. equation 4.56 we have:

$$m_{kP+P} = \frac{(P+1)\beta_1^{P-1} - P\beta_1^P - 1}{1-\beta_1^P} \tag{4.61}$$

which matches our result in Eq. equation 4.41. Similarly, from Eq. equation 4.58, take limit of $k \to \infty$, and combine with Eq. equation 4.60, we have

$$\lim_{k\to\infty} S_{kP} = \frac{1-\beta_2}{1-\beta_2^P}\left[\beta_2^{P-1}(P - \lim_{k\to\infty} m_{kP})^2 + \left[\beta_1(P - \lim_{k\to\infty} m_{kP}) - (P+1)\right]^2 \beta_2^{P-2}\frac{1-(\beta_1^2/\beta_2)^{P-1}}{1-(\beta_1^2/\beta_2)}\right] \tag{4.62}$$

Since we have the exact expression for the limit, it's trivial to check that

$$S_i \geq S_{kP}, \;\; \forall i \in [kP+1, kP+P], i \in \mathbb{N}, k \to \infty \tag{4.63}$$

Intuitively, suppose for some time period, we only observe a constant gradient -1 without observing the outlier gradient (P); the longer the length of this period, the smaller is the corresponding S value, because S records the difference between observations. Note that since last time that outlier gradient (P) is observed (at index $kP+1-P$), index kP has the longest distance from index $kP+1-P$ without observing the outlier gradient (P). Therefore, S_{kP} has the smallest value within a period of P as k goes to infinity.

For step $kP+1$ to $kP+P$, the update on parameter is:

$$index = kP+1, -\Delta_x^{kP+1} = \frac{\alpha_0}{\sqrt{kP+1}}\frac{P}{\sqrt{S_{kP}}+\epsilon} \tag{4.64}$$

$$index = kP+2, -\Delta_x^{kP+2} = \frac{\alpha_0}{\sqrt{kP+2}}\frac{-1}{\sqrt{S_{kP+1}}+\epsilon} \tag{4.65}$$

$$\ldots$$

$$index = kP+P, -\Delta_x^{kP+P} = \frac{\alpha_0}{\sqrt{kP+P}}\frac{-1}{\sqrt{S_{kP+P-1}}+\epsilon} \tag{4.66}$$

So the negative total update within this period is:

$$\frac{\alpha_0}{\sqrt{kP+1}}\frac{P}{\sqrt{S_{kP}}+\epsilon} - \underbrace{\left[\frac{\alpha_0}{\sqrt{kP+2}}\frac{1}{\sqrt{S_{kP+1}}+\epsilon} + ... + \frac{\alpha_0}{\sqrt{kP+P}}\frac{1}{\sqrt{S_{kP+P}}+\epsilon}\right]}_{P-1 \text{ terms}} \tag{4.67}$$

$$\geq \frac{\alpha_0}{\sqrt{kP+1}}\frac{P}{\sqrt{S_{kP}}+\epsilon} - \underbrace{\left[\frac{\alpha_0}{\sqrt{kP+1}}\frac{1}{\sqrt{S_{kP}}+\epsilon} + ... + \frac{\alpha_0}{\sqrt{kP+1}}\frac{1}{\sqrt{S_{kP}}+\epsilon}\right]}_{P-1 \text{ terms}} \tag{4.68}$$

$$\Big(\textit{Since } S_{kP} \textit{ is the minimum within the period}\Big)$$

$$= \frac{\alpha_0}{\sqrt{S_{kP}}+\epsilon}\frac{1}{\sqrt{kP+1}} \tag{4.69}$$

where α_0 is the initial learning rate. Note that the above result hold for every period of length P as k gets larger. Therefore, for some K such that for every $k > K$, m_{kP} and S_{kP} are close enough to their limits, the total update after K is:

$$\sum_{k=K}^{\infty} \frac{\alpha_0}{\sqrt{S_{kP}}+\epsilon}\frac{1}{\sqrt{kP+1}} \approx \frac{\alpha_0}{\sqrt{\lim_{k\to\infty} S_{kP}}+\epsilon}\frac{1}{\sqrt{P}}\sum_{k=K}^{\infty}\frac{1}{\sqrt{k}} \quad \textit{If } K \textit{ is sufficiently large} \tag{4.70}$$

where $\lim_{k\to\infty} S_{kP}$ is a constant determined by Eq. equation 4.62. Note that this is the negative update; hence ACProp goes to the negative direction, which is what we expected for this problem. Also considering that $\sum_{k=K}^{\infty}\frac{1}{\sqrt{k}} \to \infty$, hence ACProp can go arbitrarily far in the correct direction if the algorithm runs for infinitely long, therefore the bias caused by first K steps will vanish with running time. Furthermore, since x lies in the bounded region of $[-1, 1]$, if the updated result falls out of this region, it can always be clipped. Therefore, for this problem, ACProp always converge to $x = -1, \forall \beta_1, \beta_2 \in (0, 1)$. When $\beta_2 = 1$, the denominator won't update, and ACProp reduces to SGD (with momentum), and it's shown to converge. □

Lemma 4.5.3.4. *For any constant $\beta_1, \beta_2 \in [0, 1)$ such that $\beta_1 < \sqrt{\beta_2}$, there is a stochastic convex optimization problem for which Adam does not converge to the optimal solution. One example of such stochastic problem is:*

$$f_t(x) = \begin{cases} Px & \text{with probability } \frac{1+\delta}{P+1} \\ -x & \text{with probability } \frac{P-\delta}{P+1} \end{cases} \quad x \in [-1, 1] \tag{4.71}$$

Proof. See Thm.3 in [127]. □

Lemma 4.5.3.5. *For the stochastic problem defined by Eq. equation 4.71, ACProp converge to the optimal solution, $\forall \beta_1, \beta_2 \in (0, 1)$.*

Proof. The update at step t is:

$$\Delta_x^t = -\frac{\alpha_0}{\sqrt{t}} \frac{g_t}{\sqrt{S_{t-1}} + \epsilon} \tag{4.72}$$

Take expectation conditioned on observations up to step $t-1$, we have:

$$\mathbb{E}\Delta_x^t = -\frac{\alpha_0}{\sqrt{t}} \frac{\mathbb{E}_t g_t}{\sqrt{S_{t-1}} + \epsilon} \tag{4.73}$$

$$= -\frac{\alpha_0}{\sqrt{t}\Big(\sqrt{S_{t-1}} + \epsilon\Big)} \mathbb{E}_t g_t \tag{4.74}$$

$$= -\frac{\alpha_0}{\sqrt{t}\Big(\sqrt{S_{t-1}} + \epsilon\Big)} \Big[P\frac{1+\delta}{P+1} - \frac{P-\delta}{P+1}\Big] \tag{4.75}$$

$$= -\frac{\alpha_0 \delta}{\sqrt{t}\Big(\sqrt{S_{t-1}} + \epsilon\Big)} \tag{4.76}$$

$$\leq -\frac{\alpha_0 \delta}{\sqrt{t}\Big(P + 1 + \epsilon\Big)} \tag{4.77}$$

where the last inequality is due to $S_t \leq (P+1)^2$, because S_t is a smoothed version of squared difference between gradients, and the maximum difference in gradient is $P + 1$.

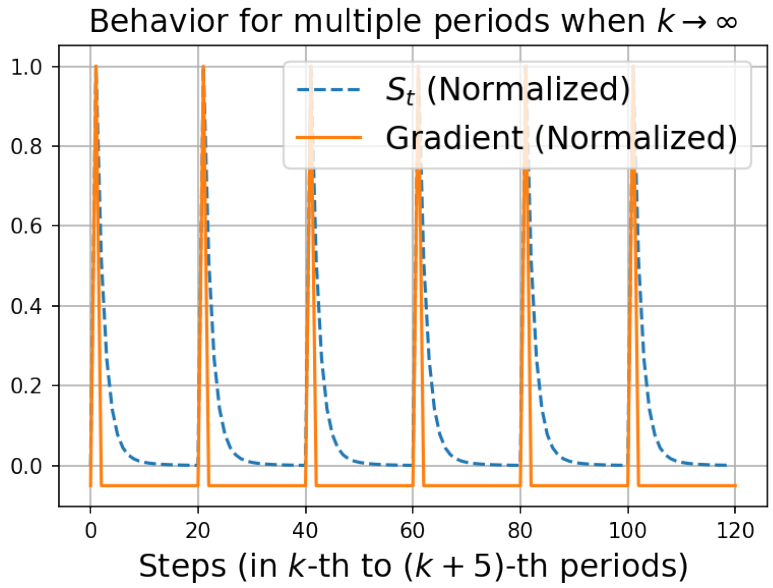


Figure 4.13: Behavior of S_t and g_t in ACProp of multiple periods for problem (1). Note that as $k \to \infty$, the behavior of ACProp is periodic.

Therefore, for every step, ACProp is expected to move in the negative direction, also considering that $\sum_{t=1}^{\infty} \frac{1}{\sqrt{t}} \to \infty$, and whenever $x < -1$ we can always clip it to -1, hence ACProp will drift x to -1, which is the optimal value. □

Numerical validations

We validate our analysis above in numerical experiments, and plot the curve of S_t and g_t for multiple periods (as $k \to \infty$) in Fig. 4.13 and zoom in to a single period in Fig. 4.14. Note that the largest gradient P (normalized as 1) appears at step $kP+1$, and S takes it minimal at step kP (e.g. S_{kP} is the smallest number within a period). Note the update for step $kP+1$ is $g_{kP+1}/\sqrt{S_{kP}}$, it's the largest gradient divided the smallest denominator, hence the net update within a period pushes x towards the optimal point.

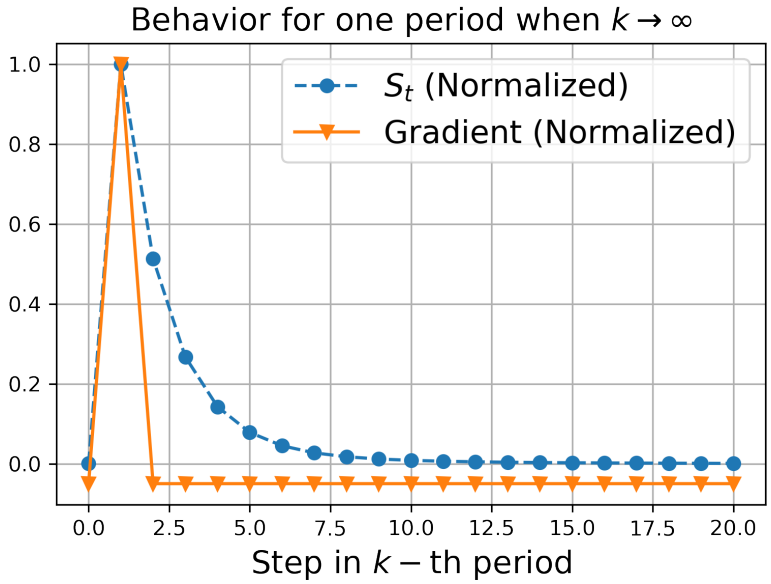


Figure 4.14: Behavior of S_t and g_t in ACProp of one period for problem (1).

Convergence analysis for Problem 4.8

Lemma 4.5.3.6. *For the problem defined by Eq. equation 4.78, consider the hyper-parameter tuple (β_1, β_2, P), there exists cases where ACProp converges but AdaShift with $n = 1$ diverges, but not vice versa.*

$$f_t(x) = \begin{cases} P/2 \times x, & t\%P == 1 \\ -x, & t\%P == P-2 \\ 0, & \textit{otherwise} \end{cases} \quad P > 3, P \in \mathbb{N}, x \in [0,1]. \tag{4.78}$$

Proof. The proof is similar to Lemma. 4.5.3.3,we derive the limit behavior of different methods.

$$index = kP,$$

$$m_{kP}, v_{kP}, s_{kP}$$

$index = kP + 1,$

$$m_{kP+1} = m_{kP}\beta_1 + (1-\beta_1)P/2 \tag{4.79}$$

$$v_{kP+1} = v_{kP}\beta_2 + (1-\beta_2)P^2/4 \tag{4.80}$$

$$s_{kP+1} = s_{kP}\beta_2 + (1-\beta_2)(P/2 - m_{kP})^2 \tag{4.81}$$

$$...$$

$index = kP + P - 2,$

$$m_{kP+P-2} = m_{kP}\beta_1^{P-2} + (1-\beta_1)\frac{P}{2}\beta_1^{P-3} + (1-\beta_1)\times(-1) \tag{4.82}$$

$$v_{kP+P-2} = v_{kP}\beta_2^{P-2} + (1-\beta_2)\frac{P^2}{4}\beta_2^{P-3} + (1-\beta_2) \tag{4.83}$$

$$\begin{aligned} s_{kP+P-2} &= s_{kP}\beta_2^{P-2} + (1-\beta_2)\beta_2^{P-3}(\frac{P}{2} - m_{kP})^2 + (1-\beta_2)\beta_2^{P-4}m_{kP+1}^2 + ... \\ &+ (1-\beta_2)\beta_2 m_{kP+P-4}^2 + (1-\beta_2)(m_{kP+P-3}+1)^2 \end{aligned} \tag{4.84}$$

$index = kP + P - 1,$

$$m_{kP+P-1} = m_{kP+P-1}\beta_1 \tag{4.85}$$

$$v_{kP+P-1} = v_{kP+P-2}\beta_2 \tag{4.86}$$

$$\begin{aligned} s_{kP+P-1} &= s_{kP}\beta_2^{P-1} + (1-\beta_2)\beta_2^{P-1}(\frac{P}{2} - m_{kP})^2 + (1-\beta_2)\beta_2^{P-3}m_{kP+1}^2 + ... \\ &+ (1-\beta_2)\beta_2^2 m_{kP+P-4}^2 + (1-\beta_2)\beta_2(m_{kP+P-3}+1)^2 + (1-\beta_2)m_{kP+P-2}^2 \end{aligned} \tag{4.87}$$

$index = kP + P,$

$$m_{kP+P} = m_{kP}\beta_1^P + (1-\beta_1)\frac{P}{2}\beta_1^{P-1} + (1-\beta_1)(-1)\beta_1^2 \tag{4.88}$$

$$v_{kP+P} = v_{kP}\beta_2^P + (1-\beta_2)\frac{P^2}{4}\beta_2^{P-1} + (1-\beta_2)\beta_2^2 \tag{4.89}$$

$$\begin{aligned} s_{kP+p} &= s_{kP}\beta_2^P + (1-\beta_2)\beta_2^{P-1}(\frac{P}{2} - m_{kP})^2 + (1-\beta_2)\beta_2^{P-2}m_{kP+1}^2 + ... \\ &+ (1-\beta_2)\beta_2^3 m_{kP+P-4}^2 + (1-\beta_2)\beta_2^2(m_{kP+P-3}+1)^2 \\ &+ (1-\beta_2)m_{kP+P-2}^2\beta_2 + (1-\beta_2)m_{kP+P-1}^2 \end{aligned} \tag{4.90}$$

Next, we derive the exact expression using the fact that the problem is periodic, hence $\lim_{k\to\infty} m_{kP} = \lim_{k\to\infty} m_{kP+P}, \lim_{k\to\infty} s_{kP} = \lim_{k\to\infty} s_{kP+P}, \lim_{k\to\infty} v_{kP} = \lim_{k\to\infty} v_{kP+P}$, hence we have:

$$\lim_{k\to\infty} m_{kP} = \lim_{k\to\infty} m_{kP}\beta_1^P + (1-\beta_1)\frac{P}{2}\beta_1^{P-1} + (1-\beta_1)(-1)\beta_1^2 \tag{4.91}$$

$$\lim_{k\to\infty} m_{kP} = \frac{1-\beta_1}{1-\beta_1^P}\Big[\frac{P}{2}\beta_1^{P-1} - \beta_1^2\Big] \tag{4.92}$$

$$\lim_{k\to\infty} m_{kP-1} = \frac{1}{\beta_1}\lim_{k\to\infty} m_{kP} \tag{4.93}$$

$$\lim_{k\to\infty} m_{kP-2} = \frac{1}{\beta_1}\Big[\lim_{k\to\infty} m_{kP-1} - (1-\beta_1)0\Big] \tag{4.94}$$

$$\lim_{k\to\infty} m_{kP-3} = \frac{1}{\beta_1}\Big[\lim_{k\to\infty} m_{kP-2} - (1-\beta_1)(-1)\Big] \tag{4.95}$$

Similarly, we can get

$$\lim_{k\to\infty} v_{kP} = \frac{1-\beta_2}{1-\beta_2^P}\Big[\frac{P^2}{4}\beta_2^{P-1} + \beta_2^2\Big] \tag{4.96}$$

$$\lim_{k\to\infty} v_{kP-1} = \frac{1}{\beta_2}\lim_{k\to\infty} v_{kP} \tag{4.97}$$

$$\lim_{k\to\infty} v_{kP-2} = \frac{1}{\beta_2}\lim_{k\to\infty} v_{kP-1} \tag{4.98}$$

$$\lim_{k\to\infty} v_{kP-3} = \frac{1}{\beta_2}\Big[\lim_{k\to\infty} v_{kP-2} - (1-\beta_2)\times 1^2\Big] \tag{4.99}$$

For ACProp, we have the following results:

$$\lim_{k\to\infty} s_{kP} = \lim_{k\to\infty} \frac{1-\beta_2}{1-\beta_2^P}\Big[\beta_2^{P-4}(\frac{P}{2} - m_{kP})^2 + \beta_2^3\frac{\beta_2^{P-5} - \beta_1^{2(P-4)}\beta_2}{1-\beta_1^2\beta_2} + \beta_2^2(m_{kP+P-3}+1)^2 + \beta_2 m_{kP+P-2}^2 + m_{kP+P-1}^2\Big] \tag{4.100}$$

$$\lim_{k\to\infty} s_{kP-1} = \lim_{k\to\infty} \frac{1}{\beta_2}\Big[s_{kP} - (1-\beta_2)m_{kP}^2\Big] \tag{4.101}$$

$$\lim_{k\to\infty} s_{kP-2} = \lim_{k\to\infty} \frac{1}{\beta_2}\Big[s_{kP-1} - (1-\beta_2)m_{kP-1}^2\Big] \tag{4.102}$$

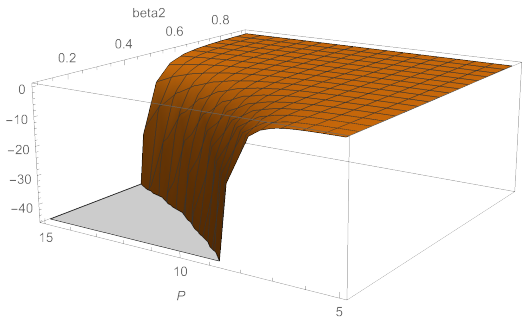


Figure 4.15: Value of $\frac{s^+}{s^-} - \frac{v^+}{v^-}$ when $\beta_1 = 0.2$

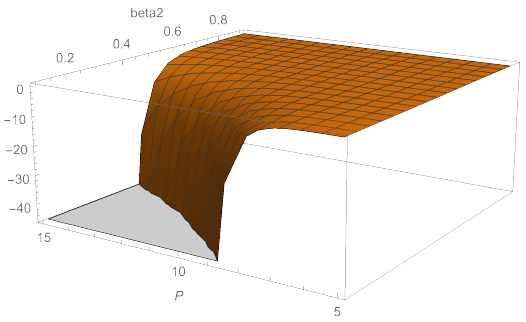


Figure 4.16: Value of $\frac{s^+}{s^-} - \frac{v^+}{v^-}$ when $\beta_1 = 0.9$

$$\lim_{k\to\infty} s_{kP-3} = \lim_{k\to\infty} \frac{1}{\beta_2}\Big[s_{kP-2} - (1-\beta_2)(m_{kP-2}+1)^2\Big] \tag{4.103}$$

$$\tag{4.104}$$

Within each period, ACprop will perform a positive update $P/(2\sqrt{s^+})$ and a negative update $-1/\sqrt{s^-}$, where s^+ (s^-) is the value of denominator before observing positive (negative) gradient. Similar notations for v^+ and v^- in AdaShift, where $s^+ = s_{kP}, s^- = s_{kP-3}, v^+ = v_{kP}, v^- = v_{kP-3}$. A net update in the correct direction requires $\frac{P}{2\sqrt{s^+}} > \frac{1}{\sqrt{s^-}}$, (or $s^+/s^- < P^2/4$). Since we have the exact expression for these terms in the limit sense, it's trivial to verify that $s^+/s^- \leq v^+/v^-$ (e.g. the value $\frac{s^+}{s^-} - \frac{v^+}{v^-}$ is negative as in Fig. 4.15 and 4.16), hence ACProp is easier to satisfy the convergence condition. □

4.5.4 Numerical validations

We conducted more experiments to validate previous claims. We plot the area of convergence for different β_1 values for problem (1) in Fig. 4.17 to Fig. 4.19, and validate the always-convergence property of ACProp with different values of β_1. We also plot the area of convergence for problem (2) defined by Eq. equation 4.78, results are shown in Fig. 4.20 to Fig. 4.22. Note that for this problem the always-convergence does not hold, but ACProp has a much larger area of convergence than AdaShift.

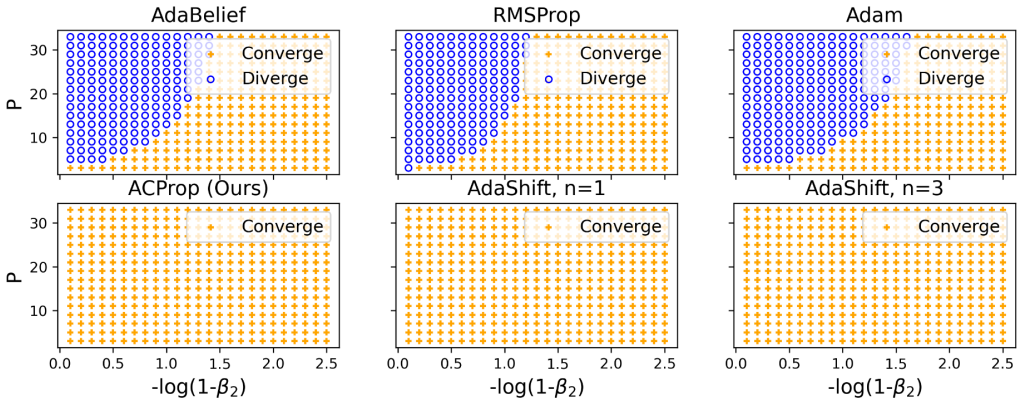


Figure 4.17: Numerical experiments on problem (1) with $\beta_1 = 0.5$

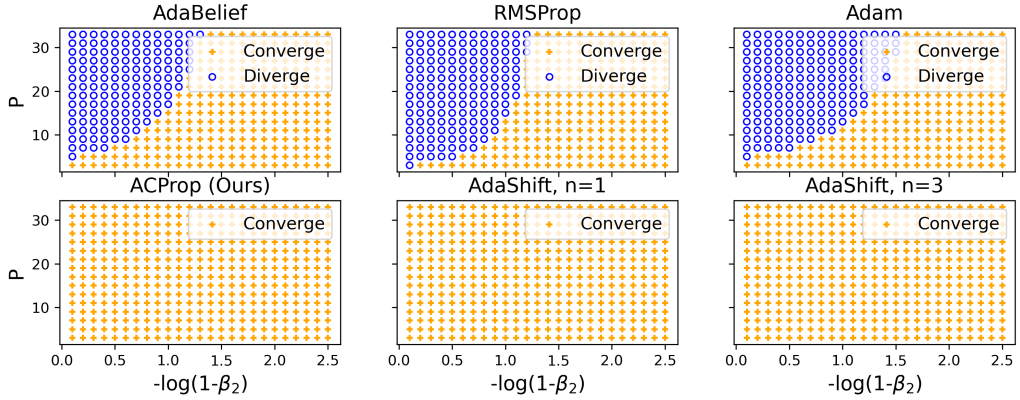


Figure 4.18: Numerical experiments on problem (1) with $\beta_1 = 0.5$

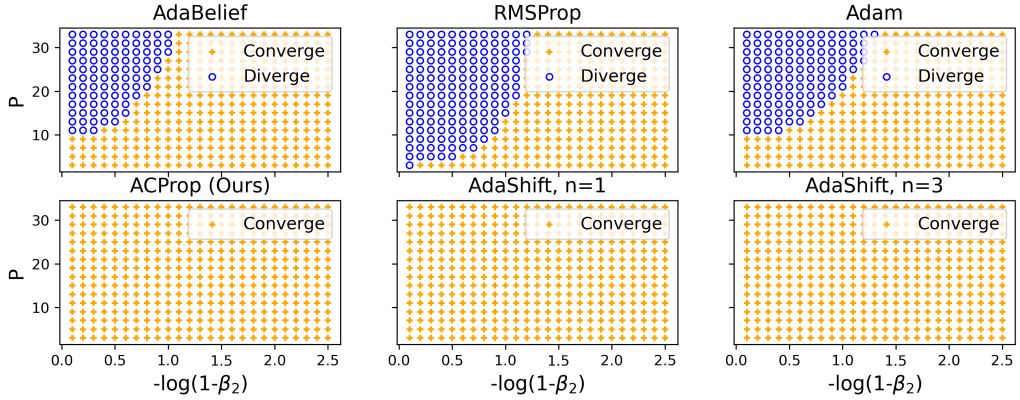


Figure 4.19: Numerical experiments on problem (1) with $\beta_1 = 0.9$

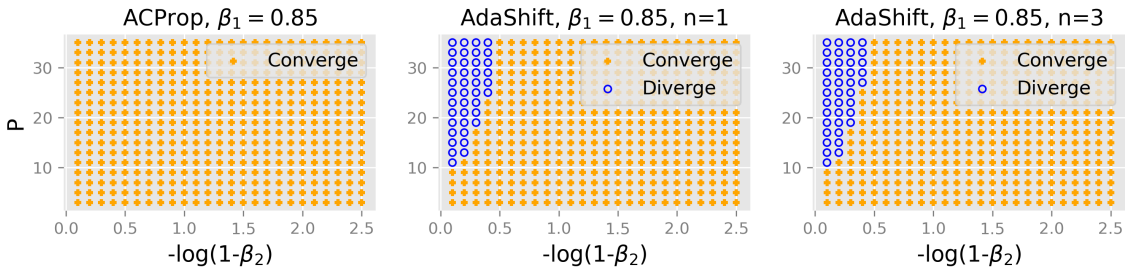


Figure 4.20: Numerical experiments on problem (43) with $\beta_1 = 0.85$

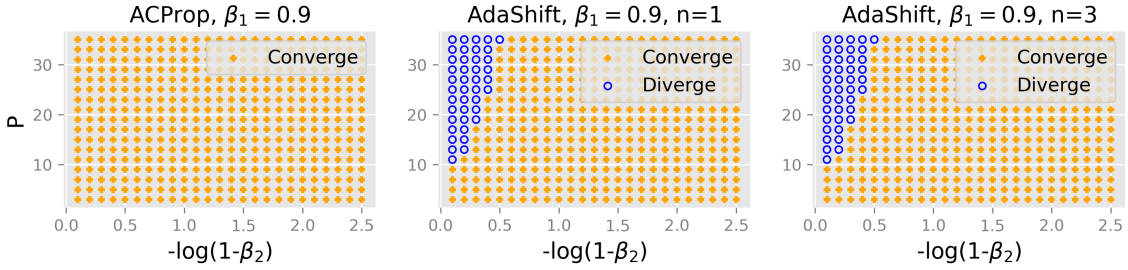


Figure 4.21: Numerical experiments on problem (43) with $\beta_1 = 0.9$

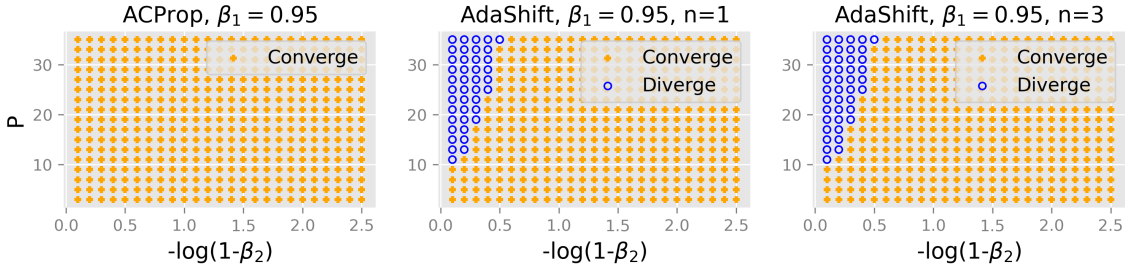
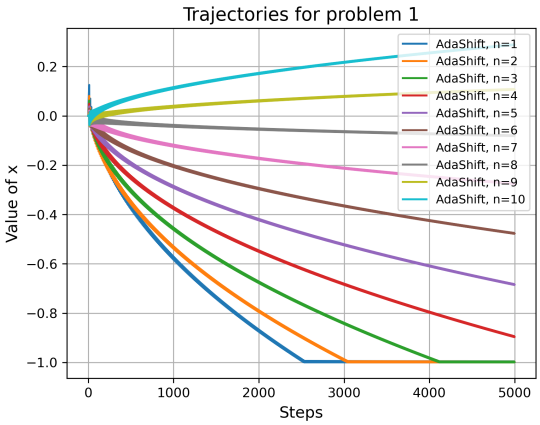
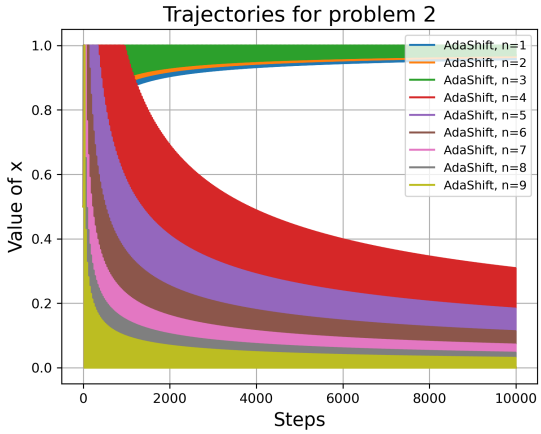


Figure 4.22: Numerical experiments on problem (43) with $\beta_1 = 0.95$



(a) Trajectories of AdaShift with various n for problem (1). Note that optimal is $x^* = -1$. Note that convergence of problem (1) requires a small delay step n, but convergence of problem (2) requires a large n, hence there's no good criterion to select an optimal n.



(b) Trajectories of AdaShift with various n for problem (43). Note that optimal is $x^* = 0.0$, and the trajectories are oscillating at a high frequency hence appears to be spanning an area.

4.5.5 Asynchronous AdaBelief matches the oracle convergence rate for stochastic non-convex optimization

Problem definition and assumptions

The problem is defined as:

$$\min_{x\in\mathbb{R}^d} f(x) = \mathbb{E}[F(x,\xi)] \tag{4.105}$$

where x typically represents parameters of the model, and ξ represents data which typically follows some distribution.

We mainly consider the stochastic non-convex case, with assumptions below.

A.1 f is continuously differentiable, f is lower-bounde by f^*. $\nabla f(f)$ is globalluy Lipschitz continuous with constant L:

$$||\nabla f(x) - \nabla f(y)|| \leq L||x-y|| \tag{4.106}$$

A.2 For any iteration t, g_t is an unbiased estimator of $\nabla f(x_t)$ with variance bounded by σ^2. The norm of g_t is upper-bounded by M_g.

$$(a) \quad \mathbb{E}g_t = \nabla f(x_t) \tag{4.107}$$

$$(b) \quad \mathbb{E}\big[||g_t - \nabla f(x_t)||^2\big] \leq \sigma^2 \tag{4.108}$$

Convergence analysis of Async-optimizers in stochastic non-convex optimization

Theorem 4.5.4 (Thm.4.1 in the main paper)**.** *Under assumptions **A.1-2**, assume f is upper bounded by M_f, with learning rate schedule as*

$$\alpha_t = \alpha_0 t^{-\eta}, \;\; \alpha_0 \leq \frac{C_l}{LC_u^2}, \;\; \eta \in [0.5, 1) \tag{4.109}$$

the sequence generated by

$$x_{t+1} = x_t - \alpha_t A_t g_t \tag{4.110}$$

satisfies

$$\frac{1}{T}\sum_{t=1}^{T}\Big|\Big|\nabla f(x_t)\Big|\Big|^2 \leq \frac{2}{C_l}\Big[(M_f - f^*)\alpha_0 T^{\eta-1} + \frac{LC_u^2\sigma^2\alpha_0}{2(1-\eta)}T^{-\eta}\Big] \tag{4.111}$$

where C_l and C_u are scalars representing the lower and upper bound for A_t, e.g. $C_l I \preceq A_t \preceq C_u I$, where $A \preceq B$ represents $B - A$ is semi-positive-definite.

Proof. Let

$$\delta_t = g_t - \nabla f(x_t) \tag{4.112}$$

then by **A.2**, $\mathbb{E}\delta_t = 0$.

$$f(x_{t+1}) \leq f(x_t) + \Big\langle \nabla f(x_t), x_{t+1} - x_t \Big\rangle + \frac{L}{2}\Big|\Big|x_{t+1} - x_t\Big|\Big|^2 \tag{4.113}$$

$$\Big(\textit{by L-smoothness of } f(x)\Big)$$

$$= f(x_t) - \alpha_t\Big\langle \nabla f(x_t), A_t g_t \Big\rangle + \frac{L}{2}\alpha_t^2\Big|\Big|A_t g_t\Big|\Big|^2 \tag{4.114}$$

$$= f(x_t) - \alpha_t\Big\langle \nabla f(x_t), A_t\big(\delta_t + \nabla f(x_t)\big) \Big\rangle + \frac{L}{2}\alpha_t^2\Big|\Big|A_t g_t\Big|\Big|^2 \tag{4.115}$$

$$\leq f(x_t) - \alpha_t\Big\langle \nabla f(x_t), A_t\nabla f(x_t) \Big\rangle - \alpha_t\Big\langle \nabla f(x_t), A_t\delta_t \Big\rangle + \frac{L}{2}\alpha_t^2 C_u^2\Big|\Big|g_t\Big|\Big|^2 \tag{4.116}$$

Take expectation on both sides of Eq. equation 4.116, conditioned on $\xi_{[t-1]} = \{x_1, x_2, ...x_{t-1}\}$, also notice that A_t is a constant given $\xi_{[t-1]}$, we have

$$\mathbb{E}\big[f(x_{t+1})|x_1, ...x_t\big] \leq f(x_t) - \alpha_t\Big\langle \nabla f(x_t), A_t \nabla f(x_t)\Big\rangle + \frac{L}{2}\alpha_t^2 C_u^2 \mathbb{E}\Big|\Big|g_t\Big|\Big|^2 \tag{4.117}$$
$$\Big(A_t \textit{ is independent of } g_t \textit{ given } \{x_1, ...x_{t-1}\},\ \textit{and } \mathbb{E}\delta_t = 0\Big)$$

In order to bound RHS of Eq. equation 4.117, we first bound $\mathbb{E}\big[||g_t||^2\big]$.

$$\mathbb{E}\Big[\Big|\Big|g_t\Big|\Big|^2\Big|x_1, ...x_t\Big] = \mathbb{E}\Big[\Big|\Big|\nabla f(x_t) + \delta_t\Big|\Big|^2\Big|x_1, ...x_t\Big] \tag{4.118}$$
$$= \mathbb{E}\Big[\Big|\Big|\nabla f(x_t)\Big|\Big|^2\Big|x_1, ...x_t\Big] + \mathbb{E}\Big[\Big|\Big|\nabla \delta_t\Big|\Big|^2\Big|x_1, ...x_t\Big] + 2\mathbb{E}\Big[\Big\langle \delta_t, \nabla f(x_t)\Big\rangle\Big|x_1, ...x_t\Big] \tag{4.119}$$
$$\leq \Big|\Big|\nabla f(x_t)\Big|\Big|^2 + \sigma^2 \tag{4.120}$$
$$\Big(\textit{By } \textbf{A.2}\textit{, and } \nabla f(x_t) \textit{ is a constant given } x_t\Big)$$

Plug Eq. equation 4.120 into Eq. equation 4.117, we have

$$\mathbb{E}\Big[f(x_{t+1})\Big|x_1, ...x_t\Big] \leq f(x_t) - \alpha_t\Big\langle \nabla f(x_t), A_t \nabla f(x_t)\Big\rangle + \frac{L}{2}C_u^2\alpha_t^2\Big[\Big|\Big|\nabla f(x_t)\Big|\Big|^2 + \sigma^2\Big] \tag{4.121}$$
$$= f(x_t) - \Big(\alpha_t C_l - \frac{LC_u^2}{2}\alpha_t^2\Big)\Big|\Big|\nabla f(x_t)\Big|\Big|^2 + \frac{LC_u^2\sigma^2}{2}\alpha_t^2 \tag{4.122}$$

By **A.5** that $0 < \alpha_t \leq \frac{C_l}{LC_u^2}$, we have

$$\alpha_t C_l - \frac{LC_u^2\alpha_t^2}{2} = \alpha_t\Big(C_l - \frac{LC_u^2\alpha_t}{2}\Big) \geq \alpha_t\frac{C_l}{2} \tag{4.123}$$

Combine Eq. equation 4.122 and Eq. equation 4.123, we have

$$\frac{\alpha_t C_l}{2}\Big|\Big|\nabla f(x_t)\Big|\Big|^2 \leq \Big(\alpha_t C_l - \frac{LC_u^2\alpha_t^2}{2}\Big)\Big|\Big|\nabla f(x_t)||^2 \tag{4.124}$$

$$\leq f(x_t) - \mathbb{E}\Big[f(x_{t+1})\Big|x_1, ...x_t\Big] + \frac{LC_u^2\sigma^2}{2}\alpha_t^2 \tag{4.125}$$

Then we have

$$\frac{C_l}{2}\Big|\Big|\nabla f(x_t)\Big|\Big|^2 \leq \frac{1}{\alpha_t}f(x_t) - \frac{1}{\alpha_t}\mathbb{E}\Big[f(x_{t+1})\Big|x_1, ...x_t\Big] + \frac{LC_u^2\sigma^2}{2}\alpha_t \tag{4.126}$$

Perform telescope sum on Eq. equation 4.126, and recursively taking conditional expectations on the history of $\{x_i\}_{i=1}^T$, we have

$$\frac{C_l}{2}\sum_{t=1}^{T}||\nabla f(x_t)\Big|\Big|^2 \leq \sum_{t=1}^{T}\frac{1}{\alpha_t}\Big(\mathbb{E}f(x_t) - \mathbb{E}f(x_{t+1})\Big) + \frac{LC_u^2\sigma^2}{2}\sum_{t=1}^{T}\alpha_t \tag{4.127}$$

$$= \frac{\mathbb{E}f(x_1)}{\alpha_1} - \frac{\mathbb{E}f(x_{T+1})}{\alpha_T} + \sum_{t=2}^{T}\Big(\frac{1}{\alpha_t} - \frac{1}{\alpha_{t-1}}\Big)\mathbb{E}f(x_t) + \frac{LC_u^2\sigma^2}{2}\sum_{t=1}^{T}\alpha_t \tag{4.128}$$

$$\leq \frac{M_f}{\alpha_1} - \frac{f^*}{\alpha_T} + M_f\sum_{t=1}^{T}\Big(\frac{1}{\alpha_t} - \frac{1}{\alpha_{t-1}}\Big) + \frac{LC_u^2\sigma^2}{2}\sum_{t=1}^{T}\alpha_t \tag{4.129}$$

$$\leq \frac{M_f - f^*}{\alpha_T} + \frac{LC_u^2\sigma^2}{2}\sum_{t=1}^{T}\alpha_t \tag{4.130}$$

$$\leq (M_f - f^*)\alpha_0 T^\eta + \frac{LC_u^2\sigma^2\alpha_0}{2}\Big(\zeta(\eta) + \frac{T^{1-\eta}}{1-\eta} + \frac{1}{2}T^{-\eta}\Big) \tag{4.131}$$

$\Big($*By sum of generalized harmonic series*,

$$\sum_{k=1}^{n}\frac{1}{k^s} \sim \zeta(s) + \frac{n^{1-s}}{1-s} + \frac{1}{2n^s} + O(n^{-s-1}), \tag{4.132}$$

$\zeta(s)$ *is Riemann zeta function.*$\Big)$

Then we have

$$\frac{1}{T}\sum_{t=1}^{T}\left|\left|\nabla f(x_t)\right|\right|^2 \leq \frac{2}{C_l}\left[(M_f - f^*)\alpha_0 T^{\eta-1} + \frac{LC_u^2\sigma^2\alpha_0}{2(1-\eta)}T^{-\eta}\right] \tag{4.133}$$

□

Validation on numerical accuracy of sum of generalized harmonic series

We performed experiments to test the accuracy of the analytical expression of sum of harmonic series. We numerically calculate $\sum_{i=1}^{N}\frac{1}{i^\eta}$ for η varying from 0.5 to 0.999, and for N ranging from 10^3 to 10^7 in the log-grid. We calculate the error of the analytical expression by Eq. equation 4.132, and plot the error in Fig. 4.24. Note that the y-axis has a unit of 10^{-7}, while the sum is typically on the order of 10^3, this implies that expression Eq. equation 4.132 is very accurate and the relative error is on the order of 10^{-10}. Furthermore, note that this expression is accurate even when $\eta = 0.5$.

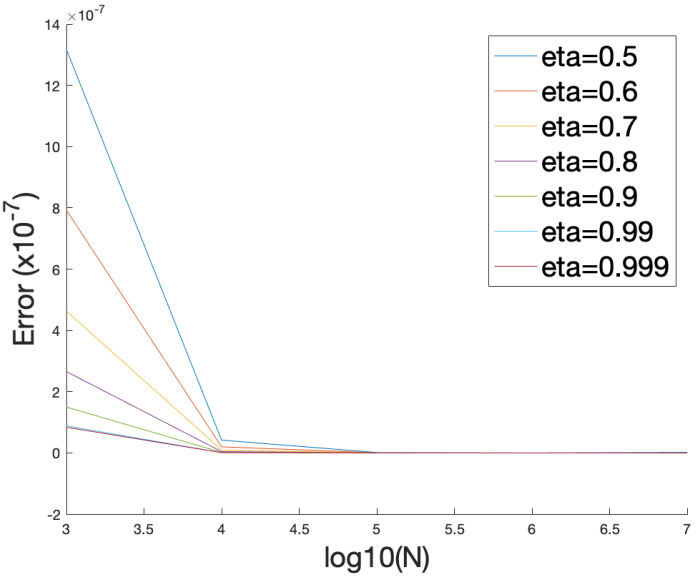


Figure 4.24: The error between numerical sum for $\sum_{i=1}^{N}\frac{1}{i^\eta}$ and the analytical form.

Chapter 5

Surrogate Gap Guided Sharpness-Aware Minimization (GSAM) improves generalization

5.1 Introduction

Modern neural networks are typically highly over-parameterized and easy to overfit to training data, yet the generalization performances on unseen data (test set) often suffer from a gap from the training performance [171]. Many studies try to understand the generalization of machine learning models, including the Bayesian perspective [97, 110], the information perspective [84], the loss surface geometry perspective [60, 66] and the kernel perspective [64, 163]. Besides analyzing the properties of a model after training, some works study the influence of training and the optimization process, such as implicit regularization of stochastic gradient descent (SGD) [14, 177], the learning rate's regularization effect [83], and the influence of the batch size [70].

These studies have led to various modifications to the training process to improve generalization. [69] proposed to use Adam in early training phases for fast convergence, then switch to SGD in late phases for better generalization. [63] proposed to average

weights to achieve a wider local minimum, which is expected to generalize better than sharp minima. A similar idea was later used in Lookahead [174]. Entropy-SGD [19] derived the gradient of local entropy to avoid solutions in sharp valleys. Entropy-SGD had a nested Langevin iteration, inducing much higher computation costs than vanilla training.

The recently proposed Sharpness-Aware Minimization (SAM) [42] is a generic training scheme that improves generalization and has been shown especially effective for Vision Transformers [36] when large-scale pre-training is unavailable [23]. Suppose vanilla training minimizes loss $f(w)$ (e.g., the cross-entropy loss for classification), where w is the parameter. SAM minimizes *perturbed loss* defined as $f_p(w) \triangleq \max_{||\delta|| \leq \rho} f(w+\delta)$, which is the *maximum* loss within radius ρ centered at model parameter w. Intuitively, vanilla training seeks a single point with a low loss, while SAM searches for a neighborhood within which the maximum loss is low. However, we show that a low perturbed loss f_p could appear in both flat and sharp minima, implying that only minimizing f_p is not always sharpness-aware.

Although the perturbed loss $f_p(w)$ might disagree with sharpness, we find *surrogate gap* defined as $h(w) \triangleq f_p(w) - f(w)$ agrees with sharpness — Lemma 5.3.0.3 shows that the surrogate gap h is an equivalent measure of the dominant eigenvalue of Hessian at a local minimum. Inspired by this observation, we propose the Surrogate **G**ap Guided **S**harpness **A**ware **M**inimization (GSAM) which jointly minimizes the perturbed loss f_p and the surrogate gap h: a low perturbed loss f_p indicates a low training loss within the neighborhood, and a small surrogate gap h avoids solutions in sharp valleys and hence narrows the generalization gap between training and test performances (Thm. 5.5.3). When both criteria are satisfied, we find a generalizable model with good performance.

GSAM consists of two steps for each update: 1) descend gradient $\nabla f_p(w)$ to minimize the perturbed loss f_p (this step is exactly the same as SAM), and 2) decompose gradient $\nabla f(w)$ of the original loss $f(w)$ into components that are parallel and orthogonal to

$\nabla f_p(w)$, i.e., $\nabla f(w) = \nabla_{\parallel} f(w) + \nabla_{\perp} f(w)$, and perform an ascent step in $\nabla_{\perp} f(w)$ to minimize the surrogate gap $h(w)$. Note that this ascent step does not change the perturbed loss f_p because $\nabla f_{\perp}(w) \perp \nabla f_p(w)$ by construction.

We summarize our contribution as follows:

- We define *surrogate gap*, which measures the sharpness at local minima and is easy to compute.
- We propose the GSAM method to improve generalization of neural networks. GSAM is widely applicable and incurs negligible computation overhead compared to SAM.
- We demonstrate the convergence of GSAM and its provably better generalization than SAM.
- We empirically validate GSAM over image classification tasks with various neural architectures, including ResNets [53], Vision Transformers [36], and MLP-Mixers [153].

5.2 Preliminaries

5.2.1 Notations

- $f(w)$: A loss function f with parameter $w \in \mathbb{R}^k$, where k is the dimension of parameters.
- $\rho_t \in \mathbb{R}$: A scalar value controlling the amplitude of perturbation at step t.
- $\epsilon \in \mathbb{R}$: A small positive constant (to avoid division by 0, $\epsilon = 10^{-12}$ by default).
- $w_t^{adv} \triangleq w_t + \rho_t \frac{\nabla f(w_t)}{||\nabla f(w_t)|| + \epsilon}$: The solution to $\max_{||w' - w_t|| \leq \rho_t} f(w')$ when ρ_t is small.

- $f_p(w_t) \triangleq \max_{||\delta|| \leq \rho_t} f(w_t + \delta) \approx f(w_t^{adv})$: The perturbed loss induced by $f(w_t)$. For each w_t, $f_p(w_t)$ returns the worst possible loss f within a ball of radius ρ_t centered at w_t. When ρ_t is small, by Taylor expansion, the solution to the maximization problem is equivalent to a gradient ascent from w_t to w_t^{adv}.

- $h(w) \triangleq f_p(w) - f(w)$: The surrogate gap defined as the difference between $f_p(w)$ and $f(w)$.

- $\eta_t \in \mathbb{R}$: Learning rate at step t.

- $\alpha \in \mathbb{R}$: A constant value that controls the scaled learning rate of the ascent step in GSAM.

- $g^{(t)}, g_p^{(t)} \in \mathbb{R}^k$: At the t-th step, the noisy observation of the gradients $\nabla f(w_t)$, $\nabla f_p(w_t)$ of the original loss and perturbed loss, respectively.

- $\nabla f(w_t) = \nabla f_{\parallel}(w_t) + \nabla f_{\perp}(w_t)$: Decompose $\nabla f(w_t)$ into parallel component $\nabla f_{\parallel}(w_t)$ and vertical component $\nabla f_{\perp}(w_t)$ by projection $\nabla f(w_t)$ onto $\nabla f_p(w_t)$.

5.2.2 Sharpness-Aware Minimization

Conventional optimization of neural networks typically minimizes the training loss $f(w)$ by gradient descent w.r.t. $\nabla f(w)$ and searches for a single point w with a low loss. However, this vanilla training often falls into a sharp valley of the loss surface, resulting in inferior generalization performance [19]. Instead of searching for a single point solution, SAM seeks a region with low losses so that small perturbation to the model weights does not cause significant performance degradation. SAM formulates the problem as:

$$\min_w f_p(w) \text{ where } f_p(w) \triangleq \max_{||\delta|| \leq \rho} f(w + \delta) \tag{5.1}$$

where ρ is a predefined constant controlling the radius of a neighborhood. This perturbed loss f_p induced by $f(w)$ is the maximum loss within the neighborhood. When the perturbed loss is minimized, the neighborhood corresponds to low losses (below the perturbed loss). For a small ρ, using Taylor expansion around w, the inner maximization in Eq. 5.1 turns into a linear constrained optimization with solution

$$\arg\max_{||\delta||\leq\rho} f(w+\delta) = \arg\max_{||\delta||\leq\rho} f(w) + \delta^\top \nabla f(w) + O(\rho^2) = \rho \frac{\nabla f(w)}{||\nabla f(w)||} \quad (5.2)$$

As a result, the optimization problem of SAM reduces to

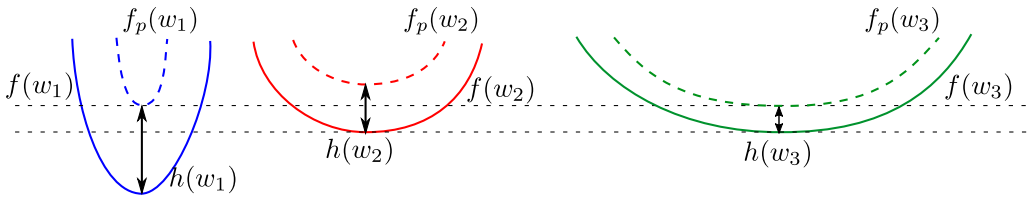
$$\min_w f_p(w) \approx \min_w f(w^{adv}) \text{ where } w^{adv} \triangleq w + \rho \frac{\nabla f(w)}{||\nabla f(w)|| + \epsilon} \quad (5.3)$$

where ϵ is a scalar (default: 1e-12) to avoid division by 0, and w^{adv} is the "perturbed weight" with the highest loss within the neighborhood. Equivalently, SAM seeks a solution on the surface of the perturbed loss $f_p(w)$ rather than the original loss $f(w)$ [42].

5.3 The surrogate gap measures the sharpness at a local minimum

5.3.1 The perturbed loss is not always sharpness-aware

Despite that SAM searches for a region of low losses, we show that a solution by SAM is not guaranteed to be flat. Throughout this paper we measure the sharpness at a local minimum of loss $f(w)$ by the dominant eigenvalue σ_{max} (eigenvalue with the largest absolute value) of Hessian. For simplicity, we do not consider the influence of reparameterization on the geometry of loss surfaces, which is thoroughly discussed in [79, 78].



$sharpness(w_1) > sharpness(w_2) > sharpness(w_3)$, $h(w_1) > h(w_2) > h(w_3)$,
$f_p(w_1) = f_p(w_3) < f_p(w_2)$, hence h agrees with sharpness while f_p might not.

Figure 5.1: Consider original loss f (solid line), perturbed loss $f_p \triangleq \max_{||\delta|| \leq \rho} f(w+\delta)$ (dashed line), and surrogate gap $h(w) \triangleq f_p(w) - f(w)$. Intuitively, f_p is approximately a max-pooled version of f with a pooling kernel of width 2ρ, and SAM minimizes f_p. From left to right are the local minima centered at w_1, w_2, w_3, and the valleys become flatter. Since $f_p(w_1) = f_p(w_3) < f_p(w_2)$, SAM prefers w_1 and w_3 to w_2. *However, a low f_p could appear in both sharp (w_1) and flat (w_3) minima, so f_p might disagree with sharpness*. On the contrary, a smaller surrogate gap h indicates a flatter loss surface (Lemma 5.3.0.3). From w_1 to w_3, the loss surface is flatter, and h is smaller.

Lemma 5.3.0.1. *For some fixed ρ, consider two local minima w_1 and w_2, $f_p(w_1) \leq f_p(w_2) \not\Longrightarrow \sigma_{max}(w_1) \leq \sigma_{max}(w_2)$, where σ_{max} is the dominant eigenvalue of the Hessian.*

We leave the proof to Appendix. Fig. 5.1 illustrates Lemma 5.3.0.1 with an example. Consider three local minima denoted as w_1 to w_3, and suppose the corresponding loss surfaces are flatter from w_1 to w_3. For some fixed ρ, we plot the perturbed loss f_p and surrogate gap h around each solution. Comparing w_2 with w_3: Suppose their vanilla losses are equal, $f(w_2) = f(w_3)$, then $f_p(w_2) > f_p(w_3)$ because the loss surface is flatter around w_3, implying that SAM will prefer w_3 to w_2. Comparing w_1 and w_2: $f_p(w_1) < f_p(w_2)$, and SAM will favor w_1 over w_2 because it only cares about the perturbed loss f_p, even though the loss surface is sharper around w_1 than w_2.

5.3.2 The surrogate gap agrees with sharpness

We introduce the surrogate gap that agrees with sharpness, defined as:

$$h(w) \triangleq f(w^{adv}) - f(w) \tag{5.4}$$

Intuitively, the surrogate gap represents the difference between the maximum loss within the neighborhood and the loss at the center point. The surrogate gap has the following properties.

Lemma 5.3.0.2. *Suppose the perturbation amplitude ρ is sufficiently small, then the approximation to the surrogate gap in Eq. 5.4 is always non-negative, $h(w) = f(w^{adv}) - f(w) \geq 0, \forall w$.*

Lemma 5.3.0.3. *For a local minimum w^*, consider the dominate eigenvalue σ_{max} of the Hessian of loss f as a measure of sharpness. Considering the neighborhood centered at w^* with a small radius ρ, the surrogate gap $h(w^*)$ is an equivalent measure of the sharpness: $\sigma_{max} \approx 2h(w^*)/\rho^2$.*

The proof is in Appendix. Lemma 5.3.0.2 tells that the surrogate gap is non-negative, and Lemma 5.3.0.3 shows that the loss surface is flatter as h gets closer to 0. The two lemmas together indicate that we can find a region with a flat loss surface by minimizing the surrogate gap $h(w)$.

5.4 Surrogate Gap Guided Sharpness-Aware Minimization

5.4.1 General idea: Jointly minimize the perturbed loss and surrogate gap

Inspired by the analysis in Section 5.3, we propose Surrogate **G**ap Guided **S**harpness-**A**ware **M**inimzation (GSAM) to minimize both the perturbed loss f_p and surrogate gap h:

$$\min_w \big(f_p(w), h(w)\big). \tag{5.5}$$

Intuitively, by minimizng f_p we search for a region with a low perturbed loss similar to SAM, and by minimizing h we search for a local minimum with a flat surface. A low perturbed loss implies low training losses within the neighborhood, and a flat loss surface reduces the generalization gap between training and test performances [19]. When both are minimized, the solution gives rise to high accuracy and good generalization.

Potential caveat in optimization It is tempting and yet sub-optimal to combine the objectives in Eq. 5.5 to arrive at $\min_w f_p(w) + \lambda h(w)$, where λ is some positive scalar. One caveat when solving this weighted combination is the potential conflict between the gradients of the two terms, i.e., $\nabla f_p(w)$ and $\nabla h(w)$. We illustrate this conflict by Fig. 5.2, where $\nabla h(w) = \nabla f_p(w) - \nabla f(w)$ (the grey dashed arrow) has a negative inner product with $\nabla f_p(w)$ and $\nabla f(w)$. Hence, the gradient descent for the surrogate gap could potentially increase the loss f_p, harming the model's performance. We empirically validate this argument in Sec. 5.6.4.

5.4.2 Gradient decomposition and ascent for the multi-objective optimization

Our primary goal is to minimize f_p because otherwise a flat solution of high loss is meaningless, and the minimization of h should not increase f_p. We propose to decompose $\nabla f(w_t)$ into components that are parallel and orthogonal to $\nabla f_p(w_t)$, respectively (see the black and blue arrows in Fig. 5.2):

$$\nabla f(w_t) = \nabla f_{\parallel}(w_t) + \nabla f_{\perp}(w_t). \tag{5.6}$$

The key is that updating in the direction of $\nabla f_{\perp}(w_t)$ does *not* change the value of the perturbed loss $f_p(w_t)$ because $\nabla f_{\perp} \perp \nabla f_p$ by construction. Therefore, we propose to perform an **ascent step in the $\nabla f_{\perp}(w_t)$ direction**, which achieves two goals simultaneously — it keeps the value of $f_p(w_t)$ intact and meanwhile decreases the surrogate gap $h(w_t) = f_p(w_t) - f(w_t)$ (by increasing $f(w_t)$).

The full GSAM Algorithm is shown in Algo. 16 and Fig. 5.2, where $g^{(t)}, g_p^{(t)}$ are noisy observations of $\nabla f(w_t)$ and $\nabla f_p(w_t)$, respectively, and $g_{\parallel}^{(t)}, g_{\perp}^{(t)}$ are noisy observations of $\nabla f_{\parallel}(w_t)$ and $\nabla f_{\perp}(w_t)$, respectively, by projecting $g^{(t)}$ onto $g_p^{(t)}$. We introduce a constant α to scale the stepsize of the ascent step. Steps 1) to 2) are the same as SAM: At current point w_t, step 1) takes a gradient ascent to w_t^{adv} followed by step 2) evaluating the gradient $g_p^{(t)}$ at w_t^{adv}. Step 3) projects $g^{(t)}$ onto $g_p^{(t)}$, which requires negligible computation compared to the forward and backward pass. In step 4), $-\eta_t g_p^{(t)}$ is the same as in SAM and minimizes the perturbed loss $f_p(w_t)$ with gradient descent, and $\alpha\eta_t g_{\perp}^{(t)}$ performs an *ascent* step in the orthogonal direction of $g_p^{(t)}$ to minimize the surrogate gap $h(w_t)$ (equivalently increase $f(w_t)$ and keep $f_p(w_t)$ intact). In coding, GSAM feeds the "surrogate gradient" $\nabla f_t^{GSAM} \triangleq g_p^{(t)} - \alpha g_{\perp}^{(t)}$ to first-order gradient optimizers such as SGD and Adam.

The ascent step along $g_\perp^{(t)}$ does not harm convergence SAM demonstrates that minimizing f_p successfully trains the network and generalizes better than minimizing f. Even though our ascent step along $g_\perp^{(t)}$ increases $f(w)$, it does not affect $f_p(w)$, so GSAM still decreases the perturbed loss f_p in a way similar to SAM. In Thm. 5.5.1, we formally prove the convergence of GSAM. In Sec. 5.6 and Appendix C, we empirically validate that the loss decreases and accuracy increases with training.

Illustration with a toy example We demonstrate different algorithms by a numerical toy example shown in Fig. 5.3. The trajectory of GSAM is closer to the ridge and tends to find a flat minimum. Intuitively, since the loss surface is smoother along the ridge than in sharp local minima, the surrogate gap $h(w)$ is small near the ridge, and the ascent step in GSAM minimizes h to pushes the trajectory closer to the ridge. More concretely, $\nabla f(w_t)$ points to a sharp local solution and deviates from the ridge; in contrast, w_t^{adv} is closer to the ridge and $\nabla f(w_t^{adv})$ is closer to the ridge descent direction than $\nabla f(w_t)$. Note that ∇f_t^{GSAM} and $\nabla f(w_t)$ always lie at different sides of $\nabla f_p(w_t)$ by construction (see Fig. 5.2), hence ∇f_t^{GSAM} pushes the trajectory closer to the ridge than $\nabla f_p(w_t)$ does. The trajectory of GSAM is like descent along the ridge and tends to find flat minima.

5.5 Theoretical properties of GSAM

5.5.1 Convergence during training

Theorem 5.5.1. *Consider a non-convex function $f(w)$ with Lipschitz-smooth constant L and lower bound f_{min}. Suppose we can access a noisy, bounded observation $g^{(t)}$ ($||g^{(t)}||_2 \leq G, \forall t$) of the true gradient $\nabla f(w_t)$ at the t-th step. For some constant α, with learning rate $\eta_t = \eta_0/\sqrt{t}$, and perturbation amplitude ρ_t proportional to the learning rate,*

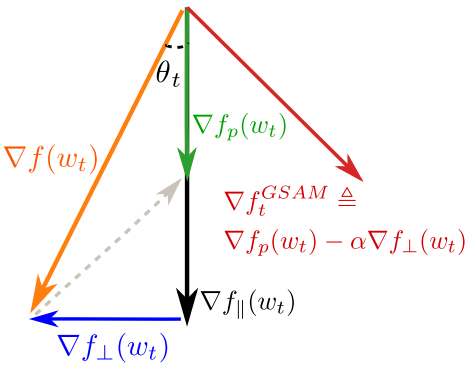


Figure 5.2: Illustration of GSAM.

Algorithm 16: GSAM Algorithm

For $t = 1$ to T

0) $\rho_t = \rho_{min} + (\rho_{max} - \rho_{min}) \frac{lr - lr_{min}}{lr_{max} - lr_{min}}$

1) $w_t^{adv} = w_t + \rho_t \frac{g^{(t)}}{||g^{(t)}|| + \epsilon}$, $g^{(t)}$ is a noisy observation of $\nabla f(w_t)$.

2) Get $g_p^{(t)}$ (a noisy observation of $\nabla f_p(w_t)$) by back-propagation at w_t^{adv}.

3) $g^{(t)} = g_\parallel^{(t)} + g_\perp^{(t)}$ Decompose $g^{(t)}$ into components that are parallel and orthogonal to $g_p^{(t)}$.

4) Update weights:

Vanilla $w_{t+1} = w_t - \eta_t g^{(t)}$

SAM $w_{t+1} = w_t - \eta_t g_p^{(t)}$

GSAM

$w_{t+1} = w_t - \eta_t g_p^{(t)} + \alpha \eta_t g_\perp^{(t)}$

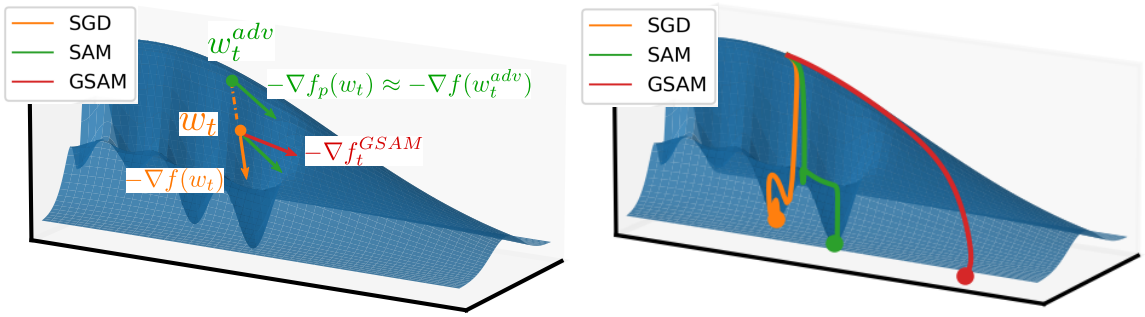


Figure 5.3: Consider the loss surface with a few sharp local minima. **Left**: Overview of the procedures of SGD, SAM and GSAM. SGD takes a descent step at w_t using $\nabla f(w_t)$ (orange), which points to a sharp local minima. SAM first performs gradient ascent in the direction of $\nabla f(w_t)$ to reach w_t^{adv} with a higher loss, followed by descent with gradient $\nabla f(w_t^{adv})$ (green) at the perturbed weight. Based on $\nabla f(w_t)$ and $\nabla f(w_t^{adv})$, GSAM updates in a new direction (red) that points to a flatter region. **Right**: Trajectories by different methods. SGD and SAM fall into different sharp local minima, while GSAM reaches a flat region. A video is in the supplement for better visualization.

e.g., $\rho_t = \rho_0/\sqrt{t}$*, we have*

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\Big|\Big|\nabla f_p(w_t)\Big|\Big|_2^2 \leq \frac{C_1 + C_2 \log T}{\sqrt{T}}, \quad \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\Big|\Big|\nabla f(w_t)\Big|\Big|_2^2 \leq \frac{C_3 + C_4 \log T}{\sqrt{T}}$$

where C_1, C_2, C_3, C_4 *are some constants.*

Thm. 5.5.1 implies both f_p and f converge in GSAM at rate $O(\log T/\sqrt{T})$ for non-convex stochastic optimization, matching the convergence rate of first-order gradient optimizers like Adam.

5.5.2 Generalization of GSAM

In this section, we show the surrogate gap in GSAM is provably lower than SAM's, so GSAM is expected to find a smoother minimum with better generalization.

Theorem 5.5.2 (PAC-Bayesian Theorem [96])**.** *Suppose the training set has* m *elements drawn i.i.d. from the true distribution, and denote the loss on the training set as* $\widehat{f}(w) = \frac{1}{m}\sum_{i=1}^{m} f(w, x_i)$*, where we use* x_i *to denote the (input, target) pair of the* i*-th element. Let* w *be learned from the training set. Suppose* w *is drawn from posterior distribution* $\mathcal{Q}$*. Denote the prior distribution (independent of training) as* $\mathcal{P}$*, then*

$\mathbb{E}_{w\sim\mathcal{Q}}\mathbb{E}_x f(w,x) \leq \mathbb{E}_{w\sim\mathcal{Q}}\widehat{f}(w) + 4\sqrt{\Big(KL(\mathcal{Q}||\mathcal{P}) + \log\frac{2m}{a}\Big)/m}$ *with probability at least* $1-a$

Corollary 5.5.2.1. *Suppose perturbation* δ *is drawn from distribution* $\delta \sim \mathcal{N}(0, b^2 I^k), \delta \in \mathbb{R}^k$*,* k *is the dimension of* w*, then with probability at least* $\Big(1-a\Big)\Big[1 - e^{-\left(\frac{\rho}{\sqrt{2}b} - \sqrt{k}\right)^2}\Big]$

$$\mathbb{E}_{w\sim\mathcal{Q}}\mathbb{E}_x f(w,x) - C \leq \underbrace{\max_{||\delta||_2\leq\rho}\widehat{f}(w+\delta) - C}_{\widehat{h}} + 4\sqrt{\Big(KL(\mathcal{Q}||\mathcal{P}) + \log\frac{2m}{a}\Big)/m} \tag{5.7}$$

$$\widehat{h} \triangleq \max_{||\delta||_2 \leq \rho} \widehat{f}(w+\delta) - \widehat{f}(w) = \frac{1}{m} \sum_{i=1}^{m} \Big[\max_{||\delta||_2 \leq \rho} f(w+\delta, x_i) - f(w, x_i) \Big] \quad (5.8)$$

where $C = \widehat{f}(w)$ is the empirical training loss, and $\widehat{h}$ is the surrogate gap evaluated on the training set.

Corollary 5.5.2.1 implies that minimizing $\widehat{h}$ (right hand side of Eq. 5.7) is expected to achieve a tighter upper bound of the generalization gap (left hand side of Eq. 5.7). The second term on the right of Eq. 5.7 is typically hard to analyze and often simplified to $L2$ regularization [42].

Theorem 5.5.3 (Unlike SAM, GSAM decreases the surrogate gap)**.** *Under the assumption in Thm. 5.5.1, Thm. 5.5.2 and Corollary 5.5.2.1, we assume the Hessian has a lower-bound $|\sigma|_{min}$ on the absolute value of eigenvalue, and the variance of noisy observation $g^{(t)}$ is lower-bounded by c^2. The surrogate gap h can be minimized by the ascent step along the orthogonal direction $g_{\perp}^{(t)}$. During training we minimize the sample estimate of h. We use $\Delta\widehat{h}_t$ to denote the amount that the ascent step in GSAM **decreases** $\widehat{h}$ for the t-th step. Compared to SAM, the proposed method generates a total decrease in surrogate gap $\sum_{t=1}^{T} \Delta\widehat{h}_t$, which is bounded by*

$$\frac{\alpha c^2 \rho_0^2 \eta_0 |\sigma|_{min}^2}{G^2} \leq \lim_{T \to \infty} \sum_{t=1}^{T} \Delta\widehat{h}_t \leq 2.7 \alpha L^2 \eta_0 \rho_0^2 \quad (5.9)$$

We provide proof in the appendix. The lower-bound of $\sum_{t=1}^{T} \Delta\widehat{h}_t$ indicates that GSAM achieves a provably non-trivial decrease in the surrogate gap. Combined with Corollary 5.5.2.1, GSAM provably improves the generalization performance over SAM.

Table 5.1: Top-1 Accuracy (%) on ImageNet datasets for ResNets, ViTs and MLP-Mixers trained with Vanilla SGD or AdamW, SAM, and GSAM optimizers.

Model	Training	ImageNet-v1	ImageNet-Real	ImageNet-V2	ImageNet-R	ImageNet-C
			ResNet			
ResNet50	Vanilla (SGD)	76.0	82.4	63.6	22.2	44.6
	SAM	76.9	83.3	64.4	23.8	46.5
	GSAM	**77.2**	**83.9**	**65.2**	**24.4**	**47.9**
ResNet101	Vanilla (SGD)	77.8	83.9	65.3	24.4	48.5
	SAM	78.6	84.8	66.7	25.9	51.3
	GSAM	**78.9**	**85.2**	**67.1**	**26.3**	**52.1**
ResNet152	Vanilla (SGD)	78.5	84.2	66.3	25.3	50.0
	SAM	79.3	84.9	67.3	25.7	52.2
	GSAM	**79.8**	**85.8**	**68.1**	**26.8**	**53.5**
			Vision Transformer			
ViT-S/32	Vanilla (AdamW)	68.4	75.2	54.3	19.0	43.3
	SAM	70.5	77.5	56.9	21.4	46.2
	GSAM	**73.1**	**80.5**	**59.7**	**23.1**	**47.2**
ViT-S/16	Vanilla (AdamW)	74.4	80.4	61.7	20.0	46.5
	SAM	78.1	84.1	65.6	**24.7**	53.0
	GSAM	**79.5**	**85.1**	**67.0**	24.6	**53.1**
ViT-B/32	Vanilla (AdamW)	71.4	77.5	57.5	23.4	44.0
	SAM	73.6	80.3	60.0	24.0	50.7
	GSAM	**76.8**	**82.8**	**62.8**	**25.3**	**52.0**
ViT-B/16	Vanilla (AdamW)	74.6	79.8	61.3	20.1	46.6
	SAM	79.9	85.2	67.5	26.4	**56.5**
	GSAM	**81.2**	**86.6**	**69.3**	**27.2**	55.6
			MLP-Mixer			
Mixer-S/32	Vanilla (AdamW)	63.9	70.3	49.5	16.9	35.2
	SAM	66.7	73.8	52.4	18.6	39.3
	GSAM	**68.6**	**75.8**	**55.4**	**23.3**	**45.3**
Mixer-S/16	Vanilla (AdamW)	68.8	75.1	54.8	15.9	35.6
	SAM	72.9	79.8	58.9	20.1	42.0
	GSAM	**75.1**	**81.9**	**61.8**	**23.9**	**48.9**
Mixer-S/8	Vanilla (AdamW)	70.2	76.2	56.1	15.4	34.6
	SAM	75.9	82.5	62.3	20.5	42.4
	GSAM	**76.8**	**83.3**	**63.7**	**23.6**	**48.2**
Mixer-B/32	Vanilla (AdamW)	62.5	68.1	47.6	14.6	33.8
	SAM	72.4	79.0	58.0	22.8	46.2
	GSAM	**73.8**	**80.0**	**60.0**	**27.8**	**52.4**
Mixer-B/16	Vanilla (AdamW)	66.4	72.1	50.8	14.5	33.8
	SAM	77.4	83.5	63.9	24.7	48.8
	GSAM	**78.3**	**84.5**	**65.7**	**28.7**	**55.0**

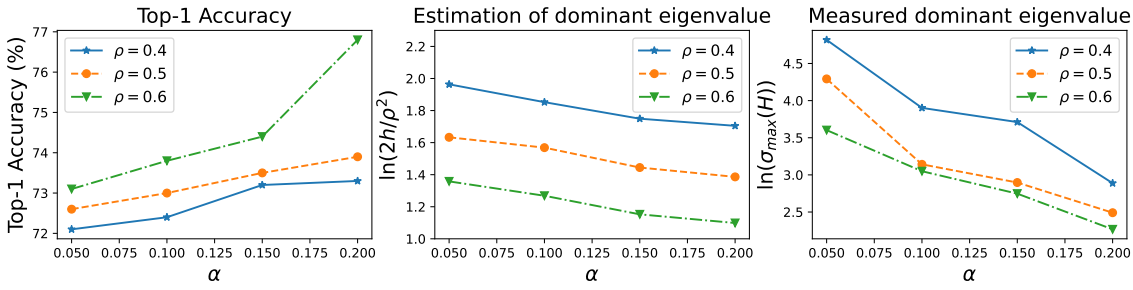


Figure 5.4: Influence of ρ and α on the training of ViT-B/32. **Left**: Top-1 accuracy on ImageNet. **Middle**: Estimation of the dominant eigenvalues from the surrogate gap, $\ln \sigma_{max} \approx \ln(2h/\rho^2)$. **Right**: Dominant eigenvalues of the Hessian calculated via the power iteration. Middle and right figures match in the trend of curves, validating that the surrogate gap can be viewed as a proxy of the dominant eigenvalue of Hessian.

5.6 Experiments

5.6.1 GSAM improves test performance on various model architectures

We conduct experiments with ResNets [53], Vision Transformers (ViTs) [36] and MLP-Mixers [153]. Following the settings by [23], we train on the ImageNet-1k [31] training set using the Inception-style [150] pre-processing without extra training data or strong augmentation. For all models, we search for the best learning rate and weight decay for vanilla training, and then use the same values for the experiments with SAM and GSAM. For ResNets, we search for ρ from 0.01 to 0.05 with a stepsize 0.01. For ViTs and Mixers, we search for ρ from 0.05 to 0.6 with a stepsize 0.05. In GSAM, we search for α in $\{0.01, 0.02, 0.03\}$ for ResNets and α in $\{0.1, 0.2, 0.3\}$ for ViTs and Mixers. Considering that each step in SAM and GSAM requires twice the computation of vanilla training, we experiment with the vanilla training for twice the epochs of SAM and GSAM, but we observe no significant improvements from the longer training (Table **??** in appendix). We summarize the best hyper-parameters for each model in Appendix **??**.

We report the performances on ImageNet [31], ImageNet-v2 [126] and ImageNet-Real [12] in Table 5.1. GSAM consistently improves over SAM and vanilla training (with SGD or AdamW): on ViT-B/32, GSAM achieves +5.4% improvement over AdamW and +3.2% over SAM in top-1 accuracy; on Mixer-B/32, GSAM achieves +11.3% over AdamW and +1.4% over SAM. We ignore the standard deviation since it is typically negligible ($< 0.1\%$) compared to the improvements. We also test the generalization performance on out-of-distribution data (ImageNet-R and ImageNet-C), and the observation is consistent with that on ImageNet, e.g., +5.0% on ImageNet-R and +6.2% on ImageNet-C for Mixer-B/32.

5.6.2 GSAM finds a minimum whose Hessian has small dominant eigenvalues

Lemma 5.3.0.3 indicates that the surrogate gap h is an equivalent measure of the dominant eigenvalue of the Hessian, and minimizing h equivalently searches for a flat minimum. We empirically validate this in Fig. 5.4. As shown in the left subfigure, for some fixed ρ, increasing α decreases the dominant value and improves generalization (test accuracy). In the middle subfigure, we plot the dominant eigenvalues estimated by the surrogate gap, $\sigma_{max} \approx 2h/\rho^2$ (Lemma 5.3.0.3). In the right subfigure, we directly calculate the dominant eigenvalues using the power-iteration [99]. The estimated dominant eigenvalues (middle) match the real eigenvalues σ_{max} (right) in terms of the trend that σ_{max} decreases with α and ρ. Note that the surrogate gap h is derived over the whole training set, while the measured eigenvalues are over a subset to save computation. These results show that the ascent step in GSAM minimizes the dominant eigenvalue by minimizing the surrogate loss, validating Thm 5.5.3.

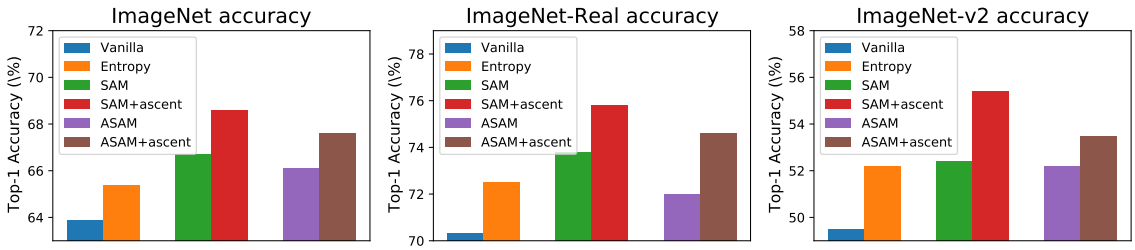


Figure 5.5: Top-1 accuracy of Mixer-S/32 trained with different methods. "+ascent" represents applying the ascent step in Algo. 16 to an optimizer. Note that our GSAM is described as SAM+ascent(**=GSAM**) for consistency.

Table 5.2: Results (%) of GSAM and $\min(f_p+\lambda h)$ on ViT-B/32

Dataset	$\min(f_p+\lambda h)$	GSAM
ImageNet	75.4	**76.8**
ImageNet-Real	81.1	**82.8**
ImageNet-v2	60.9	**62.8**
ImageNet-R	23.9	**25.3**

Table 5.3: Transfer learning results (top-1 accuracy, %)

	ViT-B/16			Mixer-B/16			Mixer-S/16		
	Vanilla	SAM	GSAM	Vanilla	SAM	GSAM	Vanilla	SAM	GSAM
CIFAR10	98.1	98.6	**98.9**	95.4	97.8	**98.5**	94.1	96.1	**98.4**
CIFAR100	87.6	89.1	**89.7**	80.0	86.4	**88.0**	77.9	82.4	**87.8**
Flowers	88.5	**91.8**	91.7	82.8	**90.0**	**90.0**	83.3	87.9	**90.5**
Pets	91.9	93.1	**94.1**	86.1	92.5	**93.5**	86.1	88.7	**93.5**
mean	91.5	93.2	**93.6**	86.1	91.7	**92.5**	85.4	88.8	**92.6**

5.6.3 Comparison with methods in the literature

Section 5.6.1 compares GSAM to SAM and vanilla training. In this subsection, we further compare GSAM against Entropy-SGD [19] and Adaptive-SAM (ASAM) [78], which are designed to improve generalization. Note that Entropy-SGD uses SGD in the inner Langevin iteration and can be combined with other base optimizers such as AdamW as the outer loop. For Entropy-SGD, we find the hyper-parameter "scope" from 0.0 and 0.9, and search for the inner-loop iteration number between 1 and 14. For ASAM, we search for ρ between 1 and 7 ($10\times$ larger than in SAM) as recommended by the ASAM authors. Note that the only difference between ASAM and SAM is the derivation of the perturbation, so both can be combined with the proposed ascent step. As shown in Fig. 5.5, the proposed ascent step increases test accuracy when combined with both SAM and ASAM and outperforms Entropy-SGD and vanilla training.

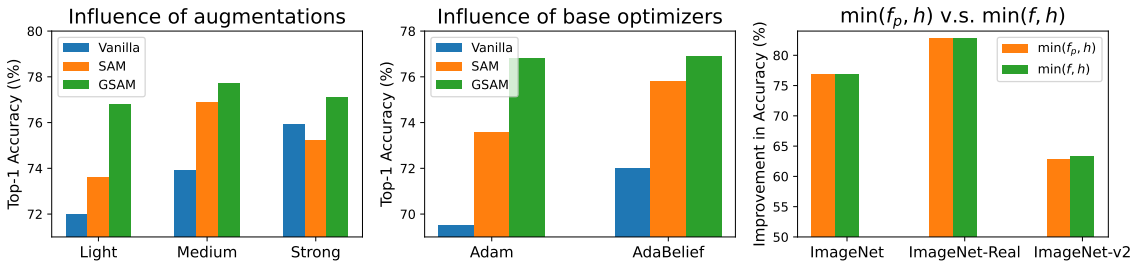


Figure 5.6: Top-1 accuracy of ViT-B/32 for the additional studies (Section 5.6.4). **Left**: from left to right are performances under different data augmentations (details in Appendix **??**) , where vanilla method is trained for $2\times$ the epochs. **Middle**: performance with different base optimizers. **Right**: Comparison between $\min(f_p, h)$ and $\min(f, h)$.

5.6.4 Additional studies

GSAM outperforms a weighted combination of the perturbed loss and surrogate gap With an example in Fig. 5.2, we demonstrate that directly minimizing $f_p(w) + \lambda h(w)$ as discussed in Sec. 5.4.1 is sub-optimal because $\nabla h(w)$ could conflict with $\nabla f_p(w)$ and $\nabla f(w)$. We empirically validate this argument on ViT-B/32. We search for λ between 0.0 and 0.5 with a step 0.1 and search for ρ in the same grid as SAM and GSAM. We report the best accuracy of each method. Top-1 accuracy in Table 5.2 show the superior performance of GSAM, validating our analysis.

$\min(f_p, h)$ vs. $\min(f, h)$ GSAM solves $\min(f_p, h)$ by descent in ∇f_p, decomposing ∇f onto ∇f_p, and an ascent step in the orthogonal direction to increase f while keep f_p intact. Alternatively, we can also optimize $\min(f, h)$ by descent in ∇f, decomposing ∇f_p onto ∇f, and a descent step in the orthogonal direction to decrease f_p while keep f intact. The two GSAM variations perform similarly (see Fig. 5.6, right). We choose $\min(f_p, h)$ mainly to make the minimal change to SAM.

GSAM benefits transfer learning Using weights trained on ImageNet-1k, we finetune models with SGD on downstream tasks including the CIFAR10/CIFAR100 [77], Oxford-flowers [112] and Oxford-IITPets [114]. Results in Table 5.3 shows that GSAM leads to better transfer performance than vanilla training and SAM.

GSAM remains effective under various data augmentations We plot the top-1 accuracy of a ViT-B/32 model under various Mixup [173] augmentations in Fig. 5.6 (left subfigure). Under different augmentations, GSAM consistently outperforms SAM and vanilla training.

GSAM is compatible with different base optimizers GSAM is generic and applicable to various base optimizers. We compare vanilla training, SAM and GSAM using AdamW [90] and AdaBelief [181] with default hyper-parameters. Fig. 5.6 (middle subfigure) shows that GSAM performs the best, and SAM improves over vanilla training.

5.7 Proofs

5.7.1 Proof of Lemma. 5.3.0.1

Suppose ρ is small, perform Taylor expansion around the local minima w, we have:

$$f(w+\delta) = f(w) + \nabla f(w)^\top \delta + \frac{1}{2}\delta^\top H \delta + O(||\delta||^3) \tag{5.10}$$

where H is the Hessian, and is positive semidefinite at a local minima. At a local minima, $\nabla f(w) = 0$, hence we have

$$f(w+\delta) = f(w) + \frac{1}{2}\delta^\top H \delta + O(||\delta||^3) \tag{5.11}$$

and

$$f_p(w) = \max_{||\delta|| \leq \rho} f(w+\delta) = f(w) + \frac{1}{2}\rho^2 \sigma_{max}(H) + O(||\delta||^3) \tag{5.12}$$

where σ_{max} is the dominate eigenvalue (eigenvalue with the largest absolute value). Now consider two local minima w_1 and w_2 with dominate eigenvalue σ_1 and σ_2 respectively,

we have

$$f_p(w_1) \approx f(w_1) + \frac{1}{2}\rho^2\sigma_1 \qquad\qquad f_p(w_2) \approx f(w_2) + \frac{1}{2}\rho^2\sigma_2$$

We have $f_p(w_1) > f_p(w_2) \implies \sigma_1 > \sigma_2$ and $\sigma_1 > \sigma_2 \implies f_p(w_1) > f_p(w_2)$ because the relation between $f(w_1)$ and $f(w_2)$ is undetermined. □

5.7.2 Proof of Lemma. 5.3.0.2

Since ρ is small, we can perform Taylor expansion around w,

$$\begin{aligned} h(w) &= f(w+\delta) - f(w) \\ &= \delta^\top \nabla f(w) + O(\rho^2) \\ &= \rho||\nabla f(w)||_2 + O(\rho^2) > 0 \end{aligned} \tag{5.13}$$

where the last line is because δ is approximated as $\delta = \rho\frac{\nabla f(w)}{||\nabla f(w)||_2+\epsilon}$, hence has the same direction as $\nabla f(w)$. □

5.7.3 Proof of Lemma. 5.3.0.3

Since ρ is small, we can approximate $f(w)$ with a quadratic model around a local minima w:

$$f(w+\delta) = f(w) + \frac{1}{2}\delta^\top H\delta + O(\rho^3)$$

where H is the Hessian at w, assumed to be positive semidefinite at local minima. Normalize δ such that $||\delta||_2 = \rho$, Hence we have:

$$h(w) = f_p(w) - f(w) = \max_{||\delta||_2\leq\rho} f(w+\delta) - f(w) = \frac{1}{2}\sigma_{max}\rho^2 + O(\rho^3) \tag{5.14}$$

where σ_{max} is the dominate eigenvalue of the hessian H, and first order term is 0 because the gradient is 0 at local minima. Therefore, we have $\sigma_{max} \approx 2h(w)/\rho^2$. □

5.7.4 Proof of Thm. 5.5.1

For simplicity we consider the base optimizer is SGD. For other optimizers such as Adam, we can derive similar results by applying standard proof techniques in the literature to our proof.

Step 1: Convergence w.r.t function $f_p(w)$

By $L-$smoothness of $f(w)$, we have

$$||\nabla f(w_1) - \nabla f(w_2)|| \leq L||w_1 - w_2|| \tag{5.15}$$

Hence we can derive the smoothness of $f_p(w)$

$$\Big|\Big|\nabla f_p(w_1) - \nabla f_p(w_2)\Big|\Big| = \Big|\Big|\nabla f\Big(w_1 + \rho\frac{\nabla f(w_1)}{||\nabla f(w_1)|| + \epsilon}\Big) - \nabla f\Big(w_2 + \rho\frac{\nabla f(w_2)}{||\nabla f(w_2)|| + \epsilon}\Big)\Big|\Big| \tag{5.16}$$

$$\leq L\Big|\Big|\Big(w_1 + \rho\frac{\nabla f(w_1)}{||\nabla f(w_1)|| + \epsilon}\Big) - \Big(w_2 + \rho\frac{\nabla f(w_2)}{||\nabla f(w_2)|| + \epsilon}\Big)\Big|\Big| \tag{5.17}$$

$$\Big(f \text{ is Lipschitz}\Big)$$

$$\leq L\Big|\Big|w_1 - w_2\Big|\Big| + L\rho\Big|\Big|\frac{\nabla f(w_1)}{||\nabla f(w_1)|| + \epsilon} - \frac{\nabla f(w_2)}{||\nabla f(w_2)|| + \epsilon}\Big|\Big| \tag{5.18}$$

$$\leq L\Big|\Big|w_1 - w_2\Big|\Big| + \frac{L\rho}{\epsilon}\Big|\Big|\nabla f(w_1) - \nabla f(w_2)\Big|\Big| \tag{5.19}$$

$$\Big(\frac{x}{||x|| + \epsilon} \text{ is Lipschitz-continuous with constant } \frac{1}{\epsilon}\Big)$$

$$\leq \Big(L + \frac{L^2\rho}{\epsilon}\Big)\Big|\Big|w_1 - w_2\Big|\Big| \tag{5.20}$$
$$\Big(f \text{ is Lipschitz}\Big)$$

Hence we prove the smoothness of f_p. For simplicity, we denote $L_p = L + \frac{L^2\rho}{\epsilon}$ as the Lipschitz constant of $f_p(w)$.

For simplicity of notation, we denote the update at step t as

$$d_t = -\eta_t g_p^{(t)} + \eta_t \alpha g_\perp^{(t)} \tag{5.21}$$

By smoothness of $f_p(w)$, we have

$$f_p(w_{t+1}) \leq f_p(w_t) + \langle \nabla f_p(w_t), d_t\rangle + \frac{L_p}{2}\Big|\Big|d_t\Big|\Big|_2^2 \tag{5.22}$$
$$= f_p(w_t) + \langle \nabla f_p(w_t), -\eta_t g_p^{(t)} + \alpha\eta_t g_\perp^{(t)}\rangle + \frac{L_p}{2}\Big|\Big|d_t\Big|\Big|_2^2 \tag{5.23}$$

Take expectation conditioned on observation up to step t (for simplicity of notation, we use $\mathbb{E}$ short for $\mathbb{E}_x$ to denote expectation over all possible data points) conditioned on observations up to step t, we have

$$\mathbb{E}f_p(w_{t+1}) - f_p(w_t) \leq -\eta_t\langle \nabla f_p(w_t), \mathbb{E}g_p^{(t)}\rangle + \alpha\eta_t\langle \nabla f_p(w_t), \mathbb{E}g_\perp^{(t)}\rangle$$
$$+ \frac{L_p}{2}\eta_t^2\mathbb{E}\Big|\Big| - g_p^{(t)} + \alpha g_\perp^{(t)}\Big|\Big|_2^2 \tag{5.24}$$
$$\leq -\eta_t\mathbb{E}\Big|\Big|\nabla f_p(w_t)\Big|\Big|_2^2 + 0 + \frac{L_p(\alpha+1)^2}{2}G^2\eta_t^2 \tag{5.25}$$
$$\Big(\text{Since } \mathbb{E}g_\perp^{(t)} \text{ is orthogonal to } \nabla f_p(w_t) \text{ by construction},$$
$$||g^{(t)}|| \leq G \text{ by assumption}\Big)$$

Hence we have

$$\eta_t \mathbb{E}\Big|\Big|\nabla f_p(w_t)\Big|\Big|_2^2 \leq f_p(w_t) - \mathbb{E}f_p(w_{t+1}) + \frac{L_p(\alpha+1)^2 G^2}{2}\eta_t^2 \tag{5.26}$$

By telescope sum, we have:

$$\sum_{t=1}^{T} \eta_t \mathbb{E}\Big|\Big|\nabla f_p(w_t)\Big|\Big|_2^2 \leq f_p(w_0) - \mathbb{E}f_p(w_T) + \frac{L_p(\alpha+1)^2 G^2}{2}\sum_{t=1}^{T}\eta_t^2 \tag{5.27}$$

Since $\eta_t = \eta_0/\sqrt{t}$, we have

$$\frac{\eta_0}{\sqrt{T}}\sum_{t=1}^{T}\mathbb{E}\Big|\Big|\nabla f_p(w_t)\Big|\Big|_2^2 \leq LHS \leq RHS \tag{5.28}$$

$$\leq f_p(w_0) - f_{min} + \frac{L_p(1+\alpha)^2 G^2 \eta_0^2}{2}\Big(1+\log T\Big) \tag{5.29}$$

$$\tag{5.30}$$

Hence

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\Big|\Big|\nabla f_p(w_t)\Big|\Big|_2^2 \leq \Big[\frac{f_p(w_0) - f_{min}}{\eta_0} + \frac{L_p(1+\alpha)^2 G^2 \eta_0^2}{2}\Big(1+\log T\Big)\Big]/\sqrt{T} \tag{5.31}$$

$$= \frac{C_1 + C_2 \log T}{\sqrt{T}} \tag{5.32}$$

where C_1, C_2 are some constants. This implies that the regret w.r.t $f_p(w)$ converges at rate $O(\log T/\sqrt{T})$.

Step 2: Convergence w.r.t. function $f(w)$

We prove the risk for $f(w)$ convergences for non-convex stochastic optimization case using SGD. Denote the update at step t as

$$d_t = -\eta_t g_p^{(t)} + \alpha\eta_t g_\perp^{(t)} \tag{5.33}$$

By smoothness of f, we have

$$f(w_{t+1}) \leq f(w_t) + \langle \nabla f(w_t), d_t \rangle + \frac{L}{2}\Big|\Big|d_t\Big|\Big|_2^2 \tag{5.34}$$

$$= f(w_t) + \langle \nabla f(w_t), -\eta_t g_p^{(t)} + \alpha\eta_t g_\perp^{(t)} \rangle + \frac{L}{2}\Big|\Big|d_t\Big|\Big|_2^2 \tag{5.35}$$

For simplicity, we introduce a scalar β_t such that

$$\nabla f_\parallel(w_t) = \beta_t \nabla f_p(w_t) \tag{5.36}$$

where $\nabla f_\parallel(w_t)$ is the projection of $\nabla f(w_t)$ onto $\nabla f_p(w_t)$. When perturbation amplitude ρ is small, we expect β_t to be very close to 1.

Take expectation conditioned on observations up to step t for both sides of Eq. 5.35, we have:

$$\mathbb{E}f(w_{t+1}) \leq f(w_t) + \left\langle \nabla f(w_t), -\frac{\eta_t}{\beta_t}\Big(\nabla f(w_t) - \nabla f_\perp(w_t)\Big) + \alpha\eta_t \mathbb{E}g_\perp^{(t)} \right\rangle + \frac{L}{2}\mathbb{E}\Big|\Big|d_t\Big|\Big|_2^2 \tag{5.37}$$

$$= f(w_t) - \frac{\eta_t}{\beta_t}\Big|\Big|\nabla f(w_t)\Big|\Big|_2^2 + \Big(\frac{1}{\beta_t} + \alpha\Big)\eta_t\Big\langle \nabla f(w_t), \nabla f_\perp(w_t)\Big\rangle + \frac{L}{2}\mathbb{E}\Big|\Big|d_t\Big|\Big|_2^2 \tag{5.38}$$

$$= f(w_t) - \frac{\eta_t}{\beta_t}\Big|\Big|\nabla f(w_t)\Big|\Big|_2^2 + \Big(\frac{1}{\beta_t} + \alpha\Big)\eta_t\Big\langle \nabla f(w_t), \nabla f(w_t)\sin\theta_t\Big\rangle + \frac{L}{2}\mathbb{E}\Big|\Big|d_t\Big|\Big|_2^2 \tag{5.39}$$

$$\Big(\theta_t \text{ is the angle between } \nabla f_p(w_t) \text{ and } \nabla f(w_t)\Big)$$

$$= f(w_t) - \frac{\eta_t}{\beta_t}\Big|\Big|\nabla f(w_t)\Big|\Big|_2^2 + \Big(\frac{1}{\beta_t} + \alpha\Big)\eta_t\Big|\Big|\nabla f(w_t)\Big|\Big|_2^2(|\tan\theta_t| + O(\theta_t^2)) + \frac{L}{2}\mathbb{E}\Big|\Big|d_t\Big|\Big|_2^2 \tag{5.40}$$

$$\Big(\sin x = x + O(x^2), \tan x = x + O(x^2) \text{ when } x \to 0.\Big)$$

Also note when perturbation amplitude ρ_t is small, we have

$$\nabla f_p(w_t) = \nabla f(w_t + \delta_t) = \nabla f(w_t) + \frac{\rho_t}{||\nabla f(w_t)||_2 + \epsilon}H(w_t)\nabla f(w_t) + O(\rho_t^2) \tag{5.41}$$

where $\delta_t = \rho_t \frac{\nabla f(w_t)}{||\nabla f(w_t)||_2}$ by definition, $H(w_t)$ is the Hessian. Hence we have

$$|\tan\theta_t| \leq \frac{||\nabla f_p(w_t) - \nabla f(w_t)||}{||\nabla f(w_t)||} \leq \frac{\rho_t L}{||\nabla f(w_t)||} \tag{5.42}$$

where L is the Lipschitz constant of f, and $L-$smoothness of f indicates the maximum absolute eigenvalue of H is upper bounded by L. Plug Eq. 5.42 into Eq. 5.40, we have

$$\mathbb{E}f(w_{t+1}) \leq f(w_t) - \frac{\eta_t}{\beta_t}\Big|\Big|\nabla f(w_t)\Big|\Big|_2^2 + \Big(\frac{1}{\beta_t} + \alpha\Big)\eta_t\Big|\Big|\nabla f(w_t)\Big|\Big|_2^2|\tan\theta_t| + \frac{L}{2}\mathbb{E}\Big|\Big|d_t\Big|\Big|_2^2 \tag{5.43}$$

$$\leq f(w_t) - \frac{\eta_t}{\beta_t}\Big|\Big|\nabla f(w_t)\Big|\Big|_2^2 + \Big(\frac{1}{\beta_t} + \alpha\Big)L\rho_t\eta_t\Big|\Big|\nabla f(w_t)\Big|\Big|_2 + \frac{L}{2}\mathbb{E}\Big|\Big|d_t\Big|\Big|_2^2 \tag{5.44}$$

$$\leq f(w_t) - \frac{\eta_t}{\beta_t}\Big|\Big|\nabla f(w_t)\Big|\Big|_2^2 + \Big(\frac{1}{\beta_t} + \alpha\Big)L\rho_t\eta_t G + \frac{L}{2}\mathbb{E}\Big|\Big|d_t\Big|\Big|_2^2 \tag{5.45}$$

$$\Big(\text{Assume gradient has bounded norm } G.\Big) \tag{5.46}$$

$$\leq f(w_t) - \frac{\eta_t}{\beta_{max}}\Big|\Big|\nabla f(w_t)\Big|\Big|_2^2 + \Big(\frac{1}{\beta_{min}} + \alpha\Big)L\rho_t\eta_t G + \frac{L}{2}\mathbb{E}(\alpha+1)^2G^2\eta_t^2 \tag{5.47}$$

$\Big($$\beta_t$ is close to 1 assuming ρ is small,

hence it's natural to assume $0 < \beta_{min} \leq \beta_t \leq \beta_{max}\Big)$

Re-arranging above formula, we have

$$\frac{\eta_t}{\beta_{max}}\Big|\Big|\nabla f(w_t)\Big|\Big|_2^2 \leq f(w_t) - \mathbb{E}f(w_{t+1}) + \Big(\frac{1}{\beta_{min}} + \alpha\Big)LG\eta_t\rho_t + \frac{L}{2}(\alpha+1)^2G^2\eta_t^2 \tag{5.48}$$

perform telescope sum and taking expectations on each step, we have

$$\frac{1}{\beta_{max}}\sum_{t=1}^{T}\eta_t\Big|\Big|\nabla f(w_t)\Big|\Big|_2^2 \leq f(w_0) - \mathbb{E}f(w_T) + \Big(\frac{1}{\beta_{min}} + \alpha\Big)LG\sum_{t=1}^{T}\eta_t\rho_t + \frac{L}{2}(\alpha+1)^2G^2\sum_{t=1}^{T}\eta_t^2 \tag{5.49}$$

Take the schedule to be $\eta_t = \frac{\eta_0}{\sqrt{t}}$ and $\rho_t = \frac{\rho_0}{\sqrt{t}}$, then we have

$$\frac{\eta_0}{\beta_{max}}\frac{1}{\sqrt{T}}\sum_{t=1}^{T}\Big|\Big|\nabla f(w_t)\Big|\Big|_2^2 \leq LHS \tag{5.50}$$

$$\leq RHS \tag{5.51}$$

$$\leq f(w_0) - f_{min} + \Big(\frac{1}{\beta_{min}} + \alpha\Big)LG\eta_0\rho_0\sum_{t=1}^{T}\frac{1}{t} + \frac{L}{2}(\alpha+1)^2G^2\eta_0^2\sum_{t=1}^{T}\frac{1}{t} \tag{5.52}$$

$$\leq f(w_0) - f_{min} + \Big(\frac{1}{\beta_{min}} + \alpha\Big)LG\eta_0\rho_0(1+\log T) + \frac{L}{2}(\alpha+1)^2G^2\eta_0^2(1+\log T) \tag{5.53}$$

Hence

$$\frac{1}{T}\sum_{t=1}^{T}\Big|\Big|\nabla f(w_t)\Big|\Big|_2^2 \leq \frac{C_3}{\sqrt{T}} + C_4\frac{\log T}{\sqrt{T}} \tag{5.54}$$

where C_1, C_4 are some constants. This implies the convergence rate w.r.t $f(w)$ is $O(\log T/\sqrt{T})$.

Step 3: Convergence w.r.t. surrogate gap $h(w)$

Note that we have proved convergence for $f_p(w)$ in step 1, and convergence for $f(w)$ in step 3. Also note that

$$\Big|\Big|\nabla h(w_t)\Big|\Big|_2^2 = \Big|\Big|\nabla f_p(w_t) - \nabla f(w_t)\Big|\Big|_2^2 \leq 2\Big|\Big|\nabla f_p(w_t)\Big|\Big|_2^2 + 2\Big|\Big|\nabla f(w_t)\Big|\Big|_2^2 \tag{5.55}$$

Hence

$$\frac{1}{T}\sum_{t=1}^{T}\Big|\Big|\nabla h(w_t)\Big|\Big|_2^2 \leq \frac{2}{T}\sum_{t=1}^{T}\Big|\Big|\nabla f_p(w_t)\Big|\Big|_2^2 + \frac{2}{T}\sum_{t=1}^{T}\Big|\Big|\nabla f(w_t)\Big|\Big|_2^2 \tag{5.56}$$

also converges at rate $O(\log T/\sqrt{T})$ because each item in the RHS converges at rate $O(\log T\sqrt{T})$. □

5.7.5 Proof of Corollary. 5.5.2.1

Using the results from Thm. 5.5.2, with probability at least $1-a$, we have

$$\mathbb{E}_{w\sim\mathcal{Q}}\mathbb{E}_x f(w,x) \leq \mathbb{E}_{w\sim\mathcal{Q}}\widehat{f}(w) + 4\sqrt{\frac{KL(\mathcal{Q}||\mathcal{P}) + \log\frac{2m}{a}}{m}} \tag{5.57}$$

Assume $\delta \sim \mathcal{N}(0, b^2 I_k)$ where k is the dimension of model parameters, hence δ^2 (element-wise square) follows a a Chi-square distribution. By Lemma.1 in [79], we have

$$\mathbb{P}\big(||\delta||_2^2 - kb^2 \geq 2b^2\sqrt{kt} + 2tb^2\big) \leq exp(-t) \tag{5.58}$$

hence with probability at least $1 - 1/\sqrt{n}$, we have

$$||\delta||_2^2 \leq b^2\left(2\log\sqrt{n} + k + 2\sqrt{k\log\sqrt{n}}\right) \leq 2b^2k\left(1 + \sqrt{\frac{\log\sqrt{n}}{k}}\right)^2 \leq \rho^2 \tag{5.59}$$

Therefore, with probability at least $1 - 1/\sqrt{n} = 1 - exp\Big(- \big(\frac{\rho}{\sqrt{2}b} - \sqrt{k}\big)^2\Big)$

$$\mathbb{E}_\delta \widehat{f}(w+\delta) \leq \max_{||\delta||_2 \leq \rho} \widehat{f}(w+\delta) \tag{5.60}$$

Combine Eq. 5.58 and Eq. 5.60, subtract the same constant C on both sides, and under the same assumption as in [42] that $\mathbb{E}_{w\sim\mathcal{Q}}\mathbb{E}_x f(w,x) \leq \mathbb{E}_{\delta\sim\mathcal{N}(0,b^2I^k)}\mathbb{E}_{w\sim\mathcal{Q}}\mathbb{E}_x f(w+\delta,x)$we finish the proof. □

5.7.6 Proof of Thm. 5.5.3

Step 1: a sufficient condition that the loss gap is expected to decrease for each step

Take Taylor expansion, then the expected change of loss gap caused by descent step is

$$\mathbb{E}\langle \nabla f_p(w_t) - \nabla f(w_t), -\eta_t \nabla f_p(w_t)\rangle \tag{5.61}$$

$$\Big(\textit{where } \mathbb{E}g_\perp = \nabla f_\perp(w_t)\Big)$$

$$= \eta_t \Bigg[- \big|\big|\nabla f_p(w_t)\big|\big|_2^2 + \big|\big|\nabla f_p(w_t)\big|\big|_2 \big|\big|\nabla f(w_t)\big|\big|_2 \cos\theta_t \Bigg] \tag{5.62}$$

where θ_t is the angle between vector $\nabla f_p(w_t)$ and $\nabla f(w_t)$.

The expected change of loss gap caused by ascent step is

$$\mathbb{E}\langle \nabla f_p(w_t) - \nabla f(w_t), \alpha\eta_t \nabla f_\perp(w_t)\rangle = -\alpha\eta_t \big|\big|\nabla f_\perp(w_t)\big|\big|_2^2 < 0 \tag{5.63}$$

Above results demonstrate that ascent step decreases the loss gap, while descent step might increase the loss gap. A sufficient (but not necessary) condition for $\mathbb{E}\langle \nabla h(w_t), dt\rangle \leq 0$ requires α to be large or $\big|\big|\nabla f(w_t)\big|\big|_2 \cos\theta_t \leq \big|\big|\nabla f_p(w_t)\big|\big|$. In practice, the perturbation amplitude ρ is small and we can assume θ_t is close to 0 and $\big|\big|\nabla f_p(w_t)\big|\big|$ is close to

$\big|\big|\nabla f(w_t)\big|\big|$, we can also set the parameter α to be large in order to decrease the loss gap.

Step 2: upper and lower bound of decrease in loss gap (by the ascent step in orthogonal gradient direction) compared to SAM.

Next we give an estimate of the decrease in $\widehat{h}$ caused by our ascent step. We refer to Eq. 5.62 and Eq. 5.63 to analyze the change in loss gap caused by the descent and ascent (orthogonally) respectively. It can be seen that gradient descent step might not decrease loss gap, in fact they often increase loss gap in practice; while the ascent step is guaranteed to decrease the loss gap.

The decrease in loss gap is:

$$\Delta\widehat{h}_t = -\langle \nabla\widehat{f}_p(w_t) - \nabla\widehat{f}(w_t), \alpha\eta_t \nabla\widehat{f}_\perp(w_t)\rangle = \alpha\eta_t \big|\big|\nabla\widehat{f}_\perp(w_t)\big|\big|_2^2 \tag{5.64}$$

$$= \alpha\eta_t \big|\big|\nabla\widehat{f}(w_t)\big|\big|_2^2 |\tan\theta_t|^2 \tag{5.65}$$

$$\sum_{t=1}^{T} \Delta\widehat{h}_t \leq \sum_{t=1}^{T} \alpha L^2 \eta_t \rho_t^2 \tag{5.66}$$

$$\Big(\text{By Eq. 5.42}\Big) \tag{5.67}$$

$$\leq \sum_{t=1}^{T} \alpha L^2 \eta_0 \rho_0^2 \frac{1}{t^{3/2}} \tag{5.68}$$

$$\leq 2.7\alpha L^2 \eta_0 \rho_0^2 \tag{5.69}$$

Hence we derive an upper bound for $\sum_{t=1}^{T} \Delta\widehat{h}_t$.

Next we derive a lower bound for $\sum_{t=1}^{T} \Delta\widehat{h}_t$ Note that when ρ_t is small, by Taylor expansion

$$\nabla\widehat{f}_p(w_t) = \nabla\widehat{f}(w_t + \delta_t) = \nabla\widehat{f}(w_t) + \frac{\rho_t}{||\nabla\widehat{f}(w_t)||}\widehat{H}(w_t)\nabla\widehat{f}(w_t) + O(\rho_t^2) \tag{5.70}$$

where $\widehat{H}(w_t)$ is the Hessian evaluated on training samples. Also when ρ_t is small, the angle θ_t between $\nabla \widehat{f}_p(w_t)$ and $\nabla \widehat{f}(w_t)$ is small, by the limit that

$$\tan x = x + O(x^2), x \to 0$$

$$\sin x = x + O(x^2), x \to 0$$

We have

$$|\tan \theta_t| = |\sin \theta_t| + O(\theta_t^2) = |\theta_t| + O(\theta_t^2)$$

Omitting high order term, we have

$$|\tan \theta_t| \approx |\theta_t| = \frac{||\nabla \widehat{f}_p(w_t) - \nabla \widehat{f}(w_t)||}{||\widehat{f}(w_t)||} = \frac{||\rho_t \widehat{H}(w_t) + O(\rho_t^2)||}{||\nabla \widehat{f}(w_t)||} \geq \frac{\rho_t |\sigma|_{min}}{G} \tag{5.71}$$

where G is the upper-bound on norm of gradient, $|\sigma|_{min}$ is the minimum absolute eigenvalue of the Hessian. The intuition is that as perturbation amplitude decreases, the angle θ_t decreases at a similar rate, though the scale constant might be different. Hence we have

$$\sum_{t=1}^{T} \Delta \widehat{h}_t = \sum_{t=1}^{T} \alpha \eta_t \big|\big|\nabla \widehat{f}(w_t)\big|\big|_2^2 |\tan \theta_t|^2 + O(\theta_t^4) \tag{5.72}$$

$$\geq \sum_{t=1}^{T} \alpha \eta_t c^2 \Big(\frac{\rho_t |\sigma|_{min}}{G}\Big)^2 \tag{5.73}$$

$$= \frac{\alpha c^2 \rho_0^2 \eta_0 |\sigma|_{min}^2}{G^2} \sum_{t=1}^{T} \frac{1}{t^{3/2}} \tag{5.74}$$

$$\geq \frac{\alpha c^2 \rho_0^2 \eta_0 |\sigma|_{min}^2}{G^2} \tag{5.75}$$

where c^2 is the lower bound of $||\nabla \widehat{f}||^2$ (e.g. due to noise in data and gradient observation). Results above indicate that the decrease in loss gap caused by the ascent step is non-trivial, hence our proposed method efficiently improves generalization compared with SAM. $\square$

5.7.7 Proof for convergence of GSAM without relying on the L-smoothness of f_p

In this section, we provide a proof for the convergence of GSAM without replying on the L-smoothness of f_p.

By $L-$smoothness of f and the definition of $f_p(w_t) = f(w_t^{adv})$, and definition of $d_t = w_{t+1} - w_t$ and $w_t^{adv} = w_t + \delta_t$ we have

$$f_p(w_{t+1}) = f(w_{t+1}^{adv}) \leq f(w_t^{adv}) + \langle \nabla f(w_t^{adv}), w_{t+1}^{adv} - w_t^{adv} \rangle + \frac{L}{2} \Big|\Big| w_{t+1}^{adv} - w_t^{adv} \Big|\Big|^2 \tag{5.76}$$

$$= f(w_t^{adv}) + \langle \nabla f(w_t^{adv}), w_{t+1} + \delta_{t+1} - w_t - \delta_t \rangle$$

$$+ \frac{L}{2} \Big|\Big| w_{t+1} + \delta_{t+1} - w_t - \delta_t \Big|\Big|^2 \tag{5.77}$$

$$\leq f(w_t^{adv}) + \langle \nabla f(w_t^{adv}), d_t \rangle + L \Big|\Big| d_t \Big|\Big|^2 \tag{5.78}$$

$$+ \langle \nabla f(w_t^{adv}), \delta_{t+1} - \delta_t \rangle + L \Big|\Big| \delta_{t+1} - \delta_t \Big|\Big|^2 \tag{5.79}$$

It's trivial to see that Eq. 5.78 has the same form as Eq. 5.22, except the constant L_p is replaced with a constant $2L$ which is even better bounded, so we can reuse RHS of Eq. 5.25 by replacing L_p with $2L$. Now we focus on the difference caused by Eq. 5.79.

By definition of δ_t, we have

$$\delta_t = \rho_t \frac{g^{(t)}}{||g^{(t)}|| + \epsilon} \tag{5.80}$$

$$\delta_{t+1} = \rho_{t+1} \frac{g^{(t+1)}}{||g^{(t+1)}|| + \epsilon} \tag{5.81}$$

where $g^{(t)}$ is the gradient of f at w_t evaluated with a noisy data sample. When learning

rate η_t is small, the update in weight d_t is small, and expected gradient is

$$\nabla f(w_{t+1}) = \nabla f(w_t + d_t) = \nabla f(w_t) + Hd_t + O(||d_t||^2) \tag{5.82}$$

where H is the Hessian at w_t. Therefore, we have

$$\mathbb{E}\langle \nabla f(w_t^{adv}), \delta_{t+1} - \delta_t \rangle = \langle \nabla f(w_t^{adv}), \rho_t \mathbb{E}\frac{g^{(t)}}{||g^{(t)}|| + \epsilon} - \rho_{t+1}\mathbb{E}\frac{g^{(t+1)}}{||g^{(t+1)}|| + \epsilon} \rangle \tag{5.83}$$

$$\leq ||\nabla f(w_t^{adv})|| \rho_t \Big|\Big| \mathbb{E}\frac{g^{(t)}}{||g^{(t)}|| + \epsilon} - \mathbb{E}\frac{g^{(t+1)}}{||g^{(t+1)}|| + \epsilon} \Big|\Big| \tag{5.84}$$

$$\leq ||\nabla f(w_t^{adv})|| \rho_t \phi_t \tag{5.85}$$

where the first inequality is due to (1) ρ_t is monotonically decreasing with t, and (2) triangle inequality that $\langle a, b \rangle \leq ||a|| \cdot ||b||$. ϕ_t is the angle between the unit vector in the direction of $\nabla f(w_t)$ and $\nabla f(w_{t+1})$. The second inequality comes from that (1) $\Big|\Big| \frac{g}{||g|| + \epsilon} \Big|\Big| < 1$ strictly, so we can replace δ_t in Eq. 5.83 with a unit vector in corresponding directions multiplied by ρ_t and get the upper bound, (2) the norm of difference in unit vectors can be upper bounded by the arc length on a unit circle.

When learning rate η_t and update stepsize d_t is small, ϕ_t is also small. Using the limit that

$$\tan x = x + O(x^2), \quad \sin x = x + O(x^2), \quad x \to 0$$

We have:

$$\tan \phi_t = \frac{||\nabla f(w_{t+1}) - \nabla f(w_t)||}{||\nabla f(w_t)||} + O(\phi_t^2) \tag{5.86}$$

$$= \frac{||Hd_t + O(||d_t||^2)||}{||\nabla f(w_t)||} + O(\phi_t^2) \tag{5.87}$$

$$\leq \eta_t L(1 + \alpha) \tag{5.88}$$

where the last inequality is due to (1) max eigenvalue of H is upper bounded by L because f is $L-$smooth, (2) $||d_t|| = ||\eta_t(g_\parallel + \alpha g_\perp)||$ and $\mathbb{E}g_t = \nabla f(w_t)$.

Plug into Eq. 5.85, also note that the perturbation amplitude ρ_t is small so w_t is close to w_t^{adv}, then we have

$$\mathbb{E}\langle \nabla f(w_t^{adv}), \delta_{t+1} - \delta_t \rangle \leq L(1+\alpha)G\rho_t\eta_t \tag{5.89}$$

Similarly, we have

$$\mathbb{E}\Big|\Big|\delta_{t+1} - \delta_t\Big|\Big|^2 \leq \rho_t^2 \mathbb{E}\Big|\Big|\frac{g^{(t)}}{||g^{(t)}|| + \epsilon} - \frac{g^{(t+1)}}{||g^{(t+1)}|| + \epsilon}\Big|\Big|^2 \tag{5.90}$$

$$\leq \rho_t^2 \phi_t^2 \tag{5.91}$$

$$\leq \rho_t^2 \eta_t^2 L^2 (1+\alpha)^2 \tag{5.92}$$

Reuse results from Eq. 5.25 (replace L_p with $2L$) and plug into Eq. 5.78, and plug Eq. 5.89 and Eq. 5.92 into Eq. 5.79, we have

$$\begin{aligned}\mathbb{E}f_p(w_{t+1}) - f_p(w_t) \leq -\eta_t \mathbb{E}\Big|\Big|\nabla f_p(w_t)\Big|\Big|_2^2 + \frac{2L(\alpha+1)^2}{2} G^2\eta_t^2 \\ + L(1+\alpha)G\rho_t\eta_t + \frac{2L^3(1+\alpha)^2}{2}\eta_t^2\rho_t^2\end{aligned} \tag{5.93}$$

Perform telescope sum, we have

$$\begin{aligned}\mathbb{E}f_p(w_T) - f_p(w_0) \leq -\sum_{t=1}^{T} \eta_t \mathbb{E}||\nabla f_p(w_t)||^2 + \Big[L(1+\alpha)^2 G^2 \eta_0^2 + L(1+\alpha)G\rho_0\eta_0\Big]\sum_{t=1}^{T}\frac{1}{t} \\ + L^3(1+\alpha)^2\eta_0^2\rho_0^2\sum_{t=1}^{T}\frac{1}{t^2}\end{aligned} \tag{5.94}$$

Hence

$$\eta_T \sum_{t=1}^{T} \mathbb{E}||\nabla f_p(w_t)||^2 \leq \sum_{t=1}^{T} \eta_t \mathbb{E}||\nabla f_p(w_t)||^2 \leq f_p(w_0) - \mathbb{E}f_p(w_T) + D\log T + \frac{\pi^2 E}{6} \tag{5.95}$$

where

$$D = L(1+\alpha)^2 G^2 \eta_0^2 + L(1+\alpha) G \rho_0 \eta_0, \quad E = L^3 (1+\alpha)^2 \eta_0^2 \rho_0^2 \tag{5.96}$$

Note that $\eta_T = \frac{\eta_0}{\sqrt{T}}$, we have

$$\frac{1}{T} \sum_{t=1}^{T} \mathbb{E}||\nabla f_p(w_t)||^2 \leq \frac{f_p(w_0) - f_{min} + \pi^2 E/6}{\eta_0} \frac{1}{\sqrt{T}} + \frac{D}{\eta_0} \frac{\log T}{\sqrt{T}} \tag{5.97}$$

which implies that GSAM converges at a rate of $O(\log T/\sqrt{T})$, and all the constants here are well-bounded.

5.8 Related works

Besides SAM and ASAM, other methods were proposed in the literature to improve generalization: [85] proposed extrapolation of gradient, [166] proposed to manipulate the noise in gradient, and [27] proved label noise improves generalization, [168] proposed to adjust learning rate according to sharpness, and [175] proposed model perturbation with similar idea to SAM. [63] proposed averaging weights to improve generalization, and [54] restricted the norm of updated weights to improve generalization. Many of aforementioned methods can be combined with GSAM to further improve generalization.

Besides modified training schemes, there are other two types of techniques to improve generalization: data augmentation and model regularization. Data augmentation typically generates new data from training samples; besides standard data augmentation such as flipping or rotation of images, recent data augmentations include label smoothing [104] and mixup [104] which trains on convex combinations of both inputs and labels, automatically learned augmentation [26], and cutout [32] which randomly masks out parts of an image. Model regularization typically applies auxiliary losses besides the training loss such as weight decay [90], other methods randomly modify the model architecture during training, such as dropout [146] and shake-shake regularization [44]. Note that the data augmentation and model regularization literature mentioned here typically train with the standard back-propagation [133] and first-order gradient optimizers, and both techniques can be combined with GSAM.

Besides SGD, Adam and AdaBelief, GSAM can be combined with other first-order gradient optimizers, such as AdaBound [92], RAdam [88], Yogi [169], AdaGrad [37], AMSGrad [127] and AdaDelta [170].

Chapter 6

Apply MDL to identify ASD from fMRI

In this section, we apply the gradient estimation in Chapter 3 and optimization technique in Chapter 4 to the MDL framework, and apply MDL on fMRI data to estimate effective connectome (EC). To deal with the long time series and noise in data, we combine the mulitple-shooting method with MDL. Furthermore, we perform classification of ASD vs control based on our estimated connectome.

6.1 Recap of Dynamic Causal Modeling

The Effective Connectome is typically estimated from the dynamical causal modeling (DCM) [43]. Suppose there are p nodes (ROIs) and denote the observed fMRI time-series signal as $s(t)$, which is a p-dimensional vector at each time t. Denote the hidden neuronal state as $z(t)$; then $z(t)$ and $s(t)$ are p-dimensional vectors for each time point t. Denote the hemodynamic response function (HRF) [87] as $h(t)$, and denote the external stimulation as $u(t)$, which is an n-dimensional vector for each t. The model is:

$$f\Big([z(t) \;\; D(t)]\Big) = \begin{bmatrix} dz(t)/dt \\ dD(t)/dt \end{bmatrix} = \begin{bmatrix} D(t)z(t) + Cu(t) \\ Bu(t) \end{bmatrix}, \;\; D(0) = A \tag{6.1}$$

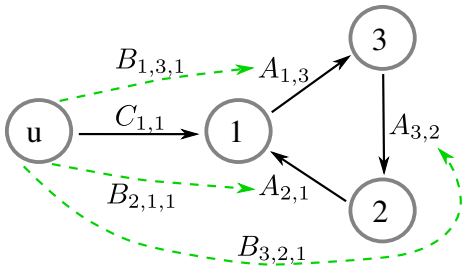


Figure 6.1: Toy example of dynamic causal modeling with 3 nodes (labeled 1 to 3). u is a 1-D stimulation signal, so $n = 1, p = 3$. A, B, C are defined as in Eq. 6.1. For simplicity, though A is a 3×3 matrix, we assume only three elements $A_{1,3}, A_{3,2}, A_{2,1}$ are non-zero.

$$s(t) = \Big(z(t) + \epsilon(t)\Big) * h(t), \quad \widetilde{z(t)} = z(t) + \epsilon(t) = Deconv\Big(s(t), h(t)\Big) \tag{6.2}$$

where $\epsilon(t)$ is the noise at time t, which is assumed to follow an independent Gaussian distribution, and $*$ represents convolution operation. $D(t)$ is a $p \times p$ matrix for each t, representing the effective connectome between nodes. A is a matrix of shape $p \times p$, representing the interaction between ROIs. B is a tensor of shape $p \times p \times n$, representing the effect of stimulation on the effective connectome. C is a matrix of shape $p \times n$, representing the effect of stimulation on neuronal state. An example of $n = 1, p = 3$ is shown in Fig. 6.1. The task is to estimate parameters A, B, C from noisy observation $s(t)$.

6.2 Overcoming long time series and noise in fMRI data with Multiple Shooting MDL (MS-MDL)

A DCM model is typically optimized using the expectation-maximization (EM) algorithm [103]. Despite its wide application and good theoretical properties, a drawback is we need to re-derive the algorithm when the forward model changes, which limits its application. Furthermore, current DCM can not handle large-scale systems, hence is unsuitable for whole-brain analysis. In this section, we propose Multiple-Shooting Adjoint (MSA), which is a generic method for parameter estimation in high-dimensional non-linear dynamical systems, and can be viewed as a special case of MDL for the continuous-time

models.

6.2.1 Notations and formulation of problem

We summarize notations here for the ease of reading, which correspond to Fig. 6.2.

- $z(t), \widetilde{z(t)}, \overline{z(t)}$: $z(t)$ is the true time-series, $\widetilde{z(t)}$ is the noisy observation, and $\overline{z(t)}$ is the estimation. If p time-series are observed, then they are p-dimensional vectors for each time t.

- $(t_i, \widehat{z}_i)_{i=0}^N$: $\{\widehat{z}_i\}_{i=0}^N$ are corresponding guesses of states at split time points $\{t_i\}_{i=0}^N$. See Fig. 6.2. $\widehat{z}_i$ are discrete points, while $\widetilde{z(t)}, z(t), \overline{z(t)}$ are trajectories.

- f_η: Hidden state $z(t)$ follows the ODE $\frac{dz}{dt} = f(z, t)$, f is parameterized by η.

- θ: $\theta = [\eta, z_0, ...z_N]$. We concatenate all optimizable parameters into one vector for the ease of notation, denoted as θ.

- $\lambda(t)$: Lagrangian multiplier in the continuous case, used to derive the adjoint state equation.

The task of DCM can be viewed as a parameter estimation problem for a continuous dynamical system, and can be formulated as:

$$\underset{\eta}{\operatorname{argmin}} \int \left(\overline{z(\tau)} - \widetilde{z(\tau)}\right)^2 d\tau \quad s.t. \ \frac{d\overline{z(\tau)}}{d\tau} = f_\eta(\overline{z(\tau)}, \tau) \tag{6.3}$$

The goal is to estimate η from observations $\widetilde{z}$. In the following sections, we first briefly introduce the multiple-shooting method, which is related to the numerical solution of a continuous dynamical system; next, we introduce the adjoint state method, which efficiently determines the gradient for parameters in continuous dynamical systems; next, we

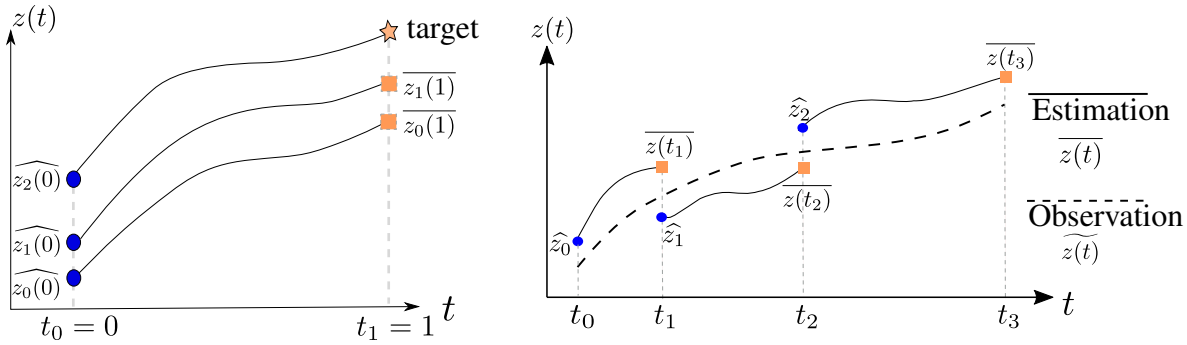


Figure 6.2: Left: illustration of the shooting method. Right: illustration of the multiple-shooting method. Blue dots represent the guess of state at split time t_i.

introduce the proposed MS-MDL method, which combines multiple-shooting and the adjoint state method, and can be applied with general forward models and gradient-based optimizers; finally, we introduce the DCM model, and demonstrate the application of MS-MDL.

6.2.2 Multiple-shooting method

The shooting method is commonly used to fit an ODE under noisy observations, which is crucial for parameter estimation in ODE. In this section, we first introduce the shooting method, then explain its variant, the multiple-shooting method, for long time-series.

Shooting method The shooting method typically reduces a boundary-value problem to an initial value problem [57]. An example is shown in Fig. 6.2: to find a correct initial condition (at $t_0 = 0$) that reaches the target (at $t_1 = 1$), the shooting algorithm first takes an initial guess (e.g. $\widehat{z_0(0)}$), then integrate the curve to reach point $(t_1, \overline{z_0(1)})$; the error term $target - z_0(1)$ is used to update the initial condition (e.g. $\widehat{z_1(0)}$) so that the end-time value $\overline{z_1(1)}$ is closer to target. This process is repeated until convergence. Besides the initial condition, the shooting method can be applied to update other parameters.

Multiple-shooting method The multiple-shooting method [13] is an extension of the shooting method to long time-series; it splits a long time-series into chunks, and applies the shooting method to each chunk. Integration of a dynamical system for a long time is typically subject to noise and numerical error, while solving short time-series is generally easier and more robust.

As shown in the right subfigure of Fig. 6.2, a guess of initial condition at time t_0 is denoted as $\widehat{z_0}$, and we can use any ODE solver to get the estimated integral curve $\overline{z(t)}, t \in [t_0, t_1]$. Similarly, we can guess the initial condition at time t_1 as $\widehat{z_1}$, and get $\overline{z(t)}, t \in [t_1, t_2]$ by integration as in Eq. 6.5. Note that each time chunk is shorter than the entire chunk ($|t_{i+1} - t_i| < |t_3 - t_0|, i \in \{1, 2\}$), hence easier to solve. The split causes another issue: the guess might not match estimation at boundary points (e.g. $\overline{z(t_1)} \neq \widehat{z_1}, \overline{z(t_2)} \neq \widehat{z_2}$). Therefore, we need to consider this error of mismatch when updating parameters, and minimizing this mismatch error is typically easier compared to directly analyzing the entire sequence.

The multiple-shooting method can be written as:

$$\underset{\eta, z_0, ...z_N}{\operatorname{argmin}} J = \underset{\eta, z_0, ...z_N}{\operatorname{argmin}} \sum_{i=0}^{N} \int_{t_i}^{t_{i+1}} \left(\overline{z(\tau)} - \widetilde{z(\tau)}\right)^2 \mathrm{d}\tau + \alpha \sum_{i=0}^{N} \left(\overline{z(t_i)} - \widehat{z_i}\right)^2 \tag{6.4}$$

$$\overline{z(t)} = \widehat{z_i} + \int_{t_i}^{t} f_\eta\big(\overline{z(\tau)}, \tau\big)\mathrm{d}\tau, \quad t_i < t < t_{i+1}, \quad i \in \{0, 1, 2, ...N\} \tag{6.5}$$

where N is the total number of chunks discretized at points $\{t_0, ...t_N\}$, with corresponding guesses $\{\widehat{z_0}, ...\widehat{z_N}\}$. We use $\overline{z(t)}$ to denote the estimated curve as in Eq. 6.5; suppose t falls into the chunk $[t_i, t_{i+1}]$, $z(t)$ is determined by solving the ODE from $(\widehat{z_i}, t_i)$, where $\widehat{z_i}$ is the guess of initial state at t_i. We use $\widetilde{z(t)}$ to denote the observation. The first part in Eq. 6.4 corresponds to the difference between estimation $\overline{z(t)}$ and observation $\widetilde{z(t)}$, while the second part corresponds to the mismatch between estimation (orange square, $\overline{z(t_i)}$)

and guess (blue circle, $\widehat{z_i}$) at split time points t_i. The second part is weighted by a hyper-parameter α. The ODE function f is parameterized by η. The optimization goal is to find the best η that minimizes loss in Eq. 6.4, besides model parameters η, we also need to optimize the guess $\widehat{z_i}$ for state at time $t_i, i \in \{0, 1, ...N\}$. Note that though previous work typically limits f to have a linear form, we don't have such limitations. Instead, multiple-shooting is generic for general f.

6.2.3 Adjoint state method

Our goal is to minimize the loss function in Eq. 6.4. Let $\theta = [\eta, z_0, ..., z_N]$ represent all learnable parameters. After fitting an ODE, we derive the gradient of loss L $w.r.t$ parameter θ and state guess $\widehat{z_i}$ for optimization.

Adjoint state equation

Note that different from discrete case, the gradient in continuous case is slightly complicated. We refer to the adjoint method [121, 180, 21]. Consider the following problem:

$$\frac{d\overline{z(t)}}{dt} = f_\theta\Big(\overline{z(t)}, t\Big), \;\; s.t. \;\; \overline{z(0)} = x, \;\; t \in [0, T], \;\; \theta = [\eta, z_0, ...z_N] \tag{6.6}$$

$$\hat{y} = \overline{z(T)}, \;\; J\Big(\hat{y}, y\Big) = J\Big(\overline{z(0)} + \int_0^T f_\theta(\overline{z}, t)dt, y\Big) \tag{6.7}$$

where the initial condition $z(0)$ is specified by input x, output $\hat{y} = \overline{z(T)}$. The loss function J is applied on $\hat{y}$, with target y. Compared with Eq. 6.3 to Eq. 6.5, for simplicity, we use θ to denote both model parameter η and guess of initial conditions $\{\widehat{z_i}\}$. The Lagrangian is

$$L = J\Big(\overline{z(T)}, y\Big) + \int_0^T \lambda(t)^\top \Big[\frac{d\overline{z(t)}}{dt} - f_\theta(\overline{z(t)}, t)\Big] dt \tag{6.8}$$

where $\lambda(t)$ is the continuous Lagrangian multiplier. Then we have the following:

$$\frac{\partial J}{\partial \overline{z(T)}} + \lambda(T) = 0 \tag{6.9}$$

$$\frac{d\lambda(t)}{dt} + \Big(\frac{\partial f_\theta(\overline{z(t)}, t)}{\partial \overline{z(t)}}\Big)^\top \lambda(t) = 0 \ \ \forall t \in (0, T) \tag{6.10}$$

$$\frac{dL}{d\theta} - \int_T^0 \lambda(t)^\top \frac{\partial f_\theta(\overline{z(t)}, t)}{\partial \theta} dt = 0 \tag{6.11}$$

We skip the proof for simplicity. In general, the adjoint method determines the initial condition $\lambda(T)$ by Eq. 6.9, then solves Eq. 6.10 to get the trajectory of $\lambda(t)$, and finally integrates $\lambda(t)$ as in Eq. 6.11 to get the final gradient. Note that Eq. 6.9 to Eq. 6.11 is generic for general θ, and in case of Eq. 6.4 and Eq. 6.5, we have $\theta = [\eta, z_0, ...z_N]$, and $\nabla\theta = [\frac{\partial L}{\partial \eta}, \frac{\partial L}{\partial z_0}, ...\frac{\partial L}{\partial z_N}]$. Note that we need to calculate $\frac{\partial f}{\partial z}$ and $\frac{\partial f}{\partial \theta}$, which can be easily computed by a single backward pass; we only need to specify the forward model without worrying about the backward, because automatic differentiation is supported in frameworks such as PyTorch and Tensorflow. After deriving the gradient of all parameters, we can update these parameters by gradient descent.

Note that though $J(\overline{z(T)}, y)$ is defined on a single time point in Eq. 6.8, it can extend to the integral form $\int_{t=0}^{T} loss(t)dt$. We can defined F as $\frac{dF(t)}{dt} = loss(t), F(0) = 0$, then $F(T)$ (for a single time point T) equals the integral.

Adaptive Checkpoint Adjoint Eq. 6.9 to Eq. 6.11 are the analytical form of the gradient in the continuous case, yet the numerical implementation is crucial for empirical performance. Note that $\overline{z(t)}$ is solved in forward-time (0 to T), while $\lambda(t)$ is solved in reverse-time (T to 0), yet the gradient in Eq. 6.11 requires both $\overline{z(t)}$ and $\lambda(t)$ in the integrand. Memorizing a continuous trajectory $\overline{z(t)}$ requires much memory; to save memory, most existing implementations forget the forward-time trajectory of $\overline{z(t)}$, and instead only

Algorithm 17: Multiple-Shooting Adjoint (MSA)

Input Observation $\widetilde{z(t)}$, number of chunks N, learning rate lr.
Initialize model parameter η, state $\{\widehat{z}_i\}_{i=0}^{N}$ at discretized points $\{t_i\}_{i=0}^{N}$
Repeat until convergence
(1) Estimate trajectory $\overline{z(t)}$ from current parameters by the multiple shooting method as in Eq. 6.5.
(2) Compute the loss J in Eq. 6.4, plug J in Eq. 6.8. Derive the gradient by ACA in Chapter 3.
(3) Update parameters with first-order gradient optimizers discussed in Chapter 4.

record the end-time state $\overline{z(T)}$ and $\lambda(T)$ and solve Eq. 6.6 and Eq. 6.9 to Eq. 6.11 in reverse-time on-the-fly.

While memory cost is low, existing implementations of the adjoint method typically suffer from numerical error: since the forward-time trajectory (denoted as $\overrightarrow{z(t)} = \overline{z(t)}$) is deleted, and the reverse-time trajectory (denoted as $\overleftarrow{z(t)}$) is reconstructed from the end-time state $z(T)$ by solving Eq. 6.6 in reverse-time, $\overrightarrow{z(t)}$ and $\overleftarrow{z(t)}$ cannot accurately overlap due to inevitable errors with numerical ODE solvers. The error $\overrightarrow{z(t)} - \overleftarrow{z(t)}$ propagates to the gradient in Eq. 6.11 in the $\frac{\partial f(z,t)}{\partial z}$ term. Please see Chapter 3 for a detailed explanation.

To solve this issue, we use the ACA (see Chapter 3) which records $\overrightarrow{z(t)}$ using a memory-efficient method to guarantee numerical accuracy. In this work, we use ACA for its accuracy.

6.2.4 Multiple-Shooting Adjoint State Method (MSA)

Procedure of MSA MSA is a combination of the multiple-shooting and MDL, which is generic for various f. Details are summarized in Algo. 17. MSA iterates over the following steps until convergence: (1) estimate the trajectory based on the current parameters, using the multiple-shoot method for integration; (2) compute the loss and derive the gradient using the adjoint method; (3) update the parameters based on the gradient.

Advantages of MSA Previous work has used the multiple-shooting method for parameter estimation in ODEs [117], yet MSA is different in the following aspects: (A) Suppose the parameters have k dimensions. MSA uses an element-wise update, hence has only $O(k)$ computational cost in each step; yet the method in [117] requires the inversion of a $k \times k$ matrix, hence might be infeasible for large-scale systems. (B) The implementation of [117] does not tackle the mismatch between forward-time and reverse-time trajectory, while we use ACA [180] for accurate gradient estimation in step (2) of Algo. 17. (C) From a practical perspective, our implementation is based on PyTorch which supports automatic-differentiation, therefore we only need to specify the forward model f without the need to manually compute the gradient $\frac{\partial f}{\partial z}$ and $\frac{\partial f}{\partial \theta}$. Hence, our method is off-the-shelf for general models, while the method of [117] needs to re-implement $\frac{\partial f}{\partial z}$ and $\frac{\partial f}{\partial \theta}$ for different f, and conventional DCM with EM needs to re-derive the entire algorithm when f changes.

6.3 Validation of MSA on toy examples

We first validate MSA on toy examples of linear dynamical systems, then validate its performance on large-scale systems and non-linear dynamical systems.

A linear dynamical system with 3 nodes We first start with a simple linear dynamical system with only 3 nodes. We further simplify the matrix A as in Fig. 6.1, where only three elements in A are non-zero. We set B as a zeros matrix, and $u(t)$ as a 1-dimensional

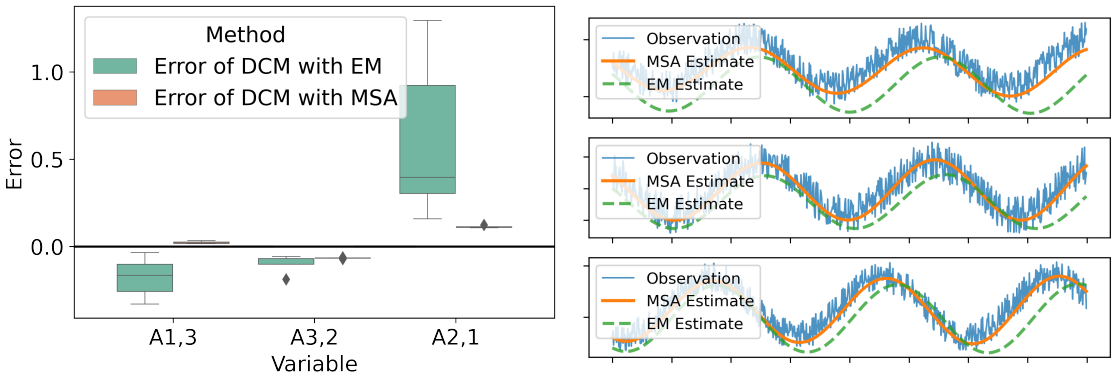


Figure 6.3: Results for the toy example of a linear dynamical system in Fig. 6.1. Left: error in estimated value of connection $A_{1,3}, A_{3,2}, A_{2,1}$, other parameters are set as 0 in simulation. Right: from top to bottom are the results for node 1, 2, 3 respectively. For each node, we plot the observation and estimated curve from MSA and EM methods. Note that the estimated curve is generated by integration of the ODE under estimated parameters with only the initial condition known, not smoothing of noisy observation.

signal. The dynamical system is linear:

$$\begin{bmatrix} dz(t)/dt \\ dD(t)/dt \end{bmatrix} = \begin{bmatrix} D(t)z(t) + Cu(t) \\ 0 \end{bmatrix}, \; D(0) = A, \;\; u(t) = \begin{cases} 1, & floor(\frac{t}{2})\%2 = 0 \\ 0, & otherwise \end{cases} \tag{6.12}$$

$$\widetilde{z(t)} = z(t) + \epsilon(t), \quad \epsilon(t) \sim N(0, \sigma^2) \tag{6.13}$$

$u(t)$ is an alternating block function at a period of 2, taking values 0 or 1. The observed function $\widetilde{z(t)}$ suffers from $i.i.d$ Gaussian noise $\epsilon(t)$ with 0 mean and uniform variance σ^2.

We perform 10 independent simulations and parameter estimations. For estimation of DCM with the EM algorithm, we use the SPM package [118], which is a widely used standard baseline. The estimation in MSA is implemented in PyTorch, using ACA [180] as the ODE solver. For MSA, we use the AdaBelief optimizer [181] to update parameters with the gradient; though other optimizers such as SGD can be used, we found AdaBelief converges faster in practice.

For each of the non-zero elements in A, we show the boxplot of error in estimation

in Fig. 6.3. Compared with EM, the error by MSA is significantly closer to 0 and has a smaller variance. An example of a noisy observation and estimated curves are shown in Fig. 6.3, and the estimation by MSA is visually closer to the ground-truth compared to the EM algorithm. We emphasize that the estimated curve is not a simple smoothing of the noisy observation; instead, after estimating the parameters of the ODE, the estimated curve (for $t > 0$) is generated by solving the ODE using only the initial state. Therefore, the match between estimated curve and observation demonstrates that our method learns the underlying dynamics of the system.

Application to large-scale systems After validation on a small system with only 3 nodes, we validate MSA on large scale systems with more nodes. We use the same linear dynamical system as in Eq. 6.12, but with the node number p ranging from 10 to 100. Note that the dimension of A and B grows at a rate of $O(p^2)$, and the EM algorithm estimates the covariance matrix of size $O(p^4)$, hence the memory for EM method grows extremely fast with p. For various settings, the ground truth parameter is randomly generated from a uniform distribution between -1 and 1, and the variance of measurement noise is set as $\sigma = 0.5$. For each setting, we perform 5 independent runs, and report the mean squared error (MSE) between estimated parameter and ground truth.

As shown in Table 6.1, for small-size systems (number of nodes $<= 20$), MSA consistently generates a lower MSE than the EM algorithm. For large-scale systems, since the memory cost of the EM algorithm is $O(p^4)$, the algorithm quickly runs out-of-memory. On the other hand, the memory cost for MSA is $O(p^2)$ because it only uses the first-order gradient. Hence, MSA is suitable for large-scale systems such as in whole-brain fMRI analysis.

Application to general non-linear systems Since neither the multiple-shoot method

nor the adjoint state method requires the ODE f to be linear, our MSA can be applied to general non-linear systems. Furthermore, since our implementation is in PyTorch which supports automatic differentiation, we only need to specify f when fitting different models, and the gradient will be calculated automatically. Therefore, MSA is an off-the-shelf method, and is suitable for general non-linear ODEs both in theory and implementation.

We validate MSA on the Lotka-Volterra (L-V) equations [160], a system of non-linear ODEs describing the dynamics of predator and prey populations. The L-V equation can be written as:

$$f\Big([z_1(t), z_2(t)]\Big) = \begin{bmatrix} dz_1(t)/dt \\ dz_2(t)/dt \end{bmatrix} = \begin{bmatrix} \zeta z_1(t) - \beta z_1(t) z_2(t) \\ \delta z_1(t) z_2(t) - \gamma z_2(t) \end{bmatrix}, \quad \begin{bmatrix} \widetilde{z_1(t)} \\ \widetilde{z_2(t)} \end{bmatrix} = \begin{bmatrix} z_1(t) + \epsilon_1(t) \\ z_2(t) + \epsilon_2(t) \end{bmatrix} \tag{6.14}$$

where $\zeta, \beta, \delta, \gamma$ are parameters to estimate, $\widetilde{z(t)}$ is the noisy observation, and $\epsilon(t)$ is the independent noise. Note that there are non-linear terms $z_1(t)z_2(t)$ in the ODE, making EM derivation difficult. Furthermore, the EM method needs to explicitly derive the posterior mean, hence needs to be re-derived for every different f; while MSA is generic and hence does not require re-derivation.

Besides the L-V model, we also consider a modified L-V model, defined as:

$$dz_1(t)/dt = \zeta z_1(t) - \beta \phi(z_2(t)) z_1(t) z_2(t) \tag{6.15}$$

$$dz_2(t)/dt = \delta \phi(z_1(t)) z_1(t) z_2(t) - \gamma z_2(t) \tag{6.16}$$

where $\phi(x) = 1/(1 + e^{-x})$ is the sigmoid function. We use this example to demonstrate the ability of MSA to fit highly non-linear ODEs.

We compare MSA with LMFIT [109], which is a well-known python package for non-linear fitting. We use L-BFGS solver in LMFIT, which generates better results than other solvers. We did not compare with original DCM with EM because it's unsuitable for

Table 6.1: Mean squared error ($\times 10^{-3}$, **lower** is better) in estimation of parameters for a linear dynamical system with different number of nodes. "OOM" represents "out of memory".

	10 Nodes	20 Nodes	50 Nodes	100 Nodes
EM	3.3 ± 0.2	3.0 ± 0.2	OOM	OOM
MSA	0.7 ± 0.1	0.9 ± 0.3	0.8 ± 0.1	0.8 ± 0.2

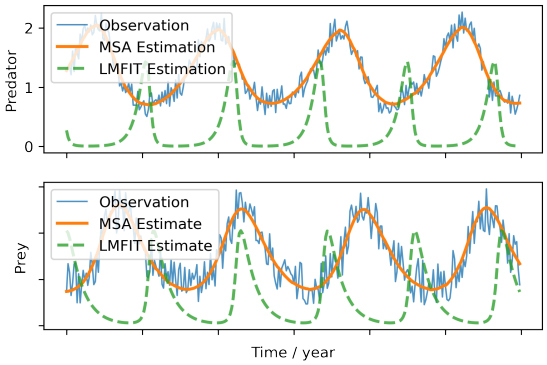


Figure 6.4: Results for the L-V model.

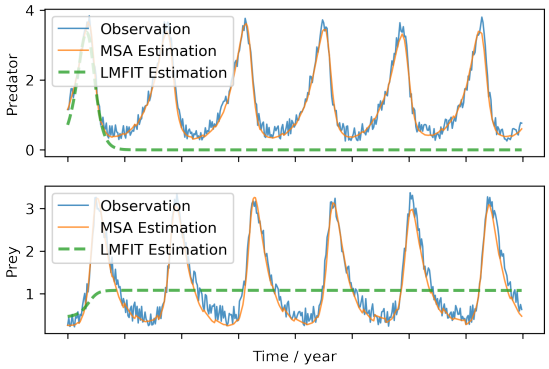


Figure 6.5: Results for the modified L-V model.

general non-linear models. The estimation of the curve for $t > 0$ is solved by integrating using the estimated parameters and initial conditions. As shown in Fig. 6.4 and Fig. 6.5, compared with LMFIT, MSA recovers the system accurately. LMFIT directly fits the long sequences, while MSA splits long-sequences into chunks for robust estimation, which may partially explain the better performance of MSA.

6.4 Apply MDL to identify ASD from fMRI data

We applied EC and FC on the classification task of ASD vs Control with both resting-state fMRI and task fMRI. Our results show that combing EC with FC achieves a consistent improvement in classification accuracy.

6.4.1 Data acquisition and pre-processing

Task-fMRI data

fMRI for 82 children with ASD and 48 age and IQ-matched healthy controls were acquired. The fMRI (BOLD, 132 volumes, TR = 2000ms, TE = 25ms, flip angle = 60°, voxel size 3.44×3.44×4 mm^3) was acquired on a Siemens MAGNETOM Trio 3T scanner.

A biological motion perception task and a scrambled motion task [68] were presented in alternating blocks (24s). fMRI data was then pre-processed with FSL with the following standard procedures: 1) motion correction, 2) interleaved slice timing correction, 3) brain extraction with BET, 3) spatial smoothing with a full-width at half-maximum (FWHM) of 5mm, 5) high-pass temporal filtering. For parcellation, we use the AAL atlas which consists of 116 ROIs.

Resting-state fMRI data

We use the ABIDE I [34] per-processed data for our experiments. We omit the data points whose length of time-series is less than 100, resulting in 301 control subjects and 264 ASD subjects. We use the same sliding window of length 30 to estimate EC and FC within each window, and adjacent windows slide by 5 frames. We choose the C-PAC [25] pre-processing pipeline consisting of the following steps: 1) structural pre-processing with AFNI and FSL, 2) slice time correction, 3) motion correction, 4) skull-strip, 5) global mean intensity normalization, 6) nuisance signal regression, 7) band-pass filtering and 8) registration to the anatomical space.

6.4.2 Improved Fitting with ACA and AdaBelief

We compare ACA vs Adjoint method in the fitting of DCM, and plot the results in Fig. 6.6. We observe that ACA consistently generates a lower training loss than the adjoint method,

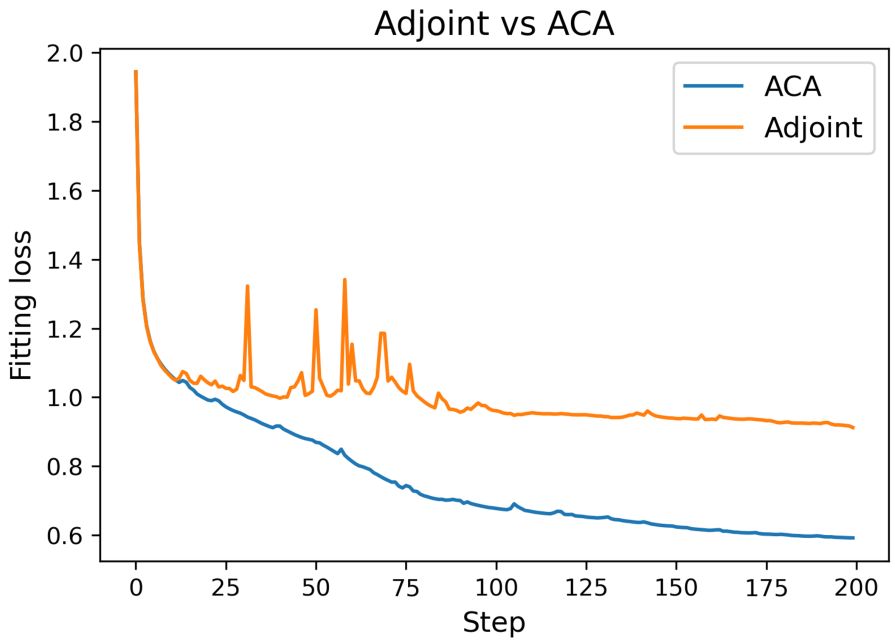


Figure 6.6: Compare the fitting loss of ACA and Adjoint method in fitting of DCM. Both curves use the AdaBelief optimizer.

validating our analysis on the numerical accuracy in gradient estimation discussed in Chapter 3.

We further compare AdaBelief with Adam in the fitting of DCM, and plot the results in Fig. 6.7. For both experiments we use the ACA method to derive the gradient accurately, and feed the gradient into different optimizers. We observe that AdaBelief achieves comparable fitting loss as Adam within half the training time.

6.4.3 Estimation of Effective Connectome and Functional Connectome

Estimation of EC

We use the AAL atlas [155] containing 116 ROIs. For each subject, the parameters for dynamic causal modeling as in Eq. 6.1 is estimated using MSA. An example snapshot of

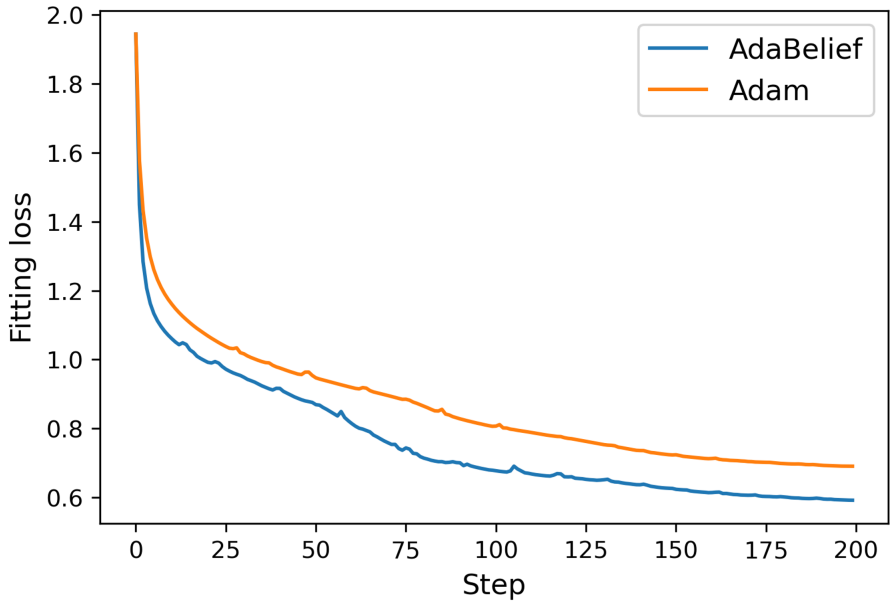


Figure 6.7: Compare the fitting loss with different optimizers in fitting of DCM. Both curves use ACA for gradient estimation.

the effective connectome (EC) during the two tasks is shown in Fig. 6.8, showing MSA captures the dynamic EC.

Estimation of FC

We estimate the Dynamic Functional Connectome (DFC) using a sliding-window with a windowsize of 20 time points, and every two adjacent windows are separated by 1 time point. We use the AAL template and calculate the Pearson Correlation among regions for each sliding window.

6.4.4 Group comparison

We perform a group comparison between the ASD and control groups, and plot the edges with a p-value smaller than 0.05 (with Bonferroni correction). We plot the results for FC in Fig. 6.10, and plot the results for EC in Fig. 6.11. Despite the difference, both FC and EC edges include regions in the frontal lobe and the temporal lobe, which match previous

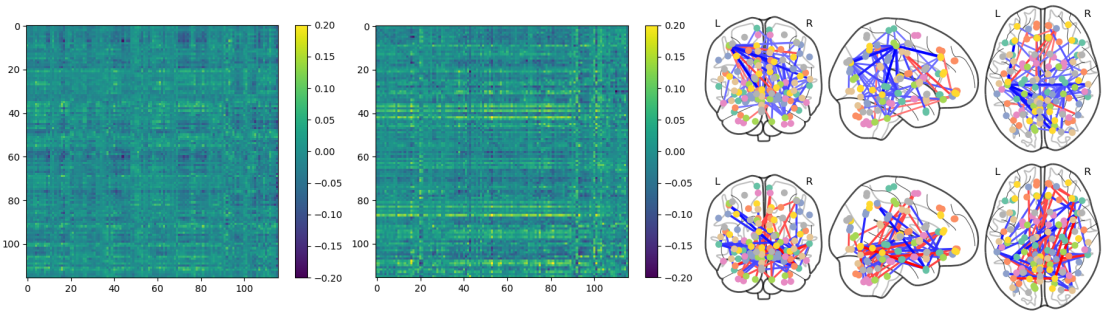


Figure 6.8: An example of MSA for one subject in task fMRI. Left: effective connectome during task 1. Middle: effective connectome during task 2. Right: top and bottom represents the effective connectome for task 1 and 2 respectively. Blue and red edges represent positive and negative connections respectively. Only top 5% strongest connections are visualized.

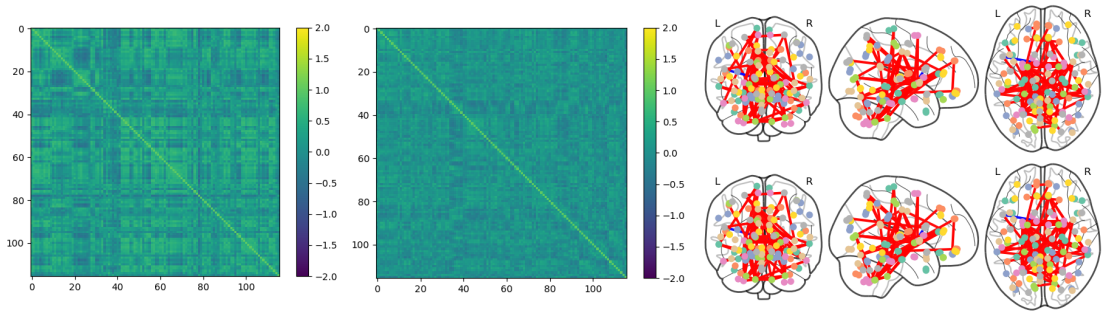


Figure 6.9: An example of Dynamic Functional Connectome for one subject in task fMRI. Left: functional connectome during task 1. Middle: functional connectome during task 2. Right: top and bottom represents the functional connectome for task 1 and 2 respectively. Blue and red edges represent positive and negative connections respectively. Only top 5% strongest connections are visualized.

findings in the literature on ASD.

6.4.5 Classification results for task fMRI

We conduct classification experiments for ASD vs. control using EC, FC and EC-FC concatenated together as the predictor. The classification of a subject is based on the majority vote of the predictions across all time points. We experimented with a InvNet [179] of 20 layers with a feature dimension of 32. Results for a 10-fold subject-wise cross validation are shown in Fig. 6.12. For classification using task-fMRI data, using EC generates slightly better accuracy, F1 score, AUC and Precision than using FC as the

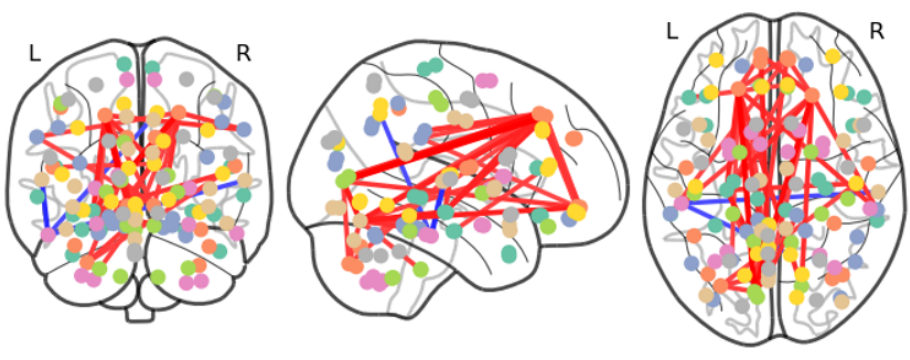


Figure 6.10: FC edges that are significantly different between ASD and control groups.

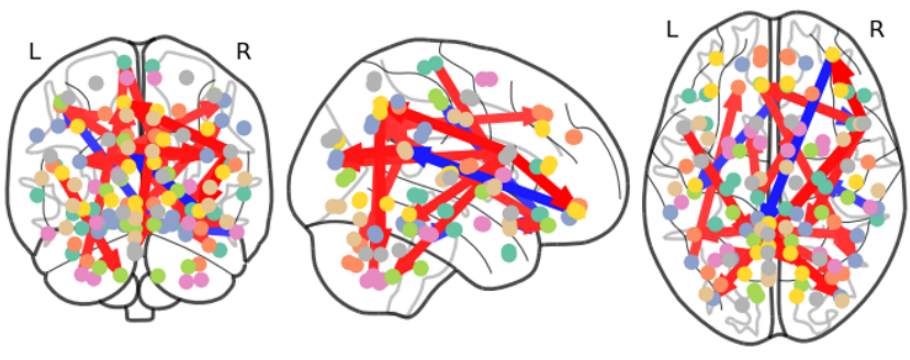


Figure 6.11: EC edges that are significantly different between ASD and control groups.

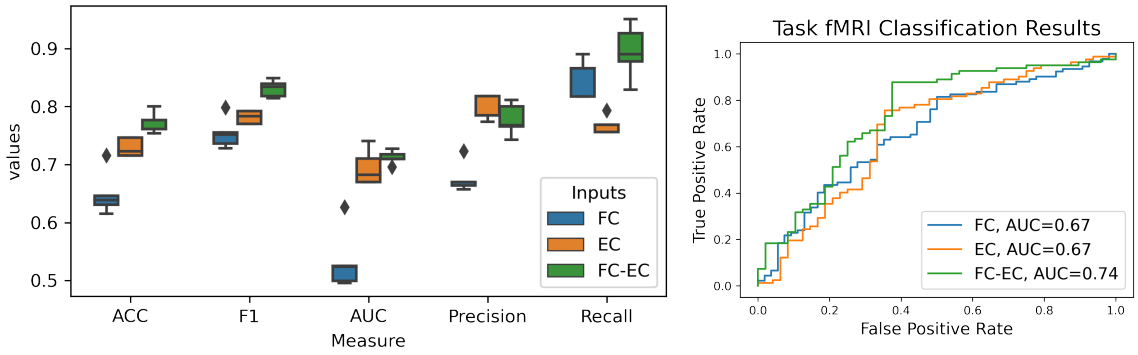


Figure 6.12: Classification results on task fMRI data

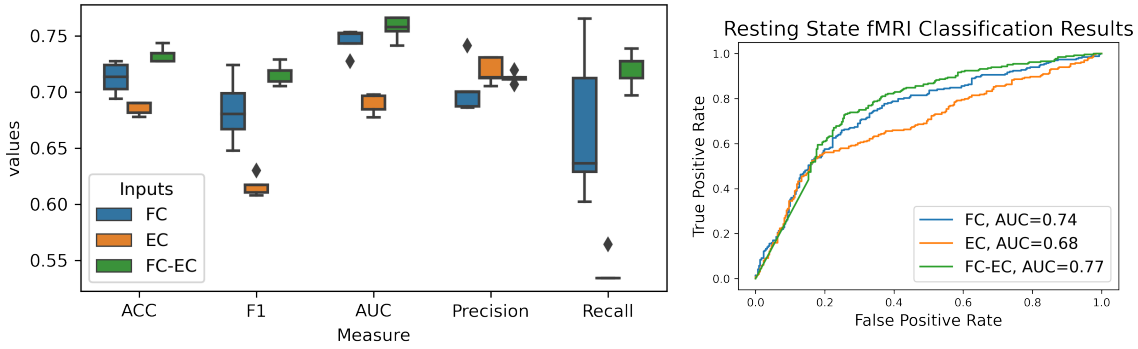


Figure 6.13: Classification results on resting-state fMRI data

predictor; we also notice that using EC-FC as input consistently improves the classification performance.

6.4.6 Classification results fo resting-state fMRI

For resting-state fMRI data, We use the ABIDE I [34] per-processed data for our experiments. We omit the data points whose length of time-series is less than 100, resulting in 301 control subjects and 264 ASD subjects. We use the same sliding window of length 20 to estimate EC and FC within each window, and choose 30 frames of EC and 30 frames of FC for each subject. We use the same neural network for classification as in task-fMRI experiments.

We report the results for a 10-fold cross validation in Fig. 6.13. We found that for

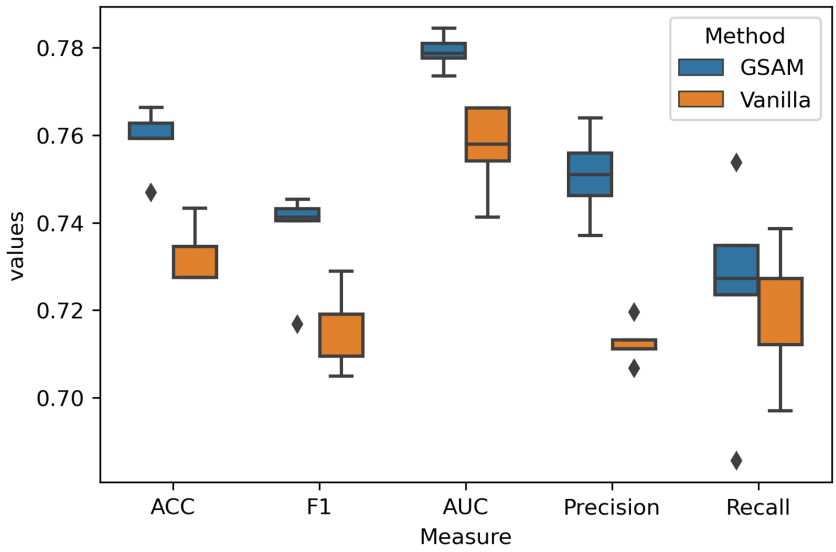


Figure 6.14: Classification results on task fMRI data, using EC and FC as input.

resting-state fMRI data, FC appears to be more predictive than EC. This could be caused by that when the subject is not performing task during resting-state fMRI scan, the external stimulation is random and can not be modeled in DCM, therefore estimation of EC is inaccurate. By combining EC and FC together as the input, we observe an improvement in classification performance.

6.4.7 Improved classification with GSAM

We further apply the GSAM method proposed in Chapter 5 to the classification task described above, and show the results for task fMRI and resting-state fMRI in Fig. 6.14 and Fig. 6.15 respectively. We observe consistent improvement with GSAM. We further plot the dominant eigenvalue of the Hessian of a trained model in Fig. 6.16 and Fig. 6.17 for task and resting-state fMRI respectively, and validate that the proposed GSAM method reduces the sharpness (dominant eigenvalue of Hessian of the prediction function).

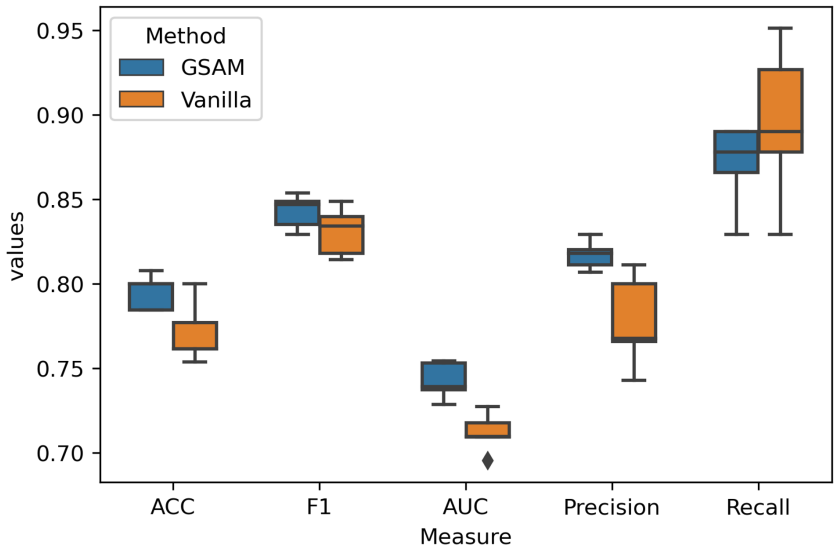


Figure 6.15: Classification results on resting-state fMRI data, using EC and FC as input.

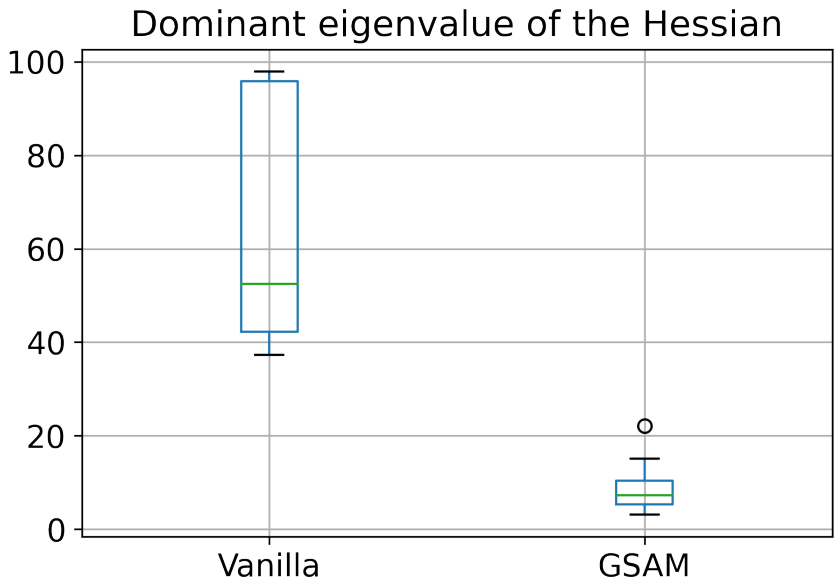


Figure 6.16: Dominant eigenvalue of the Hessian of a trained network across 10-fold cross validation for task fMRI data.

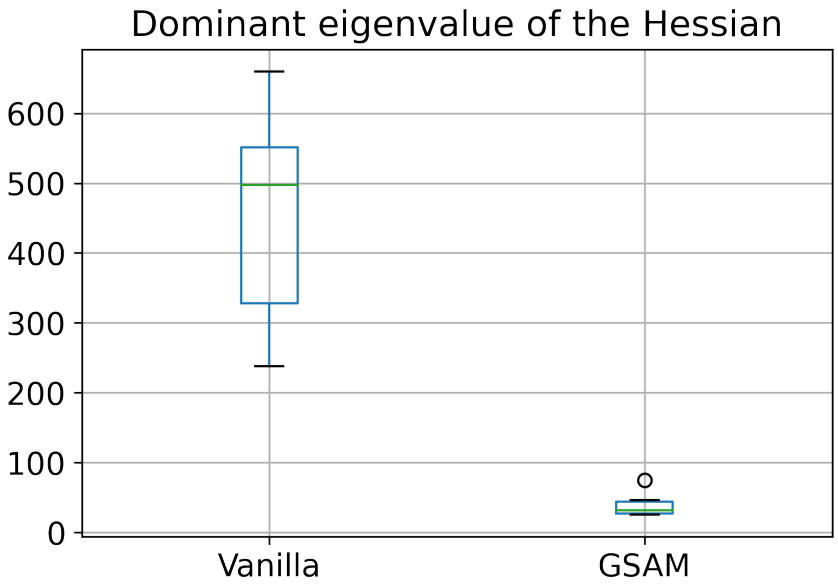


Figure 6.17: Dominant eigenvalue of the Hessian of a trained network across 10-fold cross validation for resting-state fMRI data.

Table 6.2: Classification results on task-fMRI data using FC with different window sizes.

	Window	ACC	F1	AUC
FC	10	65.4	71.2	50.5
	20	62.3	71.9	53.8
	30	61.5	69.1	53.3
	40	63.8	69.1	54.2
EC		68.9	74.0	68.8

Table 6.3: Classification results for GSAM with different α values.

	alpha	ACC	F1	AUC
GSAM	0	76.9	83.5	76
	0.1	78.5	82.9	75.6
	0.3	79.2	83.8	74.7
	0.5	77.7	83.4	72.2
Vanilla		73.3	71.6	75.7

6.4.8 Studies on hyper-parameters

We conduct extra experiments to study the influence of hyper-parameters. Specifically, we report the classification results using dynamic FC with different window sizes in Table. 6.2. For different sliding window sizes, every two adjacent windows are moved by 1 time frame. As Table. Table. 6.2. shows, using a window size of 10 generates the best performance; however, the performance of EC outperforms FC by a large margin.

We conduct extra experiments with different α values and report them in Table. 6.3. For all experiments, we use both FC and EC as the input. We observe that $\alpha = 0.3$ achieves the best result, and even the worst result with $\alpha = 0$ still outperforms results with vanilla optimizers. The results further validate that GSAM significantly reduces the sharpness of neural network and improves the identification of ASD.

Chapter 7

Conclusions

In this thesis, we aim to identify ASD based on fMRI data. fMRI data is typically analyzed by the (dynamic) Functional Connectome (FC), which is defined as the correlation between time-series from different ROIs. However, FC is a descriptive model and does not capture the underlying causal relations among brain regions. Effective Connectome (EC) is estimated from the Dynamic Causal Modeling, and captures the causal relation among regions by modeling the brain as a dynamical system. However, due to the difficulty for parameter estimation in dynamical systems, EC is typically limited to small-scale system ($<$10 nodes), which hinders its application in the whole-brain network analysis. Therefore, in this thesis, we develop the Model-Driven Learning Framework (MDL), which is a generic framework for parameter estimation in high-dimensional continuous-time models, and can be used to efficiently fit DCM and derive EC.

We first solve the gradient estimation for continuous-time models in Chapter 3. We identify that the numerical errors in forward-time and reverse-time trajectories cause the error in gradient estimation for dynamical systems. Based on above observation, we propose the Adaptive Checkpoint Adjoint (ACA) method to avoid the numerical errors. We empirically validate that ACA generates a significantly lower fitting loss than existing methods.

In Chapter 4, we propose AdaBelief, an adaptive first-order gradient optimizer that achieves fast convergence and training stability. We empirically validate that AdaBelief fits DCM 2x faster than other optimizers such as Adam.

In Chapter 5, we propose GSAM, which jointly minimizes the training loss and the curvature of the loss landscape, hence GSAM enables both a high training accuracy and generalization performance on test set.

In Chapter 6, we apply the methods developed in previous chapters to fMRI data. Specifically, we use the gradient estimation method in Chapter 3 and the optimizer in Chapter 4 to efficiently fit DCM to fMRI data and estimate EC. Next, we use EC and FC as the input to classify ASD vs control. We observe that the combination of EC and FC improves the classification performance. We then apply GSAM in Chapter 5 to further improve the classification performance, and identify that GSAM reduces curvature of the loss surface and improves generalization in the ASD vs control classification task.

In summary, we propose a series of methods that combine into the MDL method, which is a generic framework for parameter estimation in high-dimensional non-linear dynamical systems. MDL is an off-the-shelf method. We apply MDL to identify ASD based on fMRI data as the main application of our method.

Bibliography

[1] Federica Agosta, Michela Pievani, Cristina Geroldi, Massimiliano Copetti, Giovanni B Frisoni, and Massimo Filippi. Resting state fmri in alzheimer's disease: beyond the default mode network. *Neurobiology of aging*, 33(8):1564–1578, 2012.

[2] Neculai Archip, Olivier Clatz, Stephen Whalen, Dan Kacher, Andriy Fedorov, Andriy Kot, Nikos Chrisochoides, Ferenc Jolesz, Alexandra Golby, Peter M Black, et al. Non-rigid alignment of pre-operative mri, fmri, and dt-mri with intra-operative mri for enhanced visualization and navigation in image-guided neurosurgery. *Neuroimage*, 35(2):609–624, 2007.

[3] Yossi Arjevani, Yair Carmon, John C Duchi, Dylan J Foster, Nathan Srebro, and Blake Woodworth. Lower bounds for non-convex stochastic optimization. *arXiv preprint arXiv:1912.02365*, 2019.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[5] Lukas Balles and Philipp Hennig. Dissecting adam: The sign, magnitude and variance of stochastic gradients. *arXiv preprint arXiv:1705.07774*, 2017.

[6] Deanna M Barch, Gregory C Burgess, Michael P Harms, Steven E Petersen, Bradley L Schlaggar, Maurizio Corbetta, Matthew F Glasser, Sandra Curtiss, Sachin

Dixit, Cindy Feldt, et al. Function in the human connectome: task-fmri and individual differences in behavior. *Neuroimage*, 80:169–189, 2013.

[7] Evelyn ML Beale. On minimizing a convex function subject to linear inequalities. *Journal of the Royal Statistical Society: Series B (Methodological)*, 17(2):173–184, 1955.

[8] Jens Behrmann, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In *International Conference on Machine Learning*, pp. 573–582, 2019.

[9] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. signsgd: Compressed optimisation for non-convex problems. *arXiv preprint arXiv:1802.04434*, 2018.

[10] Jeremy Bernstein, Arash Vahdat, Yisong Yue, and Ming-Yu Liu. On the distance between two neural networks and the stability of learning. *arXiv preprint arXiv:2002.03432*, 2020.

[11] MA Bertocci, GM Bebko, BC Mullin, SA Langenecker, CD Ladouceur, JRC Almeida, and Mary L Phillips. Abnormal anterior cingulate cortical activity during emotional n-back task performance distinguishes bipolar from unipolar depressed females. *Psychological medicine*, 42(7):1417–1428, 2012.

[12] Lucas Beyer, Olivier J. Henaff, Alexander Kolesnikov, Xiaohua Zhai, and Aaron van den Oord. Are we done with imagenet? *arXiv preprint arXiv:2002.05709*, 2020.

[13] Hans Georg Bock and Karl-Josef Plitt. A multiple shooting algorithm for direct solution of optimal control problems. *IFAC Proceedings Volumes*, 1984.

[14] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pp. 177–186. Springer, 2010.

[15] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*. 2012.

[16] Edward T Bullmore and Danielle S Bassett. Brain graphs: graphical models of the human brain connectome. *Annual review of clinical psychology*, 7:113–140, 2011.

[17] Richard B Buxton. *Introduction to functional magnetic resonance imaging: principles and techniques*. Cambridge university press, 2009.

[18] Jeff R Cash and Alan H Karp. A variable order runge-kutta method for initial value problems with rapidly varying right-hand sides. *ACM Transactions on Mathematical Software (TOMS)*, 16(3):201–222, 1990.

[19] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124018, 2019.

[20] Jinghui Chen, Dongruo Zhou, Yiqi Tang, Ziyan Yang, Yuan Cao, and Quanquan Gu. Closing the generalization gap of adaptive gradient methods in training deep neural networks. In *IJCAI*, 2020.

[21] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 2018.

[22] Ricky TQ Chen, Jens Behrmann, David K Duvenaud, and Jörn-Henrik Jacobsen.

Residual flows for invertible generative modeling. In *Advances in Neural Information Processing Systems*, pp. 9916–9926, 2019.

[23] Xiangning Chen, Cho-Jui Hsieh, and Boqing Gong. When vision transformers outperform resnets without pretraining or strong data augmentations, 2021.

[24] Xiangyi Chen, Sijia Liu, Ruoyu Sun, and Mingyi Hong. On the convergence of a class of adam-type algorithms for non-convex optimization. *arXiv preprint arXiv:1808.02941*, 2018.

[25] Cameron Craddock, Sharad Sikka, Brian Cheung, Ranjeet Khanuja, Satrajit S Ghosh, Chaogan Yan, Qingyang Li, Daniel Lurie, Joshua Vogelstein, Randal Burns, et al. Towards automated analysis of connectomes: The configurable pipeline for the analysis of connectomes (c-pac). *Front Neuroinform*, 42:10–3389, 2013.

[26] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

[27] Alex Damian, Tengyu Ma, and Jason Lee. Label noise sgd provably prefers flat global minimizers. *arXiv preprint arXiv:2106.06530*, 2021.

[28] Talgat Daulbaev, Alexandr Katrutsa, Larisa Markeeva, Julia Gusak, Andrzej Cichocki, and Ivan Oseledets. Interpolated adjoint method for neural odes. *arXiv preprint arXiv:2003.05271*, 2020.

[29] Alfredo M Ozorio De Almeida. *Hamiltonian systems: chaos and quantization*. Cambridge University Press, 1990.

[30] Omar Dekhil, Hassan Hajjdiab, Ahmed Shalaby, Mohamed T Ali, Babajide Ayinde, Andy Switala, Aliaa Elshamekh, Mohamed Ghazal, Robert Keynton, Gregory

Barnes, et al. Using resting state functional mri to build a personalized autism diagnosis system. *PloS one*, 13(10):e0206351, 2018.

[31] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

[32] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[33] Adriana Di Martino, Kathryn Ross, Lucina Q Uddin, Andrew B Sklar, F Xavier Castellanos, and Michael P Milham. Functional brain correlates of social and nonsocial processes in autism spectrum disorders: an activation likelihood estimation meta-analysis. *Biological psychiatry*, 65(1):63–74, 2009.

[34] Adriana Di Martino, David O'connor, Bosi Chen, Kaat Alaerts, Jeffrey S Anderson, et al. Enhancing studies of the connectome in autism using the autism brain imaging data exchange ii. *Scientific data*, 2017.

[35] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[36] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[37] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

[38] Eugene P Duff, William Vennart, Richard G Wise, Matthew A Howard, Richard E Harris, Michael Lee, Karolina Wartolowska, Vishvarani Wanigasekera, Frederick J Wilson, Mark Whitlock, et al. Learning to identify cns drug action and efficacy using multistudy fmri data. *Science translational medicine*, 7(274):274ra16–274ra16, 2015.

[39] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. In *Advances in Neural Information Processing Systems*, pp. 3140–3150, 2019.

[40] Nicha C Dvornek, Pamela Ventola, Kevin A Pelphrey, and James S Duncan. Identifying autism from resting-state fmri using long short-term memory networks. In *International Workshop on Machine Learning in Medical Imaging*, pp. 362–370. Springer, 2017.

[41] Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. How to train your neural ode: the world of jacobian and kinetic regularization. In *International Conference on Machine Learning*, 2020.

[42] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*, 2020.

[43] Karl J Friston and Lee Harrison. Dynamic causal modelling. *Neuroimage*, 2003.

[44] Xavier Gastaldi. Shake-shake regularization. *arXiv preprint arXiv:1705.07485*, 2017.

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

[46] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[47] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[48] Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

[49] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[50] Abigail S Greene, Siyuan Gao, Dustin Scheinost, and R Todd Constable. Task-induced brain state manipulation improves prediction of individual traits. *Nature communications*, 9(1):1–13, 2018.

[51] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2017.

[52] YAN Hanshu, DU Jiawei, TAN Vincent, and FENG Jiashi. On robustness of neural ordinary differential equations. In *International Conference on Learning Representations*, 2019.

[53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[54] Byeongho Heo, Sanghyuk Chun, Seong Joon Oh, Dongyoon Han, Sangdoo Yun, Gyuwan Kim, Youngjung Uh, and Jung-Woo Ha. Adamp: Slowing down the

slowdown for momentum optimizers on scale-invariant weights. *arXiv preprint arXiv:2006.08217*, 2020.

[55] Volker Hesselmann, Bettina Sorger, Ralf Girnus, Kathrin Lasek, Mohammad Maarouf, Christoph Wedekind, Jürgen Bunke, Oliver Schulte, Barbara Krug, Klaus Lackner, et al. Intraoperative functional mri as a new approach to monitor deep brain stimulation in parkinson's disease. *European radiology*, 14(4):686–690, 2004.

[56] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pp. 6626–6637, 2017.

[57] Francis Begnaud Hildebrand. *Introduction to numerical analysis*. 1987.

[58] Geoffrey Hinton. Rmsprop: Divide the gradient by a running average of its recent magnitude. *Coursera*, 2012.

[59] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *arXiv preprint arXiv:1902.00275*, 2019.

[60] Sepp Hochreiter and Jürgen Schmidhuber. Simplifying neural nets by discovering flat minima. In *Advances in neural information processing systems*, pp. 529–536, 1995.

[61] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[62] Scott A Huettel, Allen W Song, Gregory McCarthy, et al. *Functional magnetic resonance imaging*, volume 1. Sinauer Associates Sunderland, MA, 2004.

[63] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[64] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *arXiv preprint arXiv:1806.07572*, 2018.

[65] Junteng Jia and Austin R Benson. Neural jump stochastic differential equations. In *Advances in Neural Information Processing Systems*, pp. 9847–9858, 2019.

[66] Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. *arXiv preprint arXiv:1912.02178*, 2019.

[67] Alexia Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard gan. *arXiv preprint arXiv:1807.00734*, 2018.

[68] Martha D Kaiser, Caitlin M Hudac, Sarah Shultz, Su Mei Lee, Celeste Cheung, et al. Neural signatures of autism. *PNAS*, 2010.

[69] Nitish Shirish Keskar and Richard Socher. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*, 2017.

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

[71] Patrick Kidger, Ricky T. Q. Chen, and Terry Lyons. "Hey, that's not an ODE": Faster ODE Adjoints with 12 Lines of Code. *arXiv:2009.09457*, 2020.

[72] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *arXiv preprint arXiv:2005.08926*, 2020.

[73] Stefan J Kiebel, Marta I Garrido, Rosalyn J Moran, and Karl J Friston. Dynamic causal modelling for eeg and meg. *Cognitive neurodynamics*, 2008.

[74] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[75] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pp. 10215–10224, 2018.

[76] Brian Knutson, Andrew Westdorp, Erica Kaiser, and Daniel Hommer. Fmri visualization of brain activity during a monetary incentive delay task. *Neuroimage*, 12 (1):20–27, 2000.

[77] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[78] Jungmin Kwon, Jeongseop Kim, Hyunseo Park, and In Kwon Choi. Asam: Adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks. *arXiv preprint arXiv:2102.11600*, 2021.

[79] Beatrice Laurent and Pascal Massart. Adaptive estimation of a quadratic functional by model selection. *Annals of Statistics*, pp. 1302–1338, 2000.

[80] Chiang-shan Ray Li, Cong Huang, R Todd Constable, and Rajita Sinha. Imaging response inhibition in a stop-signal task: neural correlates independent of signal monitoring and post-response processing. *Journal of Neuroscience*, 26(1):186–192, 2006.

[81] Xiaoyu Li and Francesco Orabona. On the convergence of stochastic gradient descent with adaptive stepsizes. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 983–992. PMLR, 2019.

[82] Xuechen Li, Ting-Kam Leonard Wong, Ricky TQ Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. *arXiv preprint arXiv:2001.01328*, 2020.

[83] Yuanzhi Li, Colin Wei, and Tengyu Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks. *arXiv preprint arXiv:1907.04595*, 2019.

[84] Tengyuan Liang, Tomaso Poggio, Alexander Rakhlin, and James Stokes. Fisher-rao metric, geometry, and complexity of neural networks. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 888–896. PMLR, 2019.

[85] Tao Lin, Lingjing Kong, Sebastian Stich, and Martin Jaggi. Extrapolation for large-batch training in deep learning. In *International Conference on Machine Learning*, pp. 6094–6104. PMLR, 2020.

[86] Ernest Lindelöf. Sur l'application de la méthode des approximations successives aux équations différentielles ordinaires du premier ordre. *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, 116(3):454–457, 1894.

[87] Martin A Lindquist, Ji Meng Loh, Lauren Y Atlas, and Tor D Wager. Modeling

the hemodynamic response function in fmri: efficiency, bias and mis-modeling. *Neuroimage*, 45, 2009.

[88] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[89] Catherine Lord, Mayada Elsabbagh, Gillian Baird, and Jeremy Veenstra-Vanderweele. Autism spectrum disorder. *The Lancet*, 392(10146):508–520, 2018.

[90] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[91] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning*, pp. 3276–3285. PMLR, 2018.

[92] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. *arXiv preprint arXiv:1902.09843*, 2019.

[93] Kaifeng Lyu and Jian Li. Gradient descent maximizes the margin of homogeneous neural networks. *arXiv preprint arXiv:1906.05890*, 2019.

[94] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes. *arXiv preprint arXiv:2002.08071*, 2020.

[95] Paul M Matthews, Garry D Honey, and Edward T Bullmore. Applications of fmri in translational medicine and clinical practice. *Nature Reviews Neuroscience*, 7(9): 732–744, 2006.

[96] David McAllester. Simplified pac-bayesian margin bounds. In *Learning theory and Kernel machines*, pp. 203–215. Springer, 2003.

[97] David A McAllester. Pac-bayesian model averaging. In *Proceedings of the twelfth annual conference on Computational learning theory*, pp. 164–170, 1999.

[98] H Brendan McMahan and Matthew Streeter. Adaptive bound optimization for online convex optimization. *arXiv preprint arXiv:1002.4908*, 2010.

[99] RV Mises and Hilda Pollaczek-Geiringer. Praktische verfahren der gleichungsauflösung. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, 9(1):58–77, 1929.

[100] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[101] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[102] Oury Monchi, John G Taylor, and Alain Dagher. A neural model of working memory processes in normal subjects, parkinson's disease and schizophrenia for fmri design and predictions. *Neural Networks*, 13(8-9):953–973, 2000.

[103] Todd K Moon. The expectation-maximization algorithm. *ISPM*, 1996.

[104] Rafael Müller, Simon Kornblith, and Geoffrey Hinton. When does label smoothing help? *arXiv preprint arXiv:1906.02629*, 2019.

[105] Ulrich Mutze. An asynchronous leapfrog method ii. *arXiv preprint arXiv:1311.6602*, 2013.

[106] Shi Naichen, Li Dawei, Hong Mingyi, and Sun Ruoyu. Rmsprop can converge with proper hyper-parameter. *ICLR*, 2021.

[107] Kate Nation, Paula Clarke, Barry Wright, and Christine Williams. Patterns of reading ability in children with autism spectrum disorder. *J Autism Dev Disord*, 2006.

[108] Yu Nesterov. A method of solving a convex programming problem with convergence rate o(1/k$\hat{2}$). In *Sov. Math. Dokl*, volume 27, 1983.

[109] Matthew Newville, Till Stensitzki, Daniel B Allen, Michal Rawlik, Antonino Ingargiola, and Andrew Nelson. Lmfit: Non-linear least-square minimization and curve-fitting for python. 2016.

[110] Behnam Neyshabur, Srinadh Bhojanapalli, and Nathan Srebro. A pac-bayesian approach to spectrally-normalized margin bounds for neural networks. *arXiv preprint arXiv:1707.09564*, 2017.

[111] Jitse Niesen et al. On the global error of discretization methods for ordinary differential equations. In *Citeseer*, 2004.

[112] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pp. 722–729. IEEE, 2008.

[113] Hirosi Okamura. Condition nécessaire et suffisante remplie par les équations différentielles ordinaires sans points de peano. *Mem. Coll. Sci., Kyoto Imperial Univ., Series A*, 24:21–28, 1942.

[114] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pp. 3498–3505. IEEE, 2012.

[115] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training

recurrent neural networks. In *International conference on machine learning*, pp. 1310–1318, 2013.

[116] Jane S Paulsen, Janice L Zimbelman, Sean C Hinton, Douglas R Langbehn, Catherine L Leveroni, Michelle L Benjamin, Norman C Reynolds, and Stephen M Rao. fmri biomarker of early neuronal dysfunction in presymptomatic huntington's disease. *American Journal of Neuroradiology*, 25(10):1715–1721, 2004.

[117] M Peifer and J Timmer. Parameter estimation in ordinary differential equations for biochemical processes using the method of multiple shooting. 2007.

[118] William D Penny, Karl J Friston, John T Ashburner, Stefan J Kiebel, and Thomas E Nichols. *Statistical parametric mapping: the analysis of functional brain images*. 2011.

[119] Mark Plitt, Kelly Anne Barnes, and Alex Martin. Functional connectivity classification of autism identifies highly predictive brain features but falls short of biomarker standards. *NeuroImage: Clinical*, 7:359–366, 2015.

[120] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[121] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. 2018.

[122] Maria Giulia Preti, Thomas AW Bolton, and Dimitri Van De Ville. The dynamic functional connectome: State-of-the-art and perspectives. *Neuroimage*, 160:41–54, 2017.

[123] Alessio Quaglino, Marco Gallieri, Jonathan Masci, and Jan Koutník. Snode:

Spectral discretization of neural odes for system identification. *arXiv preprint arXiv:1906.07038*, 2019.

[124] Alejandro F Queiruga, N Benjamin Erichson, Dane Taylor, and Michael W Mahoney. Continuous-in-depth neural networks. *arXiv preprint arXiv:2008.02389*, 2020.

[125] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[126] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? In *International Conference on Machine Learning*, pp. 5389–5400, 2019.

[127] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[128] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015.

[129] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

[130] HoHo Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 1960.

[131] Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, pp. 5320–5330, 2019.

[132] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[133] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[134] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[135] Carl Runge. Über die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46(2):167–178, 1895.

[136] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[137] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pp. 1–13, 2019.

[138] Lars Ruthotto, Stanley J Osher, Wuchen Li, Levon Nurbekyan, and Samy Wu Fung. A machine learning framework for solving high-dimensional mean field game and mean field control problems. *Proceedings of the National Academy of Sciences*, 117(17):9183–9193, 2020.

[139] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pp. 2234–2242, 2016.

[140] Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia. Hamiltonian graph networks with ode integrators. *arXiv preprint arXiv:1909.12790*, 2019.

[141] Pedro Savarese, David McAllester, Sudarshan Babu, and Michael Maire. Domain-independent dominance of adaptive methods. *arXiv preprint arXiv:1912.01823*, 2019.

[142] Mohamed L Seghier, Peter Zeidman, Alex P Leff, and Cathy Price. Identifying abnormal connectivity in patients using dynamic causal modelling of fmri responses. *Front. Neurosci*, 2010.

[143] Naichen Shi, Dawei Li, Mingyi Hong, and Ruoyu Sun. {RMS}prop can converge with proper hyper-parameter. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=3UDSdyIcBDA`.

[144] John R Silvester. Determinants of block matrices. *The Mathematical Gazette*, 84 (501):460–467, 2000.

[145] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[146] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[147] Endre Süli and David F Mayers. *An introduction to numerical analysis*. Cambridge university press, 2003.

[148] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the impor-

tance of initialization and momentum in deep learning. In *International conference on machine learning*, pp. 1139–1147, 2013.

[149] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[150] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[151] Paulo Tabuada and Bahman Gharesifard. Universal approximation power of deep neural networks via nonlinear control theory. *arXiv preprint arXiv:2007.06007*, 2020.

[152] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[153] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, et al. Mlp-mixer: An all-mlp architecture for vision. *arXiv preprint arXiv:2105.01601*, 2021.

[154] Marc Toussaint. Lecture notes: Some notes on gradient descent. 2012.

[155] Nathalie Tzourio-Mazoyer, Brigitte Landeau, Dimitri Papathanassiou, Fabrice Crivello, Olivier Etard, et al. Automated anatomical labeling of activations in spm using a macroscopic anatomical parcellation of the mni mri single-subject brain. *Neuroimage*, 2002.

[156] Martijn P Van Den Heuvel and Hilleke E Hulshoff Pol. Exploring the brain network: a review on resting-state fmri functional connectivity. *Eur Neuropsychopharmacol*, 2010.

[157] David C Van Essen, Kamil Ugurbil, Edward Auerbach, Deanna Barch, Timothy EJ Behrens, Richard Bucholz, Acer Chang, Liyong Chen, Maurizio Corbetta, Sandra W Curtiss, et al. The human connectome project: a data acquisition perspective. *Neuroimage*, 62(4):2222–2231, 2012.

[158] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30:5998–6008, 2017.

[159] Loup Verlet. Computer" experiments" on classical fluids. i. Thermodynamical properties of Lennard-Jones molecules. *Physical review*, 159(1):98, 1967.

[160] Vito Volterra. Variations and fluctuations of the number of individuals in animal species living together. *ICES Journal of Marine Science*, 3, 1928.

[161] Gerhard Wanner and Ernst Hairer. *Solving ordinary differential equations II*. Springer Berlin Heidelberg, 1996.

[162] Antoine Wehenkel and Gilles Louppe. Unconstrained monotonic neural networks. In *Advances in Neural Information Processing Systems*, pp. 1545–1555, 2019.

[163] Colin Wei, Jason Lee, Qiang Liu, and Tengyu Ma. Regularization matters: Generalization and optimization of neural nets vs their induced kernel. 2019.

[164] E Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

[165] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pp. 4148–4158, 2017.

[166] Zeke Xie, Li Yuan, Zhanxing Zhu, and Masashi Sugiyama. Positive-negative momentum: Manipulating stochastic gradient noise to improve generalization. *arXiv preprint arXiv:2103.17182*, 2021.

[167] Haruo Yoshida. Construction of higher order symplectic integrators. *Physics letters A*, 150(5-7):262–268, 1990.

[168] Xubo Yue, Maher Nouiehed, and Raed Al Kontar. Salr: Sharpness-aware learning rates for improved generalization. *arXiv preprint arXiv:2011.05348*, 2020.

[169] Manzil Zaheer, Sashank Reddi, Devendra Sachan, Satyen Kale, and Sanjiv Kumar. Adaptive methods for nonconvex optimization. In *Advances in neural information processing systems*, pp. 9793–9803, 2018.

[170] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[171] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. 2017.

[172] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *International conference on machine learning*, pp. 7354–7363. PMLR, 2019.

[173] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[174] Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. Lookahead optimizer: k steps forward, 1 step back. In *Advances in Neural Information Processing Systems*, pp. 9593–9604, 2019.

[175] Yaowei Zheng, Richong Zhang, and Yongyi Mao. Regularizing neural networks via adversarial model perturbation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8156–8165, 2021.

[176] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ode-net: Learning hamiltonian dynamics with control. *arXiv preprint arXiv:1909.12077*, 2019.

[177] Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven Hoi, et al. Towards theoretically understanding why sgd generalizes better than adam in deep learning. *arXiv preprint arXiv:2010.05627*, 2020.

[178] Zhiming Zhou, Qingru Zhang, Guansong Lu, Hongwei Wang, Weinan Zhang, and Yong Yu. Adashift: Decorrelation and convergence of adaptive learning rate methods. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HkgTkhRcKQ`.

[179] Juntang Zhuang, Nicha C Dvornek, Xiaoxiao Li, Pamela Ventola, and James S Duncan. Invertible network for classification and biomarker selection for asd. In *MICCAI*, 2019.

[180] Juntang Zhuang, Nicha Dvornek, Xiaoxiao Li, Sekhar Tatikonda, Xenophon Papademetris, and James Duncan. Adaptive checkpoint adjoint for gradient estimation in neural ode. *ICML*, 2020.

[181] Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar C Tatikonda, Nicha Dvornek,

Xenophon Papademetris, and James Duncan. Adabelief optimizer: Adapting step-sizes by the belief in observed gradients. *NeurIPS*, 2020.

[182] Fangyu Zou, Li Shen, Zequn Jie, Weizhong Zhang, and Wei Liu. A sufficient condition for convergences of adam and rmsprop. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11127–11135, 2019.